TM-LPSAT: Encoding Temporal Metric Planning in Continuous Time

by

Ji-Ae Shin

A dissertation submitted in partial fulfillment

of the requirement for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

May 2004

Ernest Davis

Dedication

To my parents and my brother Young-Woo

Acknowledgements

First of all, I would like to thank Professor Ernest Davis for making this happen.

I would like to thank the readers from my thesis committee: Professor Ernest Davis put me under the SAT-based planning; Professor Drew McDermott at Yale provided valuable suggestions and comments for the reorganization of my thesis, on which the final version is mostly based. I am also glad to have had Professor Alan Siegel and Professor Benjamin Goldberg in my thesis committee.

Special thanks should go to Steven Wolfman and Professor Daniel Weld at the University of Washington for making their LPSAT program available for my research. I would particularly like to thank Steve for always giving me prompt responses and helpful comments to questions on the LPSAT. In addition, I would like to thank Dr. Jonathan Amsterdam for whose classes I was a teaching assistant and through which I learnt all aspects of teaching.

Lastly, I would not have been able to make it through all those years without support from my family and friends, particularly my niece Jin-A, Chan-Sook Lim at USC for listening to me during times of great stress and being my systems consultant, Dr. Julian O'Rear for encouraging me to complete my degree, and my best friends from KAIST and ETRI.

Abstract

In any domain with change, the dimension of time is inherently involved. Whether the domain should be modeled in discrete time or continuous time depends on aspects of the domain to be modeled. Many complex real-world domains involve continuous time, resources, metric quantities and concurrent actions. Planning in such domains must necessarily go beyond simple discrete models of time and change.

In this thesis, we show how the SAT-based planning framework can be extended to generate plans of concurrent asynchronous actions that may depend on or make change piecewise linear metric constraints in continuous time.

In the SAT-based planning framework, a planning problem is formulated as a satisfiability problem of a set of propositional constraints (axioms) such that any model of the axioms corresponds to a valid plan. There are two parameters to a SAT-based planning system: an encoding scheme for representing plans of bounded length and a propositional SAT solver to search for a model. The LPSAT architecture is composed of a SAT solver integrated with a linear arithmetic constraint solver in order to deal with metric aspects of domains.

We present encoding schemes for temporal models of continuous time defined in PDDL+: (i) Durative actions with discrete and/or continuous changes; (ii) Real-time temporal model with exogenous events and autonomous processes capturing continuous changes. The encoding represents, in a CNF formula over arithmetic constraints and propositional fluents, time-stamped parallel plans possibly with concurrent continuous and/or discrete changes. In addition, we present encoding schemes for multi-capacity resources, partitioned interval resources, and metric

quantities which are represented as intervals. An interval type can be used as a parameter to action as well as a fluent type.

Based on the LPSAT engine, the TM-LPSAT temporal metric planner has been implemented: Given a PDDL+ representation of a planning problem, the compiler of TM-LPSAT translates it in a CNF formula, which is fed into the LPSAT engine to find a solution corresponding to a plan for the planning problem. We also have experimented on our temporal metric encodings with other decision procedure, MathSAT, which deals with propositional combinations of linear constraints and Boolean variables. The results show that in terms of searching time the SAT-based approach to temporal metric planning can be comparable to other planning approaches and there is plenty of room to push further the limits of the SAT-based approach.

Table of Contents

List of Figures

List of Tables

List of Appendices

Chapter 1 Introduction

Automated Planning is the identification of sequences of actions which will achieve specified goals from specified initial conditions.

First of all, we identify the class of domains that we are trying to solve and give an overview of the SAT-based planning framework our planner is based on and the TM-LPSAT temporal metric planner.

1.1 Problem Statement

Many complex domains in the real world involve continuous and metric time, metric resources, metric quantities, and concurrent actions. Here are two sample domains of the class we are trying to model and to generate plans for:

Bathtub Domain

More than one bathtub is in a bathroom. Each bathtub has more than one tap. The hot water taps are distinguished from the cold water taps. Each tap has a different flow rate. The process of filling a bathtub by a water tap is by turning it on. The process of draining a bathtub is triggered by pulling out the drain plug. It is also possible to add a certain amount of bath oil before the water in the bathtub reaches a certain level. When the bathtub overflows, the floor becomes wet, triggering a signal that alarms the plan executor if the planner is not in the bathroom.

A planning problem from this domain could be to generate a sequence of actions that would maintain the water temperature of the bathtub within a certain range (by adjusting the ratio of hot and cold water flowing from the taps).

The domain has the following characteristics:

- The water level of the bathtub is a continuously changing quantity *over time*, as long as a tap is on or draining occurs.
- Cumulative operations on the water level of the bathtub are possible: the "filling" process by multiple taps as well as the "draining" process.
- The planner can choose "turn-on-a-tap", "turn-off-a-tap", "plug-in", "plug-out" or "add-bath-oil."
- The planner cannot control the "filling" process that happens as a result of turning on a tap; this continues as long as the tap is on. Likewise, the planner cannot control the "draining" process that happens as a result of plugging out; this continues as long as the plug remains off.
- "Overflow" of water in the bathtub is triggered by the conditions occurring in the bathtub, not by choice of the planner.

Satellite Observation Domain

This is a simplified satellite observation scheduling domain of the Space Project at NASA. More than one satellite in orbit is available for observation. Each satellite is equipped with various pieces of observation equipment. The observations requested require particular instruments at certain times, and involve slewing the satellite to align

the instrument with the target. This takes varying amounts of time depending on the current position of the satellite. Instruments to be used must be powered on and warmed up for some time, and then calibrated. Thus, the use of instruments consumes fuel. The data observed is recorded in the on-board storage, which has limited capacity. Each observation will last either until a requested amount of data is collected or for a fixed duration. A satellite can communicate with any given ground station, as long as the station is within the orbit of the satellite. Communication with the ground station is expensive. During the down-linking, the station can download the data stored, which can occur simultaneously with collection of data by the satellite.

A possible planning problem may be scheduling the order of observation requests such that the amount of data collected is maximized[1] and the time taken to do so is minimized.

The domain has the following characteristics:

- The “angle” of the satellite is a metric quantity.
- The “on-board-storage” is a metric resource on which “collecting” activity and “downloading” activity can interplay continuously.
- The “observation” or “down-linking” activity has a duration which is determined by either the amount of collected data or the given time limit.

Planning in such domains must necessarily go beyond simple discrete models of time and change. However, the classical planning framework is founded on the assumption that time is atomic (discrete steps). A natural question is how to extend the classical

planning framework so as to reason about temporal and metric aspects of the dynamically changing world. Specifically, it was claimed that the SAT-based planning framework to continuous time would not be feasible[2] [Smith00-b] [LongFox00].

In this thesis, we attempt to extend the SAT-based planning framework to deal with metric resources and quantities in continuous time and to show that the SAT-based planning framework is feasible for representing and reasoning in continuous time.

1.2 SAT-Based Propositional Planning

Historically, in the early 1970s, planning was cast as a first-order deductive theorem-proving [Green69]. A planning problem could be formulated with axioms about actions stating that (i) the effects of an action are implied by the occurrence of the action when its preconditions hold, and (ii) frame axioms describing the propositions an action does not affect. The planning process could then be viewed as finding a deductive proof of a statement asserting that the initial conditions together with a sequence of actions imply the goal conditions.

However, this approach failed to scale up to realistically sized problems. Until the early 1990s, the idea of *Planning as Propositional Satisfiability* was not even considered to be applicable to planning in practice. Through experiments with SATPLAN in which new SAT algorithms (GSAT [Kautz92], WSAT [Selman94]) based on local searches were used as SAT solvers, Kautz and Selman [Kautz92] [Kautz96]

[1] The TM-LPSAT cannot optimize plans; this may be turned around by specifying the limits in the goal state.

showed that (i) a general propositional theorem prover outperforms traditional planning systems such as UCPOP [Penberthy92] and Nonlin[McAllester91], (ii) the propositional theorem proving scales much better than the first-order theorem proving, and (iii) different axiomatizations (encoding schemes) can have vastly different computational properties.

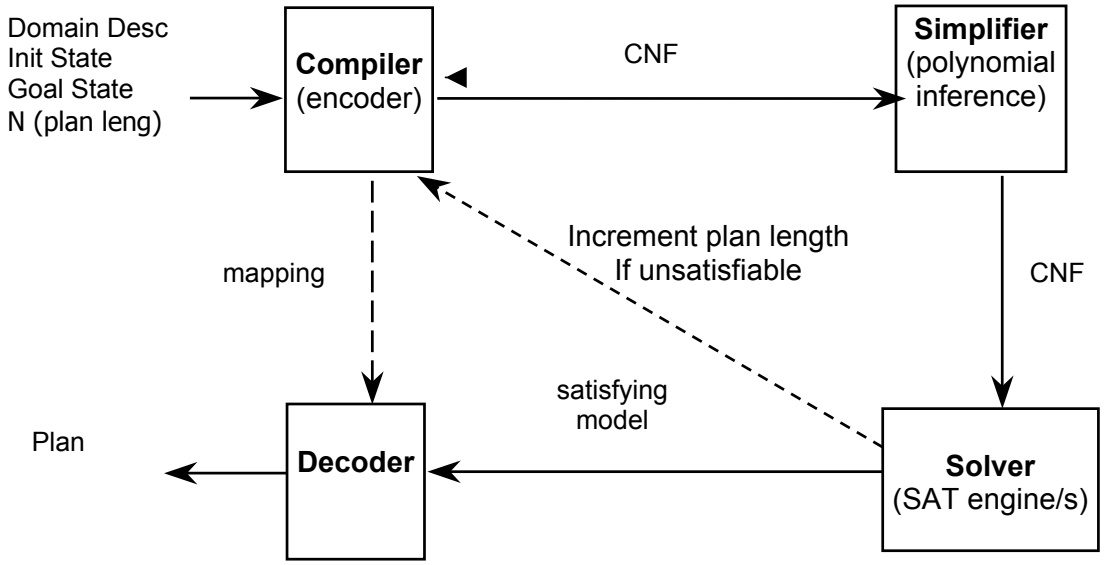

Figure 1: Architecture of the SAT-Based Planning Framework

The idea of planning as a propositional satisfiability is to reduce a planning problem to a propositional satisfiability problem. A planning problem is formulated as a set of axioms (action axioms, frame axioms, and exclusion axioms) with the property that *any* model of the axioms corresponds to a valid plan. The planning process is to

[2] Further discussion and our resolution to this issue are presented in Chapter 8.

find a model of the set of axioms, which corresponds to a correct plan for the original planning problem.

Thus, there are two parameters to SAT-based planning systems: an encoding scheme for representing plans and a SAT solver to search for models. Although there is room to optimize state-of-the-art SAT solvers specifically for planning domains [Giunchiglia98] [Mali02-b], a general SAT solver works well for planning domains. The key issue in implementing planning as satisfiability is how to encode the set of constraints.

Programs for solving propositional satisfiability have become increasingly powerful over the last decade [Malik02] [Zhang02] [Gent99]. Consequently, SAT solvers are being used as the basis for an ever-increasing array of applications of many different kinds [Armando02]. The SAT-based planning is one such highly successful application. The performances of the best SAT-based planners, such as SATPLAN [Kautz96], BlackBox [KautzSelman99] and MEDIC [Ernst97], were shown to be competitive to Graphplan-based planners or better, and both outperform partial order based planners. The main disadvantage of this approach is that the encoding size can be large, because all possible propositions and actions are represented explicitly for each step. Otimization techniques[3] to reduce the size of encoding as well as dufferent encoding methods have been extenstively explored.

1.3 SAT-Based Metric Planning

Generally, a logic-based framework needs some other tool to cope with metric aspects of the world in an efficient as well as an intuitive way. Very recently, state-of-the-art SAT solvers have been successfully integrated with domain-specific procedures so that they overcome the limit in expressiveness of propositional logic.

[3] Optimization techniques are overviewed in Section 7.2 and Section 7.3.

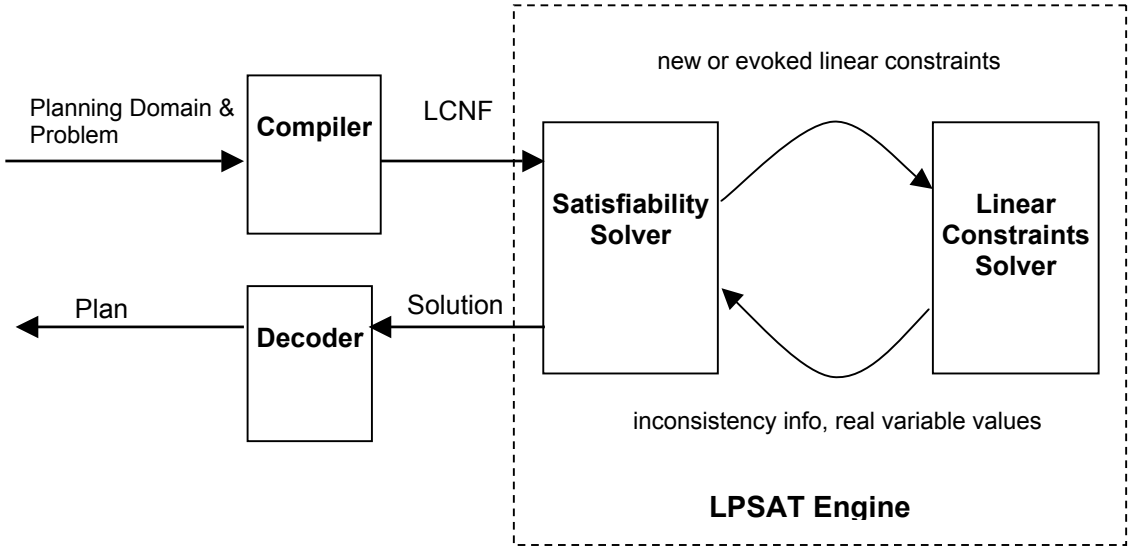

Figure 2: Architecture of the LPSAT Planning Framework

In the LPSAT framework, a SAT solver and an arithmetic constraints solver are integrated into one framework by extending a DPLL-based[4] SAT solver slightly, in such a way that the SAT solver treats each arithmetic constraint as a Boolean variable (called a *trigger*) and tests whether the generated truth assignment is satisfiable. A truth assignment satisfying all clauses propositionally can be a model for the input formulas only if the arithmetic constraints whose triggers are set to true are consistent. Thus, in the propositional reasoning part, heuristics and optimization techniques developed for SAT solvers can be applied. Considering that the computational time requirements of the integrated engine are dominated by the time requirements of the

[4] Incomplete (Stochastic or Randomized) solvers also would be possible for domains in which completeness is unnecessary: The ILP-PLAN [KautzWalser00] uses an integer local search algorithm (a variant of WSAT) for metric planning.

arithmetic constraint solver, optimization techniques [5] for reducing calls to the constraint are essential to improve the performance of the engine.

Those decision procedures have been used in various contexts requiring expressiveness beyond propositional logic, including metric planning [Wolfman00], verification of hybrid systems [Audemard03], reasoning in modal and description logics [Giunchiglia00], temporal reasoning [Armando99], and formal verification of timed systems [Audemard02-a].

Wolfman and Weld developed the LPSAT engine [Wolfman99] [Wolfman00] that is used in our TM-LPSAT and applied it to metric planning in discrete time. The encoding scheme of discrete changes on numeric-valued fluents adopted in TM-LPSAT is similar[6] to their encoding. They also experimented on their metric encoding with known heuristics of systematic SAT solvers, such as learning, back-jumping and random restarts. Their encoding is based on the assumptions that all actions are atomic (in contrast with durative) and all changes in numeric quantities are discrete at each step.

Example: The encoding of 'turn-to-target' operation in Satellite Domain by the LPSAT

Turn(?target)

Precondition:	Pointing(?direction)
	fuel $\geq$ angle(?direction,?target) * ConsumptionRate
Effect:	$\neg$pointing(?direction)
	Pointing(?target)
	fuel -= angle(?direction,?target) * ConsumptionRate

where, ?direction is the direction the satellite is pointing to now, ?target is the direction the satellite is moved to point to.

[5] Optimization techniques further discussed in Section 7.3.

[6] In LPSAT planner, in order to reduce calls to the LP solver, among the linear constraints generated to represent direct influence, those that are in conflict are encoded to be avoided in truth assignment by making them exclusive-OR. However, to preserve completeness in TM-LPSAT, we did not adapt this technique. See Axioms (3), (4) and the footnote in Section 4.2.3.

The encoding generated by the LPSAT is like:

$$\text{Turn}(T7,i) \Rightarrow \text{Pointing}(T1,i) \wedge \text{Constraint1}(i) \wedge \neg\text{Pointing}(T1,i+1) \wedge \text{Pointing}(T7,i+1) \wedge \text{Constraint2}(i+1)$$

$$\text{Constraint1}(i) \Rightarrow \text{fuel}(i) \geq \text{angle}(T1,T7) * \text{ConsumptionRate}$$

$$\text{Constraint2}(i+1) \Rightarrow \text{fuel}(i+1) = \text{fuel}(i) - \text{angle}(T1,T7) * \text{ConsumptionRate}$$

where, T7 for ?target, T1 for ?direction; Turn(T7,i) a Boolean variable representing that Turn(T7) is active or not at step I; Pointing(T1,i) a Boolean variable representing Pointing(T1) at step i is true or false; Constraint1(i), Constraint2(i+1) the triggers (Boolean variables) for the corresponding constraints.

The first axiom on Turn(T7,i) is solved by SAT engine. The axioms on constraints are solved by LP constraints solver when the corresponding triggers are true in the truth assignment made by the SAT solver.

1.4 SAT-Based Temporal Metric Planning: TM-LPSAT

The TM-LPSAT is a SAT-based temporal metric planner. It generates plans that contain concurrent asynchronous actions that may depends on arithmetic constraints and cause either discrete or continuous changes in numeric-valued fluents.

The TM-LPSAT compiler accepts a description of a domain and a problem in an extended version[7] of PDDL+, and translates it into a CNF formula (called LCNF) over Boolean variables and linear arithmetic constraints. The models of the CNF formula correspond to plans of bounded length for the given planning problem.

PDDL+ is the latest extension of McDermott's original PDDL (Planning Domain Description Language) [McDermott98] [McDermott00] developed for the International

[7] The details of the extension are discussed in Section 4.1.1.

Planning Competition. PDDL+ supports metric temporal domains. Three temporal models of continuous time are supported:

- Discretised durative actions in which all changes can occur only at the end points of the actions: continuous change is abstracted at end points, and so the value accessible by other action while the action occurs is not guaranteed to be correct.
- Continuous durative actions in which continuous changes can occur over the period of the actions
- The real-time temporal model that contains autonomous processes capturing continuous changes, exogenous events, and instantaneous actions

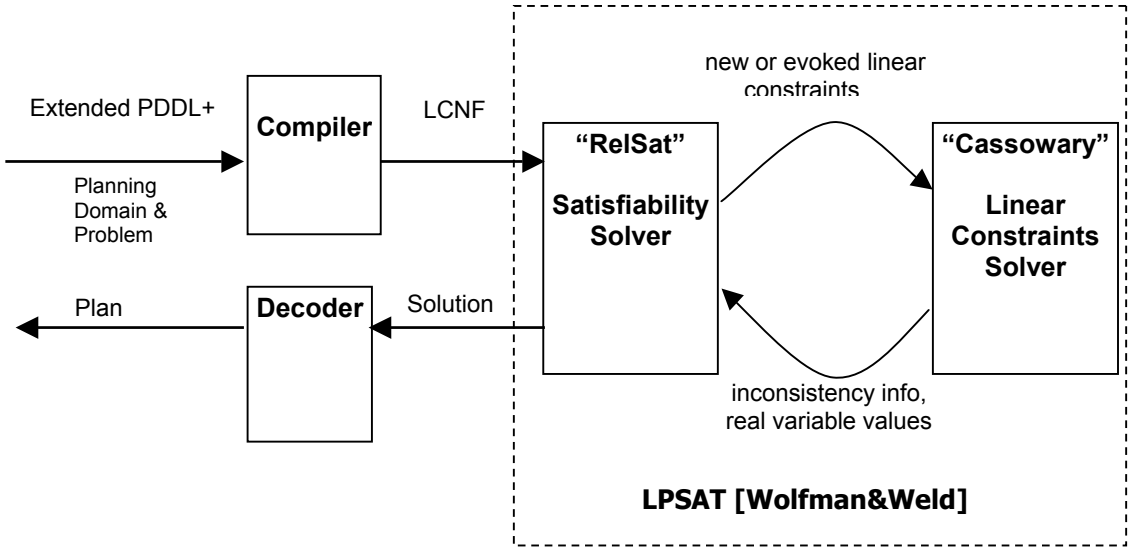

Figure 3: Architecture of the TM-LPSAT Planning Framework

We have developed encoding schemes for temporal metric plans from a planning problem described in an extended version of PDDL+. Specifically, our encodings include the following features:

- The temporal metric models supported in PDDL+ with the restrictions that all continuous changes are piecewise linear and that arithmetic constraints in the precondition are linear
- The ADL subset of PDDL+, accommodating such features as typing, negative preconditions, disjunctive preconditions, equality, quantified preconditions, and conditional effects
- Intervals as a fluent type, supported with Allen's 13 interval relations and operations for updating the interval values
- A real-valued or interval-valued parameter to action
- Sharable, reusable resources: Multi-capacity resources (e.g. identical machines in a factory; number of web designers), partitioned interval resources of which concurrent "uses" mean disjoint subintervals (e.g. main memory space allocated to concurrent processes in OS using variable-size partitions)

However, the plan metric among PDDL+ features cannot be dealt with in the TM-LPSAT.

The output of the compiler is fed into the LPSAT engine by Wolfman and Weld [Wolfman99] to generate a plan for the original planning problem.

As far as we know, there is neither any other domain-independent SAT-based temporal planning system that can reason with durative actions nor any other SAT-based temporal metric planning systems that can reason about continuous changes.

1.5 Organization of Thesis

Chapter 2 gives an overview of PDDL+ adopted as a domain specification language in TM-LPSAT. The three temporal models of continuous time supported in PDDL+ are overviewed in terms of syntax, informal semantics, and expressiveness.

Chapter 3 surveys the previous work in temporal metric planning systems that deal with durative actions or continuous changes, and gives a brief overview of formalisms developed in other areas for representing and reasoning about discrete and continuous changes, including hybrid real-time systems, qualitative processes, and logical formalisms.

Chapter 4 presents our encodings of temporal, metric planning for the three temporal models as defined in PDDL+, and also encoding schemes of multi-capacity resources, partitioned interval resources, and metric quantities of interval fluent type.

Chapter 5 describes the implementation of TM-LPSAT temporal metric planner: the LCNF compiler and its integration with Wolfman and Weld's LPSAT engine.

Chapter 6 presents the experimental results of our encodings with different decision procedure, MathSAT along with the LPSAT. The decision procedure finds solutions to propositional combinations over Boolean variables and linear arithmetic constraints.

Chapter 7 describes possible directions of extending TM-LPSAT. In particular, techniques of optimizing the encoding sizes as well as the engine are discussed further.

Chapter 8 discusses the contribution of our work and the limitations of the TM-LPSAT in terms of our encodings, as well as the approach itself.

Appendices include notational conventions, time-labeling convention, encoding of sample domains and non-interference rules of PDDL+.

Chapter 2 PDDL+ (Planning Domain Definition Language)

PDDL+ [FoxLong01] [FoxLong03] [FoxLong02-b] is the latest[8] extension of PDDL, that was intended to support temporal and metric domains. It is a declarative planning domain specification language that is based on McDermott's original PDDL [McDermott98] [McDermott00] and his maxim "physics, not advice".

PDDL+ is comprised of five levels: Level 1 contains the propositional and ADL levels of McDermott's PDDL; Level 2 adds features for numeric variables; Levels 3 and 4 contain durative actions [FoxLong02-b] [FoxLong03]; and Level 5 contains processes and exogenous events to represent real-time continuous and discrete domains [FoxLong01]. Levels 1 to 4 are collectively called PDDL2.1, officially approved by the IPC[9] committee. Level 5 is a proposal[10] by Fox and Long on a real-time temporal model that has not been approved by the committee.

The main features extended in PDDL+ include numeric-valued fluents, three temporal models of continuous time, and plan metrics. These features are reviewed in this section.

2.1 Numeric-Valued Fluents

[8] The newer release is PDDL2.2 [IPC04] for the upcoming 4th International Planning Competition.
[9] It stands for "International Planning Competition."
[10] [McDermott03-a] is an alternative proposal for the temporal model of autonomous processes, whose semantics are based on continuous branching time.

In PDDL+, a numeric-valued fluent can be represented by a function applied to arguments[11]. In the previous versions of PDDL, it was represented by a real-valued variable. Considering that numeric-valued fluents are used mainly in arithmetic computation, this functional expression form is more natural than variables (in which all intermediate calculations must be assigned either to other variables or to local variables). It supports arithmetic updating operations (*increase*, *decrease*, *assign*, etc.) and arithmetic comparison operators (=,<,<=,>,>=) between numeric functional expressions. The feature of local variables defined in an action definition and a numeric parameter to action are removed in PDDL+. We will return to this issue in Section 4.1.1.

2.2 Temporal Models of Continuous Time

A number of issues are common to any model of continuous time, whether for planning or for reasoning: (i) representation and reasoning about continuous change, (ii) concurrency, and (iii) the divided instant problem. In this section, we review the temporal models supported in PDDL+ in terms of how these issues are realized.

2.2.1 Representation of Continuous Change

There are three ways to model a continuous change in PDDL+: discretised durative actions, continuous durative actions and autonomous processes. Each model is discussed in terms of syntactic structure, (informal) semantics and expressiveness.

[11] The argument is restricted to the objects (terms) that are not numeric.

2.2.1.1 Discretised Durative Actions (Level 3)

Syntax

```
(:durative-action       <NameOfAction>
 :parameters            <ListOfArgumentsWithTypes>
 :duration              <LogicalExprOnDurationVariable>
 :condition             <LogicalExpr>
 :effect                <LogicalExpr>
)
```

<ListOfArgumentsWithTypes> is a list of variables declared along with types. <LogicalExprOnDurationVariable> is a propositional combination of numeric constraints on the duration of the action. A numeric constraint may be inequality. <LogicalExpr> is a propositional combination of fluents and arithmetic constraints, in which each proposition is temporally annotated.

The modeling of temporal relationships is done by temporally annotated conditions and effects: the temporal annotators are *at start*, *over all*, and *at end*. A duration constraint can be temporally annotated either by *at start* (by default) or *at end*.

Semantics

A condition and effect annotated by *at start* corresponds to an instantaneous action that occurs at the starting point of the durative action. Conditions annotated by *over all* correspond to invariant conditions that are required to hold over the duration of the action, but not at the starting or ending time point[12]. Conditions and effects annotated by *at end* correspond to an instantaneous action that occurs at the ending point of the durative action.

Note the expressive power of representing constraints on duration: the constraints can be (i) static (constant expression), (ii) dynamic (variable expression), and (iii) uncertain (inequality relation). The duration inequality allows the planner to adapt the duration accordingly in order to exploit concurrent activity.

Example 2-1: Bath Domain in Level 3

This is a durative action that fills a bathtub by turning on and off the tap without overflowing over [?start, ?end].

```
(:durative-action          fillBath
 :parameters    (?b – bath )
 :duration      (at end  (<=  ?duration   (/  (- (capacity  ?b) (level  ?b)) (flow ?b))))   (0)
 :condition     (and
                  (at start  (plug_in  ?b))  (at start  (not (tap_on  ?b)))                 (1)
                  (over all  (plug_in  ?b))  (over all  (tap_on  ?b))                       (2)
                  (at end (tap_on  ?b))  (at end    (plug_in ?b)))                          (3)
 :effect        (and
                  (at start  (tap_on ?b))                                                   (4)
                  (at end  (not (tap_on ?b)))                                               (5)
                  (at end  (increase  (level ?b)  (*  ?duration  (flow ?b))))               (6)
 )
```

Observe that

- (0) is a constraint on the duration: it ensures that the bath never overflows by checking, as a precondition for the updating in (6), that the quantity of water to be added does not exceed capacity. It allows this action to be concurrent with any other actions affecting the level of water during the period of the action.
- (1) and (4) are conditions and effects for the starting action of the durative action "fillBath," i.e. turn on the tap.

[12] [LongFox01] explains the reasons for adopting this temporal model and compares it with other temporal models.

- (2) are invariants to hold in the interval (?start, ?end), excluding ?start and ?end.
- (3), (5) and (6) are conditions and effects for the ending action of "fillBath," i.e. turning off the tap.
- Notice that in Level 3, during the period of the action, the "correct" value of the water level cannot be accessible by other actions[13] whose preconditions are dependent on the value. On the other hand, in Level 4 it is possible since the level of water is continuously updated. See the example in Section 2.2.1.2.

Limitations of Expressiveness

In this model, a continuous change is abstracted as a discrete change at the end points of a durative action. The value assigned to the numeric fluent while the action is occurring is not reliable. If some other concurrent action has a precondition that depends on this fluent, then the action may be not considered feasible when it is feasible or vice versa.

In terms of utilizing resources, this model adopts a conservative view: consumption of a resource is abstracted at the starting point of the durative action; production of a resource is abstracted at the ending point of the durative action. In consequence, in terms of utilization of the resources, this model cannot make the most of it. For example, a durative action *DA* that occurs over the time interval $[T_1,T_2]$ produces a resource *r* by Q_A at T_2 (actually continuously by RateProducedByDA); a durative action *DB* needs (i.e. in the precondition) the resource *r* by Q_B at T_3 such that

[13] An action like "add-bubble" that has a precondition that the level of water is more than half.

$T_1 < T_3 < T_2$ and $Q_B <= (T_3 - T_1)$ * RateProducedByDA. By taking abstraction, action *DB* cannot occur at T_3.

Obviously, there is a class of domains that can not be correctly modeled with this kind of abstraction. Such an example is data down-linking and recording in the Satellite Domain [FoxLong02-a]. Data storage in the satellite has capacity. Data can be down-linked to the ground station "at the same time" it's being collected. Two actions interplay on the level of data storage. Conservative decrease of data by the *downlink* action can allow actual overflow of data by the action of *collection*. This kind of domain can be modeled only in the temporal models of Level 4 or Level 5.

2.2.1.2 Continuous Durative Actions (Level 4)

Syntax

In addition to the syntactic form of discretised durative actions, in Level 4 it is possible to represent a continuous effect:

(**increase**[14] <NumericFluent> (* ***#t*** <ArithmeticExpr>)) or,

(**increase** <NumericFluent> (* <ArithmeticExpr> ***#t***))

where, ***#t*** represents the elapsed time since the action started.

Also, instantaneous actions are allowed in Level 4.

Semantics

A continuous change statement represents the value of <NumericFluent> which is continuously increasing/decreasing over the period of the durative action by the rate

of <ArithmeticExpr>. At any point within the period, the exact value of the fluent is accessible.

Example 2-2: Bath Domain in Level 4

```
(:durative-action          fillBath
 :parameters     (?b – bath )
 :duration       ( )
 :condition      (and
                   (at start  (plug_in  ?b))  (at start  (not (tap_on  ?b)))      (1)
                   (over all  (plug_in  ?b))  (over all  (tap_on  ?b))            (2)
                   (over all   (<=  (level ?b) (capacity ?b)))                    (2')
                 (at end (tap_on  ?b))  (at end    (plug_in ?b)))                 (3)
 :effect         (and
                    (at start  (tap_on ?b))                                       (4)
                   (at end  (not (tap_on ?b)))                                    (5)
                   (increase  (level ?b)  (* #t  (flow ?b))))                     (6')
)
```

Observe that the constraint on duration, (0), in Example 2-1 is represented by (2') and (6') in Level 4. Unlike Level 3, the exact value of the level is accessible to any other concurrent action such as addBubble in Appendix 3.

Limitations of Expressiveness

In this model, the continuous change in a durative action is bound by the duration of the action. In contrast to this, there are situations in which the period of continuous change is determined by the environment and so is beyond the planner's control. Such an example is "Rover" Domain [FoxLong02-a] which models a rover's recharging process that is triggered as soon as it is in the sunlight. A rover cannot make any decision on how long the recharging process continues but can only exploit

[14] Likewise, decrease.

the effect. Also, occurrences that are triggered by the environment cannot be represented in Level 4.

2.2.1.3 Autonomous Processes (Level 5)

Syntax

```
(:process/event      <NameOfAction>
 :parameters         <ListOfArgumentsWithTypes>
 :precondition       <LogicalExpr>
 :effect             <LogicalExpr>
 )
```

The syntactic components of Level 5 are processes[15], events and instantaneous actions. PDDL+ requires that a process has at least one continuous effect and an event has at least one numeric constraint in the precondition.

Semantics

The preconditions of a process are triggering condition, invariant condition, and also terminating condition. Continuous changes are captured by processes, each of which is triggered and terminated either by actions, events, or ongoing processes. A process triggered by actions can often be captured by a durative action with flexible duration under the planner's control. A process triggered by events cannot be modeled by a durative action, whose duration the planner may not know or care about. An event, either deterministic or nondeterministic, makes an instantaneous transition between states of the environment, which is not a choice of the planner.

Naturally the following semantic constraints are imposed:

- *Preservation of continuity*: The continuous change over $[T_1,T_2]$ should be equal to the sum of the change over $[T_1,T_3]$ and the change over $[T_3,T_2]$, for any T_3, $T_1 < T_3 < T_2$.
- *Triggering of an event with no slip of time*: as soon as its precondition is satisfied, the event should be triggered.
- *Triggering of a process with no slip of time*: like triggering of an event
- *Termination of a process with no slip of time*: as soon as its precondition is violated by continuous or discrete changes, the process should be terminated.

Example 2-3: Bath Domain in Level 5

```
(:process              bath_filling
 :parameters   (?b - bath)
 :precondition (and    (<= (level ?b) (capacity ?b))
                       (> (flow ?b) 0))
 :effect               (increase (level ?b) (* #t (flow ?b)))
 )

(:event        flood
 :parameters   (?b - bath)
 :precondition (and    (>= (level ?b) (capacity ?b)) (> (flow ?b) 0)
                       (dry_floor ?b))
 :effect       (and    (wet_floor ?b) (not (dry_floor ?b))))
```

Observe that

- The “bath_filling” process should be triggered as soon as the level of the bath is less than the capacity and there is inflow to the bathtub. The process should be active until these conditions become false, either by discrete changes or

[15] The process model in McDermott’s Opt [McDermott03-a] contains triggering conditions and effects, invariants, and terminating effects. So, its syntactic structure is more like a durative action in PDDL+.

continuous changes. The process should be terminated as soon as these conditions become false.

- As soon as the level of water exceeds its capacity, event "flood" should be triggered.

2.2.2 Concurrent Plans

Plans with concurrent actions were not considered in the previous versions of PDDL. Concurrency is restricted by non-interference rules of the PDDL+ in Appendix F; any two actions violating these rule(s) cannot be run simultaneously. In particular, non-interference rules adopted in PDDL+, an extension of mutex rules for Graphplan [Blum97] with numeric features, include the "*no moving targets rule*," which is stronger than the commonly used "*no concurrent actions can affect the parts of the state relevant to the precondition tests of other actions in the set, regardless of whether those effects might be harmful or not.*" The reason[16] for adopting the rule is to make concurrency checking in polynomial time of the size of actions and pre- and post-conditions.

The introduction of concurrency into the framework of actions raises the following issues on interactions among actions being executed in parallel [Pinto00]:

- *Precondition interaction problem*: Depending on whether or not other actions are performed concurrently, actions can or cannot be performed. One example

[16] A good example is action A: (p V q) => r, action B: p => ($\neg$p $\wedge$ s) [FoxLong03]. Handling the case implied by this example requires checking the consequence of interleaving preconditions and effects in all possible ways. Thus so, PDDL+ semantics defines these two actions interfering.

of such an action is that if each dancer in a closed circle takes one step to the right, then each action is possible only if all dancers move synchronously.

- *Synergistic effects*: Actions that are performed concurrently produce effects that neither action would have if performed in isolation. An example of this is, when lifting both ends of a table simultaneously, it has the effect of raising the table off the ground.
- *Cancelled effects*: Two actions cancel each other's effects. An example of this would be a door that is pushed and pulled at the same time with the same force.
- *Cumulative effects:* More than one action makes changes concurrently on the same fluent. An example of this is the direct influence on a numeric fluent.
- *Interactions between durative actions that are overlapped*, such as pulling the door while holding a spring loaded latch open.

These issues are realized in PDDL+ in the following ways:

- Precondition interaction problem: There is no feature in PDDL+ to support multi-agent environments.
- Synergistic effects and Cancelled effects: Precise simultaneity is outside of the control of an executor. The interpretation of simultaneity in PDDL+ is that an executor can execute the two actions within a fine but nonzero tolerance, and the effects can occur [FoxLong02].

- Cumulative effects: Simultaneous updates on numeric-valued fluents are allowed, only if the operations are commutative. Asynchronous concurrency [Brenner01] is not possible with propositional fluents.
- Interactions between durative actions executing overlapped: Interactions of this kind can be modeled as invariants within a durative action.

2.2.3 Divided Instant Problem

The Divided Instant Problem [Vila94] is about the truth value of a propositional fluent at the moment the state transition is happening. The solution adopted in PDDL+ is to model an action as an instantaneous state transition whose effects are effective at the moment of the application of the action. Thus, the state preceding the action holds over an interval that is open on the right (i.e. not including the instant in which the action takes place). The effect of the action holds over the interval that is closed on the left (i.e. all real valued times equal or greater than the time the action takes place.).

2.3 Plan Metrics[17]

As mentioned before, PDDL+ was intended to support temporal metric domains. In temporal metric domains, it is more likely that plan quality is judged by temporal quality such as makespan (total time of the plan) or plan cost such as resource consumption or cumulative action cost, rather than solely based on the length of a

[17] The TM-LPSAT cannot handle the plan metric, which is an inherent difficulty with a SAT–based approach.

plan. To evaluate the quality of a plan specific to a given problem, PDDL+ supports representation of the plan metric optionally given to the planner. The plan metric is expressible in maximization or minimization of functional expression of numeric-valued fluents.

Chapter 3 Previous and Related Work

To our knowledge, there is no previous domain-independent planning SAT-based system, which can reason about actions over continuous time. There are a few SAT-based metric planners: the LPSAT metric planner [Wolfman99] [Wolfman00], and as a variant [18] of the SAT-based approach, the ILP [19] metric planner [KautzWalser00] [KautzWalser99] and the MILP[20] metric planner [Bockmayr98] [Bockmayr99]. These systems, however, plan in atomic time. Our encoding of numeric-valued fluents with only discrete changes, as defined in Level 3 of PDDL+, is similar to that of LPSAT, only differing in the definition and reasoning of time points: time points in our encoding are metric and variable distance away, in contrast to their step-based definition of time points.

In Section 3.1, temporal metric planning systems dealing with temporal models of durative actions or continuous changes are reviewed and their relation to our work is considered. In Section 3.2, we briefly give an overview of formalisms developed in different areas for modeling dynamic worlds with continuous behaviors as well as discrete behaviors: hybrid automata, qualitative processes, and logical formalisms.

3.1 Domain-Independent Planning Systems Dealing with Continuous Time

3.1.1 Planning Systems Dealing with Durative Actions

[18] SAT-encoding can be easily converted (transformed) into encoding for 1/0 Linear Programming; Stronger formulations of ILP problem in planning domains, rather than direct translation of SAT encoding, have been explored [Bockmayr98] [Bockmayr99] [Dimopoulos02] [Vossen00] in order to improve performance.

[19] ILP stands for Integer Linear Programming.

Generally, the expressiveness of temporal constraints that state-of-the-art temporal planners can handle is limited: constant durations and STRIPS-style operators, hardly handling external events and autonomous processes. There are a few temporal and metric planners such as TP4 [HaslumGeffner01], LPG [Gerevini03] and Sapa [Do03-b], but they can deal mostly with discrete changes within a durative action. Sapa and LPG claim to be able to deal with continuous change "in theory", but their scalability and concurrency is unclear.

There are three temporal planning systems dealing with durative actions, which are related to our work in some aspects:

Mali's SAT-based Temporal Encoding [Mali02-a] is an extension of state-space encoding [KautzMcAllester96] of duration 1 (i.e. discrete time) with constant integer durations. The temporal model adopted is that used in TGP [21] [Smith99]: all propositions are either undefined or persistent over the duration so that they should be protected over the duration. A step is defined at each integer time. His encoding could be mapped to the encoding in continuous time by scaling up unit time by integer 1, but it is certainly not a good idea to define a step at each unit time. The optimization technique using plan graph used in his encoding is specific to the actions of constant integer durations.

---

[20] MILP stands for Mixed Integer Linear Programming.

[21] The temporal model used in TGP and TP4 is called "Blackbox" model: (i) the preconditions should be satisfied at the starting point of the action, (ii) The values of propositions in the effects are changed sometime during the action (so, the values are "undefined"); their values are valid only the ending point of the action, (iii) precondition whose value is not changed in the effect should be persistent over the duration of the action.

MILP Temporal Planner [DimopoulosGerevini02] is based on a novel architecture for the temporal planning. The planning problem is encoded as two parts: (i) 0/1 integer encoding of plan graph generated from the non-temporal aspect of the problem, and (ii) 0/1 integer linear encoding of temporal constraints in terms of start times and durations of actions and start times of fluents at each layer. The two encodings are glued together via 0/1 action variables. All inequalities from both encodings are solved using a MILP solver (Mixed Integer and Linear Programming). It plans by branch and bound through bounded-length plans based on optimization function. Thus, the plan is suboptimal in terms of the given objective functions, and it cannot guarantee optimality in terms of makespan (total time of the plan), since a plan taking less time can have more steps.

As compared to the LPSAT architecture in which the linear constraint solver and the propositional constraint solver are separate modules, the unified encoding in MILP has the potential to exploit the strong interactions which may lead to extensive value propagation. Their temporal encoding is assumed to have actions of constant real-valued durations; otherwise it becomes nonlinear.

LPGP [LongFox03] is a temporal planner based on the Graphplan framework [Blum97], in which Level 3 of PDDL+ is adopted as a temporal model, dealing with discretized durative actions with constant duration. The plan graph constructed is a representation of the logical structure of a plan: each action layer corresponds to the occurrence of interesting instantaneous activities. Thus, conceptually, each state (fact layer) carries duration. In the graph construction phase, a durative action is decomposed into instantaneous actions: a starting action, an ending action, and an

action with invariants as preconditions. The instantaneous actions are causally linked with artificial tokens (propositional fluents). During the plan extraction phase, linear constraints on the durations of fact layers and durative actions included in the current plan are constructed backward; whenever a start action is added in the current plan, the consistency of the linear constraints is checked by LP solver, which is that the sum of durations of fact layers between start action and end action should be equal to duration of the durative action.

The disadvantage of this approach is that, compared to TGP[22] [Smith99] or TP4[23] [HaslumGeffner01] where the plan graph is a representation as a flow of uniform time, it cannot guarantee optimization of makespan, since graph generation is separated from temporal optimization. The model of time used in the plan graph is the same as the model of time adopted in our TM-LPSAT, which entirely complies with the semantics of PDDL+. The difference is in how the temporal constraints are reasoned: in the TM-LPSAT, temporal constraints intertwined with metric constraints at the time points are searched non-directionally by a SAT solver. Thus, the TM-LPSAT does not suffer from the difficulties caused by backward search, such as handling with a durative action whose ending action is not included in the plan, but whose starting action needs to be included in the plan.

[22] TGP, Graphplan-based temporal planner using extended mutex reasoning, can find a plan which takes the shortest, i.e. optimal in terms of makespan.
[23] TP4 heuristic-based planner finds an optimal plan in terms of makespan, using heuristic function.

3.1.2 Planning Systems Dealing with Continuous Changes

There have been planning systems that can reason about "continuous change" in restricted contexts including Hendrix's Processes[Hendric73], Vere's DEVISER [Vere83], Wilkin's SPIE [Wilkins88], Simmon's GORDIUS [Simmons88], Dean's FORBIN [Dean88], Drabble's EXCALIBUR [Drabble93], and ZENO [Penberthy93] [Penberthy94]. All of these are partial order planners. Compared to the other systems, the ZENO system can handle quite expressive temporal and metric constraints, although its handling of continuous changes is restricted to non-concurrent updates. However, the difficulty of managing the temporal constraints involved, typical of partial order planning, turns out to be the bottleneck in its performance. It has been reported that the ZENO is unable to solve even the simplest metric logistics problem that the LPSAT metric planner can do [Wolfman00], although the ZENO algorithm is complete and sound. Compared to the ZENO, our TM-LPSAT planner can handle concurrent continuous updates as defined in semantics of PDDL+, which include any concurrent combination of continuous and discrete changes consistent with the "no moving target" rule mentioned in Chapter 2. Also, autonomous process triggered by nondeterministic exogenous events can be handled in the TM-LPSAT.

Very recently, McDermott has presented a heuristics-based planner called Optop (actually an extension to the Optop planner) using an estimated regression graph [McDermott03-a] [McDermott03-b]. To our knowledge, this is the only domain-independent planner which can handle autonomous processes as well as objective functions. It is reported that performance of the approach is not very promising at least at this point. In the SAT-based planning framework, it would not be feasible to deal

with optimality in terms of given objective functions, although there are simulated SAT approaches or variants of SAT-encoding such as MILP which can find optimality (optimal within the plan length) to objective functions. Certainly it is an advantage of heuristic-based planners that they are open to be able to find global optimality. Another characteristic of Optop is that it does not instantiate all action schemas in advance, which makes it feasible for domains with objects dynamically created or for domains involving numbers and other infinite sets. That is clearly a strong point over the planning frameworks searching through all grounded actions, especially, over the Graphplan-based planning framework. The TM-LPSAT can deal with domains involving numbers and (a subset[24] of) dynamically created objects as dealt in Section 4.5 as multi-capacity objects with variable capacity, although it is necessary to ground all action schema at the encoding stage in the SAT-based framework.

3.2 Formalisms for Modeling Continuous and Discrete Changes

3.2.1 Hybrid System

A hybrid system [Larsen97] [Henzinger96] is a formalism for modeling a dynamical system whose state has both a discrete component, which is updated in a sequence of steps, and a continuous component, which evolves over continuous time. A hybrid system is composed of a collection of hybrid automata, which communicate either by shared variables or by synchronization on a channel. A hybrid automaton is an extended FSM that models a discrete behavior by a FSM as well as a continuous

[24] If it is not necessary to identify the objects individually.

behavior by a real-valued variable. A node contains (i) activities: functions describing continuous behaviors of real-valued variables; (ii) invariants: formulas over the values of the variables to be in this state. A transition is labeled by (i) a precondition (guard): a formula over the variables specifying condition for the transition to be taken; (ii) an action: an instantaneous discrete change; (iii) a post-condition (assignment): a discrete change to the real-valued variables possibly depends on the previous values to the variables. The state of the hybrid automaton can change either by an instantaneous discrete change or by time passage. Linear Hybrid Systems, in which invariants, guards and activities are linear in time, form a subclass for which tractable algorithms for automatic analysis, like Timed Automata (in which the only continuous variables are clocks) are known.

The semantics of PDDL+ Level 4 & 5 (modeling of continuous and discrete changes) [FoxLong01] is described using Hybrid Automaton. [FoxLong02-c] studies the use of HyTech[Henzinger97], a model checking tool for hybrid systems, for domain analysis in planning. [Audemard03] formulates bounded reachability problem of a linear hybrid system as a satisfiability problem over propositional and linear constraints, and solved it with MathSAT SAT solver [Audemard02-b]. However, concurrent continuous and/or discrete changes have not dealt with in the work.

3.2.2 Qualitative Reasoning

Discovering qualitative techniques for representing and reasoning about a continuously changing world is the focus of research in qualitative physics [Forbus96]. Qualitative simulation [Kuipers01] is the construction of a set of possible behaviors consistent with a model of dynamic systems represented by “qualitative” version of differential equations. The following are reasoning systems based on qualitative simulation, integrating discrete changes by actions and continuous changes by qualitative processes.

Forbus’ Qualitative Process Theory with Actions [Forbus89] Given an initial state, a set of actions, and a set of processes, qualitative simulator generates all possible action sequences intertwined with all possible process evolutions, that is, entire plan space; then, determine a sequence of actions solving the problem within the plan space. Clearly the bottleneck in his system is combinatorial explosion.

Drabble’s EXCALIBUR [Drabble93] is a hierarchical partial-order planner that can reason about external event, complex resources, and continuous change. Using qualitative simulation, processes are arranged in the plan generated ignoring all metric preconditions and effects; if the plan does not satisfy the original metric preconditions, the plan is repaired using a variety of heuristic techniques. The heuristic strategies used in plan repair are not presented clearly.

Farquhar’s Qualitative Process Compiler [Farquhar94] extends Qualitative Process Theory and is implemented in QSIM. A physical world is represented by a set of model fragments, each of which captures some aspects of the domain by providing knowledge of both algebraic and logical nature. The definition of a model fragment

contains a set of participants, the relations which must hold in order to instantiate the fragment and consequences. Given the entities involved, their relationships and initial conditions, the compiler instantiates proper model fragments. It then does qualitative simulation on the domain. In case the predicted behavior extends across the boundary of applicability a fragment, a new model of the resulting situation is dynamically constructed by considering other fragments.

3.2.3 Logical Formalisms

A number of works integrate continuously changing quantities into a logical formalism: McDermott's Temporal Logic [McDermott83]; Sandwell's Features and Fluents [Sandwell89]; Davis' Axiomatization of Qualitative Process Theory in First-Order Theory [Davis92]; Davis' Modeling of Autonomous Agents in terms of continuous control and choice [Davis94]; Reiter's extension [Reiter96] and Pinto's extension [Pinto98-b] into Situation Calculus; Miller and Shanahan's extension into Event Calculus [Miller96-a] [Miller96-b] [Shanahan90]; Thielscher's extension into Fluent Calculus [Thielscher99].

In particular, the representation used in Event Calculus is similar to our SAT encoding, although the reasoning in Event Calculus involves circumscription. To integrate the continuous changes generally represented in algebraic equations, the notion of trajectory is introduced into Event Calculus. A trajectory describes the functional relationship between the value of continuously changing quantity and the time it elapsed since it started to change. A trajectory is attached to a fluent such as

“flowing” by axioms, which ensures that the trajectory is valid as long as the related fluent to which the trajectory is attached holds.

Chapter 4 Encoding Temporal Metric Planning in Continuous Time

In this section, we show encoding schemes for temporal models of continuous time as defined in PDDL+: durative actions with discrete changes, durative actions with continuous changes, and a real-time temporal model of processes and events. The encoding covers the ADL subset of PDDL+, accommodating such features as negative preconditions, disjunctive preconditions, equality, quantified preconditions, and conditional effects. We also show, by extending numeric-valued fluents, how multi-capacity resources and interval-valued fluents can be encoded in the SAT-based framework.

In Section 4.1, we show the adaptations of PDDL+ made in the TM-LPSAT and the encoding scheme adopted in our temporal encoding. The representation of time and fluents is shown in Section 4.2. Section 4.3 presents the temporal encoding of durative actions as defined in PDDL 2.1 Levels 3 and 4. The real-time temporal encoding as defined in PDDL+ Level 5 is presented in Section 4.4. The encoding of multi-capacity resources and interval-valued fluents is shown in Section 4.5.

The features of these temporal models are overviewed in Chapter 2. The summary of syntax and semantics and additional assumptions made in the encoding are recapitulated in the subsections, 4.3.1 for Level 3 and 4 and 4.4.1 for Level 5. The sets, predicates and functions used in this Chapter as well as the time-labeling convention are defined in Appendix A. Encodings of sample domains are given in Appendix C for durative actions and Appendix D for the real-time temporal model.

4.1 Preliminaries

4.1.1 Adaptation of PDDL+

Open-World Assumption vs. Closed-World Assumption

According to the document of PDDL2.1 [FoxLong02-b], Closed World Assumption is not sustained any more: Requirement flag for assumption on world, which existed in the earlier versions of PDDL, was removed in PDDL+. What world, then, is supposed to be assumed? Is it up to the domain designer? Open World Assumption and Closed World Assumption are quite different assumptions. We adopt Closed World Assumption for the encodings presented below.

Numeric-Valued Parameter to Action

This feature has been removed with the latest updates in PDDL+. The reason for this change, they say, is that it generates an infinite search space, which is a serious bottleneck to planning frameworks[25] based on searching feasible actions over all grounded actions such as Graphplan [Blum97]. However, we think the domain definition language should not be restricted by the limitation of certain frameworks. More importantly, there are domains, like pouring flour into a bin with a measuring cup, that can only be represented using actions with numeric-valued parameters. This

[25] TM-LPSAT is one of the planning frameworks searching over all ground actions, but a numeric-valued parameter can be dealt with a real variable in TM-LPSAT. Likewise, the same technique can be used in Graphplan planning framework.

feature[26] is reinstated in TM-LPSAT. As a natural extension, we allow an interval-valued parameter to action.

Extension of Function Types

In PDDL+, the type of a function is assumed to be numeric-valued (real number, possibly). There are no means to declare other types or features. Thus, a natural direction of extension is type:

- Number: { positive | negative } { integer | real } { float | fluent }
- Interval: interval { float | fluent }

Such type specifications of a numeric-valued fluent can be useful in allowing an arithmetic constraint solver to restrict the feasible solution spaces [Borning98][27]. In particular, it is useful for reducing the size of encoding in the TM-LPSAT to distinguish *float* and *fluent*: only a continuously changing numeric-valued *fluent* needs to be encoded with two real variables[28]. Otherwise, a numeric-valued *float* is replaced by its value.

Most numeric-valued fluents in planning contexts have *capacity*, with which the use of the fluent must be checked for validity. This capacity could be constant or time-varying. In Section 4.5, we show how multi-capacity resources (i.e. sharable, reusable

[26] This feature in discrete time was employed in the LPSAT metric planner [Wolfman00]. The same encoding can be used for durative actions in continuous time, since the parameter is like *float* local to the action. However, in durative actions it is necessary that the variable corresponding to a numeric-valued parameter needs to be time-labeled. This is so in order to allow to be concurrent two instances of the same durative action with numeric parameters that is started at different time points.

[27] In the Cassowary arithmetic solver integrated in the LPSAT engine, all numbers are assumed to be real. On the other hand, in the standard Simplex method, variables are assumed to be non-negative real numbers.

metric quantity with *capacity*) and interval-valued fluents can be encoded as constraints in the SAT-based framework.

4.1.2 Encoding Scheme

According to the classification of encoding schemes in [Ernst97], the encoding scheme we adopt is "regular explanatory frame representation[29]". The reason why we adopt this scheme is that (i) it has been proven empirically to perform better [Giunchiglia98] [Ernst97], (ii) it is easier to apply general optimization techniques as well as to adaptation to other encodings, and (iii) it supports parallelism.

To adjust the encoding scheme for continuous time as well as numeric-valued fluents, the conventions adopted are as follows, in comparison with the encoding in discrete time:

- Each time point is an instant over **R*** at which some interesting activities may happen, where interestingness is with respect to goal achievement; a metric value is bound to each time point; T_i is the next time point to T_{i-1}, and they are a variable distance away.
- The values of all propositional fluents and (Boolean variables for) ground actions are defined at each time point; two values[30] for each numeric-valued fluent are defined at each time point.

---

[28] Refer the Section 4.2.2. For a numeric-valued fluent changing only discretely, one real variable is enough for encoding.
[29] It is also called "State Space Encoding". There are other variants of state space encoding, though.
[30] In order to deal with continuous change, two variables are used to represent numeric-valued fluents. Refer Section 4.2.2.

- Generally[31], the precondition of a happening at time point T_i is defined at previous time point T_{i-1}, the post-condition is defined at time point T_i.

The encoding scheme consists of the following axioms, which are shown along with examples from Block World domain:

Move(X,Y,Z) : move X from Y to Z
Precondition: Clear(X) ^ On(X,Y) ^ Clear(Z)
Post-condition: Clear(Y) ^ On(X,Z) ^ ¬ Clear(Z) ^ ¬ On(X,Y)

(1) Universal Axioms

Initial state is at time point T_0; goal state varies with temporal models.

(2) Action Representation

Action implies its precondition and post-condition

e.g. Move(A,B,D, T_i) => Clear(A, T_{i-1}) ^ On (A,B, T_{i-1}) ^ Clear(D, T_{i-1}) ^
Clear(B, T_i) ^ On(A,D, T_i) ^ ¬ Clear(D, T_i) ^ ¬ On(A,B, T_i)

(3) Explanatory Frame Axioms

For each propositional fluent, it enumerates the set of actions that could have occurred in order to account for a state change; it contrapositively denotes persistency.

e.g. Clear(D, T_{i-1}) $\wedge$ ¬Clear(D, T_i) => (Move(A,B,D, T_i) $\vee$ …$\vee$ Move(C,Table,D, T_i))

For a numeric-valued fluent, it requires linear equations of simultaneous discrete changes and concurrent continuous changes between two neighboring time points, T_{i-1} and T_i. Those equations are given in equations (1) and (2) of Section 4.2.2. These explanatory axioms support *parallelism* and ensure that the encoding is *sound*.

[31] This is the case for event and action, but not for process. Refer Section 4.4.3.

(4) Conflict Exclusion Axiom

In order to make any total order plan generated from the given parallel plan a valid plan, add clauses of mutual exclusion for each pair of conflicting actions, which is based on the interference rules defined in PDDL+. The details of the rules are given in Section 4.3.5.

e.g. ($\neg$ Move(A,B,D, T_i) $\vee$ $\neg$ Move(A,D,B, T_i)) for fluent Clear(D) or Clear(B)

4.2 Representation of Time and Fluents

4.2.1 Model of Time

Time is modeled as linear and isomorphic to the real numbers.

Time points, with metric values defined over **R***(nonnegative real number), represent instants at which *interesting instantaneous activities* happen. A state is defined in terms of a finite set of propositional fluents and numeric fluents in the domain. The transition between states happens by instantaneous actions or events, changing the values of propositional fluents or discrete changes on numeric-valued fluents; any continuous change can go on within a state. This entirely complies with the semantics of PDDL+ given in the elements of Hybrid Automata [FoxLong01]. Although time is continuous and an action can be scheduled to begin at any time point, only a finite number of happenings between any two time points are allowed by the semantics of PDDL+.

Representation and reasoning of actions and changes is based on time points, rather than on interval: the values of all fluents and ground actions are defined in terms of time points.

4.2.2 Representation of Fluents

Propositional Fluents

The truth value of a propositional fluent ***pf*** at time point T_i is ***Proposition(pf,*** T_i***)***.

Numeric-Valued Fluents

In order to deal with continuous changes, two variables are introduced to represent values of each numeric-valued fluent at time point T_i:

Value$_{before}$***(nf,*** T_i***)***

- To capture all concurrent continuous changes made on a numeric fluent ***nf*** by durative actions or processes over the interval (T_{i-1},T_i)

Value$_{after}$***(nf,*** T_i***)***

- To capture all simultaneous discrete changes made on a numeric fluent ***nf*** by actions (events) happening at T_i

This representation[32] makes possible to reason about concurrent continuous and discrete changes, while preserving continuity on a continuously changing fluent ***nf***.

4.2.3 Axioms on Numeric-Valued Fluents

The time-labeling convention used is defined in Appendix B.

Arithmetic Constraints between $Value_{before}(nf,T_i)$ and $Value_{after}(nf,T_i)$

(1) $Value_{before}(nf,T_i) = Value_{after}(nf,T_{i-1}) + \sum a \; RateOfChange(a,nf,T_{i-1}) * (T_i - T_{i-1})$

$= Value_{after}(nf,T_{i-1}) + \sum a \; NetContiChange(a,nf,T_{i-1},T_i)$

- ***a*** is either a durative action or a process; it distinguishes instances of the same durative action (or process) at different time points to support concurrency
- If ***a*** is not active at T_{i-1},

$$RateOfChange(a,nf,T_{i-1}) = NetContiChange(a,nf,T_{i-1},T_i) = 0$$

- Let us call this equation $LinearContinuousEq(nf,T_i)$

(2) $Value_{after}(nf,T_i) = Value_{before}(nf,T_i) + \sum a \; DiscreteChange(a,nf,T_i)$

(2') $Value_{after}(nf,T_i) =$ <*NewValue*>

- Exactly one of (2) and (2') should be activated at each time point, which is imposed by Axiom (6): (2) is for additive discrete changes; (2') is for assignment by a new value
- ***a*** is either an event or an instantaneous action including start or end actions of durative actions; again, it distinguishes instances of the same durative action started at different points to support concurrency

---

[32] In the temporal encoding of Level 3, a numeric-valued fluent can be encoded with only one real variable, since continuous change is not allowed. In the TM-LPSAT, *float* and *fluent* are distinguished.

- If ***a*** is not active at T_{i-1}, ***DiscreteChange(a,nf,*** T_i***)*** = 0
- Let us call (2) ***LinearDiscreteEq(nf,*** T_i***)***

Discrete Change on a numeric-valued fluent via *increase*, *decrease*, or *assign* statement

(***increase*** <***nf***> <*DiscreteChange*>) at time point T_i on numeric-valued fluent ***nf*** is encoded as follows:

(3)[33] **Active**(**a**,T_i) $\Rightarrow$ [**DiscreteChange(a,**T_k**,nf,**T_i**) =** <DiscreteChange>[**Ti-**]]
(4) $\neg$ Active(a,T_i) $\Rightarrow$ [DiscreteChange(a,T_k,nf,T_i) = 0]

- [***DiscreteChange(a,nf,*** T_i***) =*** <*DiscreteChange*>[T_i**-**]] is an arithmetic constraint
- (3) & (4) ensure that the discrete change made by action ***a*** is valid if the action is active at T_i; otherwise, the change is 0.
- ***DiscreteChange(a,nf,*** T_i***)*** is accumulated in (2)

(***assign*** <***nf***> <*NewValue*>)$_{Ti}$ is encoded as follows:

(5) ***Active***(***a***,T_i) $\Rightarrow$ [***Value***$_{after}$***(nf,*** T_i***)*** **=** <*NewValue*>[***Ti-***]]

- (5) ensures that linear constraint [***Value***$_{after}$***(nf,*** T_i***)*** **=** <*NewValue*>[***Ti-***]] is imposed, if action ***a*** is active at T_i

[33] The current version of TM-LPSAT implements rather restricted formulations for (3) and (4):
(3) ***Active***(***a***,T_i) $\Leftrightarrow$ [***DiscreteChange(a,nf,*** T_i***) =*** <*DiscreteChange* >[T_i**-**]]
(4) [***DiscreteChange(a,nf,*** T_i***) =*** <*DiscreteChange* >[T_i**-**] $\oplus$ [***DiscreteChange(a,nf,*** T_i***) =*** 0]
which reduce calls to LP solver. However, this is based on the assumption that the discrete change by active actions is never 0.The same technique is used for formulations relating an action (event or process) and its corresponding linear constraints.

The following two axioms are to ensure that exactly one of additive discrete change defined in (2) and discrete change by an assignment statement defined in (5) is always active at each time point:

(6) [$\wedge$ $\boldsymbol{a} \in$ ***Set(nf)*** $\neg$ ***Active***$(\boldsymbol{a}, \boldsymbol{T_i})$] $\Rightarrow$ ***LinearDiscreteEq(nf,T_i)***

(7) $\forall$ $\boldsymbol{a_i} \in$ ***Set(nf)*** $\forall$ $\boldsymbol{a_j} \in$ ***Add(nf)*** $\bigcup$ ***Set(nf)*** [$\neg$ ***Active***$(\boldsymbol{a_i}, \boldsymbol{T_i})$ $\vee$ $\neg$ ***Active***$(\boldsymbol{a_j}, \boldsymbol{T_i})$]

- (6) ensures that ***LinearDiscreteEq(nf,T_i)*** defined in (2) should be active unless any of action with an *assign* statement on ***nf*** is active
- (7) ensures that any action with an *assign* statement on ***nf*** is mutually exclusive with any other action with an *assign* statement on ***nf*** or any action with an *increase* or *decrease* statement on ***nf***

Continuous Change on a numeric-valued fluent ***nf*** over a period of time

(***increase***[34] <***nf***> (* ***#t*** <*RateOfChange*>))$_{Ti}$ over $(\boldsymbol{T_{i-1}}, \boldsymbol{T_i})$ is encoded as follows:

(8) ***Active***$(\boldsymbol{a}, \boldsymbol{T_{i-1}}) \Rightarrow$ [***NetContiChange(a,nf,T_{i-1},T_i) = (T_i – T_{i-1})*** * <*RateOfChange*>]

(9) $\neg$ ***Active***$(\boldsymbol{a}, \boldsymbol{T_i}) \Rightarrow$ [***NetContiChange(a,nf,T_{i-1},T_i) =*** 0]

- To be the continuous change in $(\boldsymbol{T_{i-1}}, \boldsymbol{T_i})$ piecewise linear, <*RateOfChange*> is assumed to be constant (known in a ground action)
- (8) & (9) ensures that if ***a*** (process or durative action) is active at $\boldsymbol{T_{i-1}}$, constraint [***NetContiChange(a,nf,T_{i-1},T_i) = (T_i – T_{i-1})*** * <*RateOfChange*>] should be imposed; otherwise, [***NetContiChange(a,nf,T_{i-1},T_i) = 0***]
- ***NetContiChange(a,nf,T_{i-1},T_i)*** is accumulated in equation (1)

[34] "*decrease*" is analogous. In that case, <*RateOfChange*> should be prefixed by minus sign.

4.3 Representation of Temporal Model of Durative Actions (PDDL+: Level 3 & 4)

A durative action with discrete changes (Level 3) is a special case of a durative action with discrete and continuous changes (Level 4). The encoding for Level 4 is presented in this section.

The encoding for Level 3 can be encoded by using only one variable for each numeric-valued fluent at each time point, rather than $\boldsymbol{Value_{before}}$ and $\boldsymbol{Value_{after}}$, since there is no continuous change. It can be encoded as done in discrete time with additional constraints on time points, which are now variable distance away rather than uniform time away.

Starting with recapitulation of syntax and semantics, axioms for initial and goal states, axioms for actions, frame axioms and axioms for mutual exclusions are presented in that order. The sample domain in Level 4 and its encoding are in Appendix C. The time-labeling convention is defined in Appendix B.

4.3.1 Syntax and Semantics

The syntactic form of a durative action in Level 3 & 4 is consisted of a start action with temporal annotator *at start* (setting up local conditions for the durative action), an end action with temporal annotator *at end*, and invariants with temporal annotator *over all*. The start and end actions are instantaneous; the invariants are held over the period of the action duration excluding end points. No point between those two end points is accessible; all discrete changes can happen only as effects of the start and end action.

A constraint on duration of a durative action can be represented as a conjunction of comparisons (=, <, >) of arithmetic expression and *?duration*.

The continuous change of a numeric-valued fluent can be represented in Level 4, which is over the period of a durative action. The value of a numeric-valued fluent, continuously changing, are updatable as well as accessible by other actions at any time point over the period of the durative action. In the TM-LPSAT, it is assumed that between any two time points, the rate of change is constant, i.e., *piecewise linear*.

Along with durative actions with continuous changes, Level 4 contains all types of actions defined in Level 1, 2, and 3, including instantaneous actions and durative actions with discrete changes.

Concurrency

More than one instance of the same action (i.e. the same durative actions started at the different time points) can be concurrent, since each of those instances may have different durations and another instance can start even before the instances started in the previous time points are ended.

A durative action with a numeric- or interval-valued parameter needs to be encoded for the parameter with a different variable in each instance, since more than one instance can be concurrent. Consequently, this prevents the invariants to be shared among the instances of the same action started at different time points, which is substantial increase in complexity as length of plan increases. More on this issue will be discussed in the encoding of the invariants.

4.3.2 Representation of Initial State and Goal State

Initial State

Initial state is implemented as effects of a dummy action with non-precondition.

Time-labeling: <Initial State>[T_0+]

- A propositional fluent is time-labeled by T_0
- An initial assignment to a numeric-valued fluent is time-labeled by $Value_{after}$ at T_0.

Goal States

The semantics of PDDL+ requires that a durative action started in a plan should be finished in the plan; in consequence, the goal state may be satisfied in a state before all durative actions initiated are finished. The goal state is represented as a disjunction of the goal at each time point, except T_0.

Time-labeling at T_i: <Goal State>[T_i+]

- A propositional fluent is time-labeled by T_i
- A reference to a numeric-valued fluent is by $Value_{after}(nf,T_i)$

4.3.3 Representation of Durative Actions

Define a durative action ***DA*** = (***As***,***Ae***,***Inv***), where ***As*** is the start action, ***Ae*** is the end action, ***Inv*** is the invariant conditions. Let ?$duration(DA)_{Ti}$ be the duration of ***DA*** starting at time point T_i.

Axiom on time point variables

$$\forall i,\ 0 < i <= n\ \ [\ \boldsymbol{T_0 = 0}\ \wedge\ \boldsymbol{T_i}\ \geq\ \boldsymbol{T_{i-1}}\ +\ \mathcal{E}\ ]$$

- Each time point is assumed to be distinct, not considering tolerance (i.e. each time point has different time value over **R***). More than one activity can happen at each time point simultaneously.
- In practice, $\mathcal{E}$ should be small enough not to make any change in the values of quantities in the domain.

Axioms for Start Action and End Action

Assume that start action is at $\boldsymbol{T_i}$ and its corresponding end action is at $\boldsymbol{T_j}$.

(1) $\boldsymbol{Active(As,T_i)}\ \Rightarrow$ <PreCondition>[$\boldsymbol{T_i}$-] $\wedge$ <PostCondition>[$\boldsymbol{T_i}$+]

- A durative action activated at time point $\boldsymbol{T_i}$ implies its precondition at $\boldsymbol{T_{i-1}}$

(2) $\boldsymbol{Active(Ae,T_j)}\ \Rightarrow$ <PreCondition>[$\boldsymbol{T_i}$-] $\wedge$ <PostCondition>[$\boldsymbol{T_i}$+]

- For a durative action started at time point $\boldsymbol{T_i}$, all time points after $\boldsymbol{T_i}$ should be considered as a time point for its end action to happen.

(3)[35] $\boldsymbol{Active(Ae,T_j)}\ \Rightarrow\ \boldsymbol{Active(As,T_i)}\ \wedge\ [\ \boldsymbol{T_i}\ +\ ?\boldsymbol{duration(DA)_{Ti}}\ =\ \boldsymbol{T_j}\ ]$

- A causal link between the start action and its end action

(4) $\boldsymbol{Active}(\boldsymbol{As},\boldsymbol{T_i})\ \Rightarrow\ [\ \exists\ \boldsymbol{T_k}\ \ \boldsymbol{T_i} < \boldsymbol{T_k}\ <= \boldsymbol{T_n}\ \ \boldsymbol{Active}(\boldsymbol{Ae},\boldsymbol{T_k})\ ]$

- A durative action started in a plan should be finished in the plan.

[35] This axiom is based on the assumption that each time points have different time value.

Note that the constraints on duration are dealt as the same way as preconditions in start action or end action, depending on time annotator attached on them.

Axioms for Invariants (Persistent Conditions)

For a durative action which starts at T_i and ends at T_j, invariants checking is required for all T_k $T_i < T_k < T_j$. In general, invariant checking[36] at T_k includes

- <Invariants>[T_k-] to check continuous changes over (T_{k-1}, T_k)
- <Invariants>[T_k+] to check discrete changes at T_k.

In addition,

- $Value_{after}$ at starting time T_i

Two versions of encoding are distinguished, depending on (i) concurrency among the instances of the same durative action started at different time points and (ii) numeric-valued or interval-valued parameters to actions.

Encoding A handles the case where invariants (checking) can be shared among the same durative actions, regardless of the starting time points. This encoding can be used only if either (i) numeric- or interval-valued parameters are not allowed, (ii) numeric- or interval-valued parameters is allowed, but it is not allowed for the instances of the same ground action to be concurrent, or (iii) the invariants are not dependent on numeric-valued or interval-valued parameters. By encoding invariants checking at each point, the invariant conditions are checked as long as at least one of

[36] Note that these are to check values for continuously changing fluents; it suffices to check <Invariants>[T_k-] for propositional fluents. However, ADL features such as disjunctive precondition of propositional and numeric fluents makes difficult to seperate these checks.

instances of the same ground action *continues* at the point. The complexity of encoding for invariants of each ground action is: where, *n* is number of steps

$O(n)$ invariant checking + $O(n^3)$ *Continues* fluent generation

(5) $\textbf{\textit{Active}}(\textbf{\textit{Ae}},\textbf{\textit{T}}_j) \Rightarrow [\ \forall\ \textbf{\textit{T}}_k,\ \textbf{\textit{T}}_i < \textbf{\textit{T}}_k < \textbf{\textit{T}}_j \quad \textbf{\textit{Continues}}(\textbf{\textit{DA}},\textbf{\textit{T}}_k)\ ]$

- A causal link between the end action and persistency of the invariants

(5') $[\ \forall\ \textbf{\textit{T}}_k,\ \textbf{\textit{T}}_1 < \textbf{\textit{T}}_k < \textbf{\textit{T}}_n$

$\textbf{\textit{Continues}}(\textbf{\textit{DA}},\textbf{\textit{T}}_k) \Rightarrow$ <Invariants >[$\textbf{\textit{T}}_k$-] $\wedge$ <Invariants >[$\textbf{\textit{T}}_k$+]]

- Whenever any instantce of the durative action *continues* at the time point, invariants checking should be activated: <Invariants >[$\textbf{\textit{T}}_k$-] for checking with continuous change, <Invariants >[$\textbf{\textit{T}}_k$+] for checking with discrete change

(5'') $[\ \forall\ \textbf{\textit{T}}_k,\ \textbf{\textit{T}}_1 < \textbf{\textit{T}}_k < \textbf{\textit{T}}_n$

$\neg\ \textbf{\textit{Continues}}(\textbf{\textit{DA}},\textbf{\textit{T}}_{k-1}) \wedge \textbf{\textit{Continues}}(\textbf{\textit{DA}},\textbf{\textit{T}}_k) \Rightarrow$ <Invariants >[$\textbf{\textit{T}}_{k-1}$+]]

- At the time point where a start action is active, but no *continues*; discrete change on a numeric-valued fluent is necessary to be checked at such time points.
- Note that this axiom ensures that the invariant condition of a durative action which is over only two time points can be checked.

Encoding B handles the case where the ground action started at each time point should be distinguished. This is necessary to deal with invariant conditions that are dependent on numeric-valued or interval-valued parameters. The complexity of encoding for invariants of each ground action is

$\sum_{i=1}^{n-1}$ *O(n-i-2)* invariant checking + $\sum_{i=1}^{n-1}$ *O(n-i-2) Continues* fluent generation

= $O(n^2)$ invariant checking + $O(n^2)$ *Continues fluent generation*

(5) [$\forall$ k i < k < n [***Active***(***Ae***,T_{k+1}) $\vee$ … $\vee$ ***Active***(***Ae***,T_n)] $\Rightarrow$ ***Continues***(***DA***,T_k)]

- A causal link between the end action of durative action ***DA***[37] started at T_i and the invariant conditions checking; ***Ae*** is causally linked with ***As*** by Axiom (3) and (4)

(5') [$\forall$ k i < k < n ***Continues***(***DA***,T_k) $\Rightarrow$ <Invariants >[T_k-] $\wedge$ <Invariants >[T_k+]]

- The encoding for checking invariant conditions is necessary at all time points right after starting time point T_{i+1} to T_{n-1}

(5'') ***Active***(***As***,T_i) $\Rightarrow$ <Invariants >[T_i+]

- The checking at [T_k+] is necessary only at the start time, not any other points.

Axioms for Continuous Changes

As presented in equations (8) and (9) of Section 4.2.3,

(***increase*** <***nf***> (* ***#t*** <*RateOfChange*>)) in an action started at T_i and ended at T_j is encoded as:

$\forall$ T_k, T_i < T_k <= T_j ,

(6) ***Active***(***Ae***,T_i) $\Rightarrow$

[***NetContiChange(DA,nf,***T_{k-1},T_k***)*** = (T_k – T_{k-1}) * <*RateOfChange*>]

(6') $\neg$ ***Active***(***Ae***,T_i) $\Rightarrow$ [***NetContiChange(DA,nf,***T_{k-1},T_k***)*** = 0]

[37] Remember that ***Continues***(***DA***,T_k) distinguishes the instances of the same durative action started at different time points

- The end action acts like the representative for the durative action
- ***NetContiChange(DA*,*nf,T_{k-1},T_k)*** is accumulated in ***$Value_{before}$(nf,T_k)***
- ***DA*** distinguishes instances of the same durative action so that the net changes made by concurrent instances of the same durative action can be accumulated.

Axiom on Termination with No Time Slip

When a durative action continues at a time point, it is necessary to have a *constraint* to make sure that the time point is ending point of the durative action as soon as the value of an invariant condition becomes from ***TRUE*** to ***FALSE***. The formulation is given in 4.4.3.3.

Conditional Effect

In an instantaneous action ***a*** at time point T_i,

(***when*** <Antecedents> <Consequents>) is encoded as follows:

For each conjunct <Consequent-i> in <Consequents>,

$$[\ \textbf{\textit{Active}}(a, T_i) \ \wedge\ \text{<Antecedents>}[T_i\text{-}]\ ] \Rightarrow\ \text{<Consequent-i>}[T_i\text{+}]$$

In a durative action, each conjunct in <Antecedents> and <Consequents> are labeled by a time annotator. In particular, a conjunct in <Antecedents> may be labeled by *at start* or *over all* when it comes in the conditional effect of an end action: the conjunct time annotated by *at start* should be time-labeled as if it is a precondition[38] of start

[38] Refer Appendix C.3.

action of the durative action; likewise, a conjunct with annotator *over all* should be converted into conjunction at all time points over the period of the durative action.

4.3.4 Frame Axioms

In Level 3 & 4, frame axioms are over ***A***, ***As*** and ***Ae*** of a durative action, which make discrete changes: $\boldsymbol{a} = \boldsymbol{A} \cup \boldsymbol{As} \cup \boldsymbol{Ae}$.

Explanatory Frame Axioms for Propositional Fluents:

(7) $[\ \boldsymbol{Proposition(pf,T_{i-1})} \wedge \neg \boldsymbol{Proposition(pf,T_i)} \Rightarrow \exists\, \boldsymbol{a} \in \boldsymbol{NegEffect(pf)}\ \boldsymbol{Active(a,T_i)}\ ]$

(8) $[\ \neg \boldsymbol{Proposition(pf,T_{i-1})} \wedge \boldsymbol{Proposition(pf,T_i)} \Rightarrow \exists\, \boldsymbol{a} \in \boldsymbol{PosEffect(pf)}\ \boldsymbol{Active(a,T_i)}\ ]$

Explanatory Frame Axioms for Numeric-Valued Fluents:

(9) $[\ \forall\, \boldsymbol{a} \in \boldsymbol{Set(nf)}\ \neg \boldsymbol{Active}(\boldsymbol{a},\boldsymbol{T_i})\ ] \Rightarrow \boldsymbol{LinearDiscreteEq(nf,T_i)}$

- ***LinearDiscreteEq(nf,T_i)*** is imposed if no action with an *assign* statement on ***nf*** is active

(10) $\boldsymbol{Value_{before}(nf,T_i)} = \boldsymbol{Value_{after}(nf,T_{i-1})} + \sum_{a \in DA} \boldsymbol{NetContiChange(a,nf,T_{i-1},T_i)}$

(11) $\boldsymbol{Value_{after}(nf,T_i)} = \boldsymbol{Value_{before}(nf,T_i)} + \sum_{a} \boldsymbol{DiscreteChange(a,nf,T_i)}$

- (10) is linear equation of the sum of the previous[39] value at $\boldsymbol{T_{i-1}}$ and concurrent continuous changes over ($\boldsymbol{T_{i-1}}$,$\boldsymbol{T_i}$) on a numeric-valued fluent ***nf***.

[39] Means the value before discrete changes happen at the time point.

- (11) is the linear equation of simultaneous discrete changes at T_i on a numeric-valued fluent

Fluent as a Conditional Effect in Frame Axioms

In Axiom (7), (8) and (9), if the literal happens as a conditional effect, ***Active(a,T_i)*** should be replaced by

[<Antecedents> $\wedge$ ***Active(a,T_i)***]

to ensure that if the change in the value of the fluent is to be caused by ***a***, its antecedents also should be true.

4.3.5 Mutual Exclusivity

The non-interference rules defined in PDDL+ are in Appendix F.

Mutex rules are checked among ***A***, ***As***, and ***Ae***. Any two actions violating the non-interference rules should be pair-wise mutual exclusive:

(12) [$\neg$ ***Active***(a_i,T_i) $\vee$ $\neg$ ***Active***(a_j,T_i)]

Fluent as a Conditional Effect in Mutual Exclusivity

In Axiom (12), if the fluent leading to mutual exclusivity with the other action is a conditional effect in the action, ***Active(a,T_i)*** should be replaced by

[<Antecedents> $\wedge$ ***Active***(***a***,T_i)]

This should be applied to numeric-valued fluents as well as propositional fluents.

4.4 Representation of Temporal Model of Real-Time (Level 5)

Starting with recapitulation of syntax and semantics, axioms for initial and goal states, axioms for operators (process, event, action), frame axioms and axioms for mutual exclusions are presented in that order. The sample domain in Level 5 and its encoding are in Appendix D. The time-labeling convention is defined in Appendix B.

4.4.1 Syntax and Semantics

There are three components for this temporal model: processes capturing continuous changes, exogenous instantaneous events, and instantaneous actions. A process can be triggered and terminated by discrete changes or continuous changes: discrete changes made by atomic actions or events, continuous changes by active concurrent processes. An event can be triggered by discrete changes or continuous changes. Only actions are under plan executor's choice; events and processes are autonomous.

The semantics is based on Hybrid Automata Theory [Henzinger96]: A state carries continuous changes; discrete changes by events or actions cause state transition instantaneously:

time points: T_{k-1} T_k

states: $Q_1 \rightarrow \ldots \rightarrow Q_{2(k-1)} \rightarrow Q_{2k-1} \rightarrow Q_{2k} \rightarrow \ldots \rightarrow Q_{2n}$ **…**

Starting at the initial state Q_1, Q_{2k-1} k = 1,2,… stand for states carrying durations (i.e $T_k - T_{k-1}$), Q_{2k} k = 1,2,… stand for instantaneous state changes. Processes triggered by events or actions occurring in $Q_{2(k-1)}$ or ongoing processes in $Q_{2(k-1)-1}$ are going on in Q_{2k-1}, and then are terminated or discontinued by events or actions occurring in Q_{2k} or

terminated by the violation of invariant condition caused by concurrent continuous changes in $\mathbf{Q_{2k\text{-}1}}$.

The semantics naturally imposes the following two constraints: (i) *Continuity constraint* on continuously changing quantity to be preserved. That is, the continuous change from time T_1 to T_2 is the same as the sum of the change from T_1 to T_3, and change from T_3 to T_2, for any $T_3 \in [T_1, T_2]$. (ii) Processes and events should be triggered with *no slip of time* at the moment its preconditions are satisfied, in contrast to actions. In Section 4.4.2, we show how these constraints are handled in our encoding.

Concurrency

Actions can be concurrent with events as long as they do not interfere according to the non-interference rules of PDDL+. It is domain designer's responsibility that events that can be triggered at the same time are to be non-interfering.

The encoding given in this section is based on the assumption that two instances of the same process cannot be concurrent.

4.4.2 Representation of Initial State and Goal State

Initial State

Initial state is represented as effects of a dummy action with non-precondition.

Time-labeling: <Initial State>[$\boldsymbol{T_0}$+]

- A propositional fluent is time-labeled by $\boldsymbol{T_0}$

- An assignment to a numeric-valued fluent is time-labeled by $Value_{after}$ at T_0.

Goal States

In the last step, at least one action should happen, since subsequent events will not show up in a plan. Thus, we put constraints that (i) at least one of actions happens in the last step (ii) the goal state is either the last step or the step right before the last step.

Time-labeling at T_i:

- A propositional fluent is time-labeled by T_i
- A reference to a numeric-valued fluent is by $Value_{after}(nf,T_i)$

4.4.3 Representation of Operators

Axiom on Time Points

(1) $[\ T_0 = 0\ ] \wedge [\ \forall i,\ 0 < i <= n \quad T_i >= T_{i-1}\ ]$

- The next time point is greater or equal to the current time point.

4.4.3.1 Representation of Events

Axioms on Precondition and Effect:

(2) $Active(e,T_i) \Leftrightarrow$ <PreCondition>$[T_i-]$

(3) $Active(e,T_i) \Rightarrow$ <PostCondition>$[T_i+]$

Axioms on Triggering with No Slip of Time

There are two cases that an event is triggered:

- An event is triggered by discrete change(s), in which case linear arithmetic constraints may not cross equality (i.e. the inequality is satisfied, but equality is not.). Axiom (4) below ensures this case.
- An event is triggered by continuous changes, in which case *the earliest time* at which the precondition is satisfied should be captured by using the threshold value and continuity of linear constraints. Axiom (5) below ensures this case.

The first case :

(4) <Precondition>[T_{i-1}+] $\Rightarrow$ [$T_i = T_{i-1}$]

- If an event is triggered by actions or events (i.e. by discrete changes), the time value becomes the same as the previous time value. This ensures that an event is triggered by discrete change without slip of time.

The second case:

In ADL features, the precondition of an event (or a process) can be any expression of literals and arithmetic constraints connected by logical operators {*not, or, and, imply*}. To deal with ADL features, convert the precondition into disjunctive normal form:

$$[\, C_1 \vee \ldots \vee C_k \vee \ldots \vee C_p \,]$$

Define $C_k \equiv [\, P_{k1} \wedge \ldots \wedge P_{km} \wedge (F_{k(m+1)} <= d_{k(m+1)}) \wedge \ldots \wedge (F_{k(m+n)} <= d_{k(m+n)}) \,]$,

where, $P_{ij}(t)$ is a positive or negative literal; $(F_{lm} <= d_{lm})$ is a linear arithmetic constraint.

The earliest time that C_k can be true is the earliest time the last remaining conjunct becomes true (all other conjunction are already true); certainly the last conjunct is one of arithmetic constraints, not a propositional constraint, because the value of

propositional fluent does not change over the interval (T_{i-1}, T_i). The earliest time for C_k to become true can be found by the following axiom:

$[\wedge_j [\wedge_l P_{kl}[T_{i-1}] \wedge (F_{kj}[T_{i-1}+] > d_{kj}) \wedge [\wedge_{p \neq j} (F_{kp}[T_{i-1}+] > d_{kj}) \Rightarrow (F_{kp}[T_i-] <= d_{kj})] \Rightarrow (F_{kj}[T_i-] >= d_{kj})]]$

The earliest time the precondition becomes true is determined by

(5) $[\wedge_k [\wedge_j$

$[\wedge_l P_{kl}[T_{i-1}] \wedge (F_{kj}[T_{i-1}+] > d_{kj}) \wedge [\wedge_{p \neq j} (F_{kp}[T_{i-1}+] > d_{kj}) \Rightarrow (F_{kp}[T_i-] <= d_{kj})]$

$\Rightarrow (F_{kj}[T_i-] >= d_{kj})]]]$

Note that when an event is triggered by discrete changes, $F_{kj}[T_{i-1}+] = F_{kj}[T_i-]$ by Axiom (4); Thus, the left-side of the formula above becomes false always, and the formula is true, always.

4.4.3.2 Representation of Actions

(6) ***Active***$(a, T_i) \Rightarrow$ <PreCondition>$[T_i-] \wedge$ <PostCondition>$[T_i+]$

4.4.3.3 Representation of Process

Axiom on Precondition

(7) ***Active(pr,*** T_i***)*** $\Leftrightarrow$ [<PreCondition>$[T_i+] \wedge$ <PreCondition>$[T_{i+1}-]$]

- Axiom (7) ensures that ***pr*** activated at T_i is active over an interval starting at T_i. Note that the time point when ***pr*** terminates should be a significant time point, and the precondition holds after T_i and before T_{i+1}.

Axiom on Triggering by Continuous Changes with No Slip of Time

The same axiom used for events can be used to make a process to get triggered at the earliest time the precondition becomes true by continuous changes:

(8)[40] $[\wedge_k [\wedge_j$

$[\wedge_l P_{kl}[\boldsymbol{T_{i-1}}] \wedge (F_{kj}[\boldsymbol{T_{i-1}}+] > d_{kj}) \wedge [\wedge_{p \neq j} (F_{kp}[\boldsymbol{T_{i-1}}+] > d_{kj}) \Rightarrow (F_{kp}[\boldsymbol{T_i}-] <= d_{kj})]$

$\Rightarrow (F_{kj}[\boldsymbol{T_i}-] >= d_{kj})]]]$

Axiom on Termination with No Slip of Time

As soon as the precondition of a process becomes false, the process should be terminated. It suffices to ensure that this happens with continuous changes[41].

Precondition of a process can be in any logical expression of literals and arithmetic constraints connected by {*not, or, and, imply*}. To deal with ADL features, convert the precondition into a conjunctive normal form:

$$[D_1 \wedge \ldots \wedge D_k \wedge \ldots \wedge D_p]$$

Define $D_k \equiv [P_{k1} \vee \ldots \vee P_{km} \vee (F_{k(m+1)} <= d_{k(m+1)}) \vee \ldots \vee (F_{k(m+n)} <= d_{k(m+n)})]$,

where, P_{ij} is a positive or negative literal; $(F_{lm} <= d_{lm})$ is a linear arithmetic constraint.

[40] If a process is triggered by discrete changes, $F_{ij}[\boldsymbol{T_i}-] = F_{ij}[\boldsymbol{T_i}+]$; all antecedents are false, so the formula is true.

[41] Since it is assumed that two instances of the same process in the ground form are not allowed to be concurrent in this encoding, at each time point, each ground process is considered for triggering. The process triggered is either newly triggered one at the point, or resumed one of the ongoing process since (at least) the previous time point and discontinued by discrete changes at the point. It is unnecessary to have provision to ensure continuity of ongoing processes.

In order to make the process to be terminated as soon as the precondition (persistent conditions) is violated by continuous changes caused by concurrent processes, *the earliest time* at least one of D_k becomes false should be the next time point. The earliest time a disjunction D_k becomes false can be captured by the following formula (that is, the earliest time the last linear constraint remaining true, all others are already false, is about to become false)[42]:

$[\wedge\ j\ [\wedge l \neg P_{kl}[\boldsymbol{T_i}]\ ]\ \wedge\ (F_{kj}[\boldsymbol{T_i}\textbf{+}] < d_{kj})\ \wedge\ [\ \wedge\ p \neq j\ (F_{kp}[\boldsymbol{T_i}+] <= d_{kj}) => (F_{kp}[\boldsymbol{T_{i+1}}\textbf{-}] > d_{kj})\ ]$

$\Rightarrow (F_{kj}[\boldsymbol{T_{i+1}}\textbf{-}] <= d_{kj})\ ]\ ]$

The earliest time the precondition is about to become false can be captured by the following formula:

(9) $[\wedge k\ [\wedge j$

$[\wedge l \neg P_{kl}[\boldsymbol{T_i}]\ \wedge\ (F_{kj}[\boldsymbol{T_i}\textbf{+}] < d_{kj})\ \wedge\ [\ \wedge\ p \neq j\ (F_{kp}[\boldsymbol{T_i}+] <= d_{kj}) => (F_{kp}[\boldsymbol{T_{i+1}}\textbf{-}] > d_{kj})\ ]$

$\Rightarrow (F_{kj}[\boldsymbol{T_{i+1}}\textbf{-}] <= d_{kj})\ ]\ ]\ ]$

Axioms for Effects (Continuous Changes)

(*increase* <***nf***> (* ***#t*** <*RateOfChange*>))$_{\boldsymbol{Ti}}$ is encoded as:

(10) $\boldsymbol{Active}(\boldsymbol{pr},\boldsymbol{T_i}) \Rightarrow [\ \boldsymbol{NetContiChange(pr,nf,T_i,T_{i+1}) = (T_{i+1} - T_i)} * <RateOfChange>]$

(11) $\neg\ \boldsymbol{Active}(\boldsymbol{pr},\boldsymbol{T_i}) \Rightarrow [\ \boldsymbol{NetContiChange(pr,nf,T_i,T_{i+1})} = 0\ ]$

- ***NetContiChange(pr,nf,T_i,T_{i+1})*** is accumulated in the equation of ***$Value_{before}$(nf,T_{i+1})*** given in Section 4.2.2.

[42] Recall that if any of P_{ij} in D_k is true, D_k does not constrain the next time point.

4.4.4 Frame Axioms

In Level 5, the frame axioms are over actions and events. The same frame axioms defined in Section 4.3.4 can be applied with $\boldsymbol{a} = \boldsymbol{A} \cup \boldsymbol{E}$.

4.4.5 Mutual Exclusivity

Interference constraints (mutex rules) are checked among the actions and events that happen in the same time point: pairwise mutex checking between any action and any other action, and between any action and any event. It is the domain designer's responsibility to make sure that events happening at the same time point do not interfere. Any conflict between an action and an event is resolved in a way that gives priority to the event over the action. This is enforced by asserting necessary and sufficient conditions between occurrence of event and its precondition, given in Axiom (7).

The mutual exclusivity axioms in Section 4.3.5 can be applied to Level 5 as follows:

$\forall\ \boldsymbol{a_i} \in \boldsymbol{A}\quad \boldsymbol{a_j} \in \boldsymbol{A} \cup \boldsymbol{E}$ such that $\boldsymbol{a_i}$ and $\boldsymbol{a_j}$ are in mutex,

(12) $[\ \neg\, \boldsymbol{Active}(\boldsymbol{a_i}, \boldsymbol{T_i})\ \vee\ \neg\, \boldsymbol{Active}(\boldsymbol{a_j}, \boldsymbol{T_i})\ ]$

4.5 Extensions for Metric Quantities

In this section, we extend the uses of functions in PDDL+ to support metric-quantities[43] with the *capacity* feature and metric quantities of interval type. The motivation behind extending numeric-valued fluents with *capacity f*eature is that resources[44] in planning can be usually represented with metric quantities with capacity, but PDDL+ has limitations[45] in simulating the uses of resources. The interval type is to represent metric quantities with lower and upper limits and to allow operations based on Allen's interval relations [Allen83].

Among the definitions of resource [LongFox00][46] [Bedrax03] [Smith97], let us adopt the definition given by Laborie [Laborie02]: A *resource* is any substance or (set of) object(s) whose cost or available quantity (capacity) induce some constraint on the operations that *use* it. Naturally, resources in planning are closely connected with time and concurrency: the reason why resources have received little attention in the planning community is that most early planning formalisms could not handle

[43] Notice that it is not difficult to extend this encoding for resources or metric quantities defined over **R** to an encoding for multi-dimensional resource or quantities over $\mathbf{R}^n$ [Smith97], such as a cargo ship with capacity of weight and volume, storage space with capacity of volume, elevator with capacity of persons and weight, or even simple geometric shapes used in user interface design. The idea is component-wise checking of capacity and propagation, if necessary.

[44] There are proposals [Bedrax03] [McDermott03-c] to formalize a resource and to extend PDDL with a resource "explicitly" declared as an object with rich features in a style of object oriented programming.

[45] PDD+ cannot at all represent the use of multi-capacity resources by an instantaneous action. On the other hand, in a durative action, multi-capacity resource can be simulated by allocating at the starting point, capacity-checking as invariants, and deallocating at the ending point. The tricky part is that extent of invariants in a durative action of PDDL+ does not include the end points of the action; thus, the same capacity checking should be done in the precondition of the ending action. Then, *no moving targets rule* will not allow an action updating the resource to be concurrent at the end points of the durative action. The restriction on concurrent uses at the end points happens with representation of any type of metric resources.

concurrent actions. Moreover, in the context of continuous time (e.g. durative actions), the management of resources become more complicated.

In this section, we present encoding schemes for a class of *sharable*[47]*, reusable resources*[48] *that are used exclusively during an action, but are not affected by the action in any other way*[49]:

- Multi-capacity resources (discrete, sharable, reusable resources): these are represented as a numeric-valued fluent with the capacity feature.

  e.g. identical machines in a factory or number of personnel

- Partitioned interval resources (continuous, sharable, reusable resource) whose concurrent "uses" mean disjoint subintervals: these are represented as an interval-valued fluent with capacity feature.

  e.g. main memory (RAM) allocated to concurrent processes

Also, we present an encoding scheme for a metric quantity represented in terms of interval used as a parameter to action as well as a function type

e.g. allocation of your time in interleaved concurrency to various activities such as emailing, latexing and web browsing, which may be related to tasks with deadlines

---

[46] [LongFox00] defines that being resource or not is dependent on the context it is used.
[47] "Sharable" resource is allowed to be used simultaneously, within its capacity.
[48] Generally, the management of sharable resource is difficult in other planning frameworks [Smith00-b], because it involves identifying potential resource conflicts which grow exponentially with the number of actions sharing it.
[49] Notice that in this sense, in the Gripper domain, the number of balls or number of grippers for a robot is not a sharable, reusable resource, because the effect is propagated beyond the duration of the action (pick up, put down), although those are resource in a general sense in planning. Other types of resource such as continuous consumable/producible resource can be dealt with in the same way as temporal metric encodings given in Section 4.3 and 4.4, with the explicit identifications of being resources, like "consume" or "produce" statements.

First, we review what features are supported to represent metric quantities in PDDL+, and define their syntactic extensions to support multi-capacity resources and intervals. In Section 4.5.1, the encoding for multi-capacity resources is presented. The encoding of resources based on intervals ("*partitioned interval resources*") is presented in Section 4.5.2. The encoding for interval-valued fluents is given in Appendix E.

Representation of Metric Quantities in PDDL+

In PDDL+, metric quantities are represented as parameterized functions, arithmetic comparison operators (>,>=,<,<=,=) between functional expressions, and updating statements (*increase*, *decrease*, *assign*) of the value of function. Metric quantities with capacity such as multi-capacity resources can be represented "implicitly" through parameterized functions and arithmetic constraints to check its capacity while they are in use. Apart from the limitations[45] resulting from these means and semantics adopted in PDDL+, the general implicit representation of resources makes it difficult to do reasoning about resources. Most techniques for reasoning about resources require explicit identification of the activities using the resource.

Numeric-valued types are not allowed as a parameter to action in PDDL+.

Extension of Function Definition with Interval Type and Features

(**:*functions*** { <Def-of-Function> {***capacity:*** <Constant>} **--** <Type-of-Funct> }$^{+}$)

<Def-of-Function> is a function definition as defined in PDDL+
<Constant> = <Number> | <Interval>

<Interval> = **[** <Number> **,** <Number> **]**

<Type-of-Funct> = { positive | negative } { integer | real } { float | fluent }

| { interval } { float | fluent }

The nature of sharable, reusable resources as defined in this section is that they may be allocated at the beginning of the action and released at the end of action. ***use*** subsumes[50] both the allocation and the release.

(***use*** *?resource ?quantity*)

- This is used as an effect in atomic or durative actions(events, processes)
- *?resource* is a parameterized function of either numeric type or interval type
- *?quantity*[51] is an arithmetic expression of constants, variables from numeric-valued / interval-valued parameter of action, functions which are either float or fluent and duration variables of durative actions.
- The value of arithmetic expression for *?quantity* is evaluated in terms of allocation time, if it contains fluents
- The quantity on request can be duration-dependent (for instance, number of web designers for a project running for *a week*, given available designers and man-power-per-day?), but not time-dependent in the sense that the whole amount should be allocated at the beginning of the action, not gradually needed over time.

[50] We turn around the limitation in simulating resources as numeric-valued fluents, that is, freeing resources of the “no moving targets” rule.

[51] All the known reasoning techniques of resources deal only with known (constant) quantities on request!

4.5.1 Representation of Multi-Capacity Resources

There are several ways to encode multi-capacity resources in the SAT-based planning framework:

(1) The simplest way is to name objects individually; this approach, however, generates search space with evident symmetry [Rintanen00] [FoxLong99] [FoxLong02-e]. It works reasonably well with a small number of objects that are known at the encoding stage.

(2) If it is unnecessary to care about the identity of each object, it can be encoded as *n*-ary mutex relation at each step: $\neg(A_1 \wedge A_2 \wedge \ldots \wedge A_n)$ in ${}_{\mathbf{a}}\mathbf{C}_{\mathbf{n}}$ ways, assuming that **a** is the number of actions using a unit of the resource, and **n** is greater than the capacity of the resource and less than or equal to **a**. ${}_{\mathbf{a}}\mathbf{C}_{\mathbf{n}}$ grows exponentially with the number of actions involved. This can be extended to allow more than one quantity on request. However, the fundamental assumption of this approach is that the quantities requested and capacity should be known (constant) at the encoding stage.

(3) If the identity of each object does not matter, the objects can collectively be encoded as a numeric-valued fluent with capacity-checking done at each time point, rather than represented in the *n*-ary mutex relations. The advantages of this approach are as follows:

- There is no more symmetry in the search space caused by uses of this resource.
- Quantity requested or capacity does not have to be known (constant) at encoding stage.

- By allowing the capacity to be variable, it can support the dynamic creation[52] of objects of the resource type in the domains where the identity of objects does not matter.

In this section, an encoding scheme for multi-capacity resources based on the third approach is presented with an extension of richer expressiveness such as variable quantity.

Example: Sharing multi-capacity resources among durative actions

A software company has some amount of software under development and a finite number of software engineers. As usual, stages of software development are composed of design, implementation and testing, among which obvious precedence relations exist. Each stage has its own estimated manpower (let's say in a day unit) How should engineers be allocated to projects?

```
;; This action starts with the available engineers and will last until its own
;; manpower meets. Likewise, design and testing can be represented
(:durative-action          implementation
 :parameters     (?s – software   ?e – real)
 :duration       (=  (* ?duration ?e)  (manpower ?s IMPLEMENTATION))
 :condition      (and     (at start (designed ?s))  (at start (not (implemented ?s)))
                          (overall (not (implemented ?s)))
                          (at end (not (implemented ?s))))
 :effect         (and     (at end (implemented ?s))
                          (use (engineers) ?e))
 )

;; to consider the number of engineers taking a day off
(:durative-action          OneDayOff
 :parameters     ()
 :duration       (= ?duration 1)
 :precondition   ()
 :effect         (use (engineers) 1)
)
```

[52] The Setters Domain used in IPC3 is such an example which needs the dynamic creation of objects.

Representation of Function of Multi-capacity Resource

The function defined as a numeric type with the capacity feature is encoded with two variables at each time point:

- ***Value***$_{before}$***(?resource,***T_i***)*** is the level of the resource at a time point T_i before any discrete change is made at T_i
- ***Value***$_{after}$***(?resource,***T_i***)*** is the level of the resource at a time point T_i after any discrete change is made at T_i.

Additionally, capacity checking at each time point is required. This is given in (9) ~ (12) below.

Notice that if the multi-capacity resource is shared only among instantaneous actions exclusively or among durative actions, one variable at each time point is enough. Two variables are needed only to be shared[53] among durative actions and instantaneous actions. The encoding of multi-capacity resources presented below can deal with sharing among any kind of actions.

In an Instantaneous Action at T_i,

Each ***use*** statement in precondition of an instantaneous action at a time point can be encoded with two variables:

- *Quantity*$_{before}$*(?resource,a,*T_i*)* is the amount allocated to the action at T_i on ?resource.
- *Quantity*$_{after}$*(?resource,a,*T_i*)* is the amount released by the action at T_i on ?resource.

(***use*** ?resource ?quantity) is encoded as follows:

(1) $\textbf{\textit{Active}}(\boldsymbol{a},\boldsymbol{T_i}) \Leftrightarrow [\ \textit{Quantity}_{\textit{before}}(?\textit{resource},a,T_i) = ?\text{quantity}\ ]$

(2) $\textbf{\textit{Active}}(\boldsymbol{a},\boldsymbol{T_i}) \Leftrightarrow [\ \textit{Quantity}_{\textit{after}}(?\textit{resource},a,T_i) = -?\text{quantity}\ ]$

(3) $[\ [\ \textit{Quantity}_{\textit{before}}(?\textit{resource},a,T_i) = ?\text{quantity}\ ] \oplus$

$[\ \textit{Quantity}_{\textit{before}}(?\textit{resource},a,T_i) = 0]\ ]$

(4) $[\ [\ \textit{Quantity}_{\textit{after}}(?\textit{resource},a,T_i) = -?\text{quantity}\ ] \oplus$

$[\ \textit{Quantity}_{\textit{after}}(?\textit{resource},a,T_i) = 0]\ ]$

- (1) and (3) ensure that allocation of the resource by the ***use*** statement is valid only when the action containing the statement is active; otherwise, the quantity of allocation by the statement is 0.
- (2) and (4) ensure the same as above on deallocation
- $\textit{Quantity}_{\textit{before}}(?\textit{resource},a,T_i)$ is accumulated in (9)
- $\textit{Quantity}_{\textit{after}}(?\textit{resource},a,T_i)$ is accumulated in (10)

Note that if the resource is shared among instantaneous actions, only one variable is used, and the total of uses is checked at each time point (i.e. checking allocation only).

<u>In a Durative Action over</u> [?start,?end],

Each ***use*** statement in invariants of a durative action can be encoded with two variables[54]:

- $\textit{Quantity}_{\textit{after}}(?\textit{resource},As,?\textit{start})$, is the amount allocated to the durative action at *?start* on ?resource.

---

[54] Notice that the difference in (1) and (5), likewise (2) and (6): "*after*" and "*before*" are switched.

- $Quantity_{before}(?resource,Ae,?end)$ is the amount deallocated from the durative action at *?end* on ?resource.

(***use*** ?resource ?quantity) is encoded as follows:

(5) ***Active***(***As***,***?start***) ⇔ [$Quantity_{after}(?resource,As,?start)$ **=** ?quantity]

(6) ***Active***(***Ae***,***?end***) ⇔ [$Quantity_{before}(?resource,Ae,?end)$ **=** - ?quantity]

(7) [[$Quantity_{after}(?resource,As,?start_i)$ **=** ?quantity] ⊕

[$Quantity_{after}(?resource,As,?start)$ **=** 0]]

(8) [[$Quantity_{before}(?resource,Ae,?end)$ **=** - ?quantity] ⊕

[$Quantity_{before}(?resource,Ae,?end)$ **=** 0]]

- ***As*** for starting action of a durative action; ***Ae*** for ending action for a durative action
- (5) and (7) ensure that allocation of the resource by the ***use*** statement is valid only when the starting action of the durative action containing the statement is active; otherwise, the quantity of allocation by the statement is 0 in (9).
- Since that ***As*** and ***Ae*** are causally linked in a durative action representation ((6) in Section 4.3.3), (6) and (8) ensure deallocation by quantity is valid only when the ending action is active at ?end and the starting action is active at ?start.. Otherwise, the deallocation is 0 in (10).
- $Quantity_{before}(?resource,As,?start)$ is accumulated in (9).
- $Quantity_{after}(?resource,Ae,?end)$ is accumulated in (10).

Constraints at each time point:

(9) $\boldsymbol{Value_{before}(?resource, T_i)}$ =

$$\boldsymbol{Value_{after}(?resource, T_{i-1})} + \sum Quantity_{before}(?resource,a,T_i)$$

(10) $\boldsymbol{Value_{after}(?resource,T_i)} =$

$$\boldsymbol{Value_{before}(?resource, T_i)} + \sum Quantity_{after}(?resource,a,T_i)$$

(11) for all $\boldsymbol{T_i}$, 0 <= $\boldsymbol{Value_{before}(?resource, T_i)}$ <= Capacity(?resource)

(12) for all $\boldsymbol{T_i}$, 0 <= $\boldsymbol{Value_{after}(?resource, T_i)}$ <= Capacity(?resource)

- (9) accumulates all (de)allocations at the "*before*" value on the previous level of the resource, and (11) checks for capacity limit. (10) and (12) do the same for the "*after*" value
- Notice that this encoding can handle the cases that the resource is shared among instantaneous actions and durative actions, as in Level 4 of PDDL+. If an instantaneous action is concurrent with the starting action of a durative action, and both actions are accessing the resource, the "after" value at *?start* is changed by deallocation by the instantaneous action as well as by allocation by the durative action.

4.5.2 Representation of Interval Type

The ways "interval" is used in Knowledge Representation can generally be classified into two categories: One is representation of uncertainty (fuzziness), in which case reasoning usually involves Interval Arithmetic [Funge99] [OlderVellino90] [Baioletti03]. The other is representation of continuous metric quantity (e.g. a period of time or line segment), not the range of possibility. Here we use interval to represent continuous metric quantities with lower and upper limits.

In this section we present encoding schemes for the uses of interval as (i) a parameter to an action, (ii) a function type using operations based on Allen's interval relations, and (iii) shared reusable interval resources.

Representation of Interval as a Parameter[55] to Action

A parameter of interval type in a ground action at time point T_i can be encoded with two variables, $\boldsymbol{I_{start}}(?p,a,T_i)$ and $\boldsymbol{I_{end}}(?p,a,T_i)$ to represent the two end points of the interval. This encoding can be used for both an instantaneous action (or in discrete time) and a durative action, since the parameter acts like *float* within the action. However, if concurrency among the instances of the same durative action started at different time points is allowed, invariants checking can not be shared among the instances. Thus, the cost of using numeric-/interval-valued parameter is high in SAT encoding.

Representation of Function of Interval Type

A function defined as interval *float* can be encoded with two variables, $\boldsymbol{I_{start}}(?i)$ and $\boldsymbol{I_{end}}(?i)$ to represent the two end points of the interval.

[55] As mentioned in Section 4.1.1, we reinstated the real-valued parameter to action in the TM-LPSAT. The idea of encoding is to introduce a real-valued variable for each numeric-valued parameter in a ground action. No additional care for the case with durative actions in continuous time is necessary, since the parameter is like *float* local to the grounded action. However, if instances of the same ground action started at different time points are allowed to be concurrent, they needs to be distinguished with different variables for the parameter. Consequently, invariants checking cannot be shared among the same actions, as mentioned in Section 4.3.3.

A function defined as interval *fluent* can be encoded with two variables, $\boldsymbol{l_{start}}(?i,T_i)$ and $\boldsymbol{l_{end}}(?i,T_i)$ to represent the two end points of the interval at each time point.

Operations on Interval-valued fluents are defined based on Allen's thirteen interval relations [Allen83] [Davis90]. The encoding is straightforward and lengthy and is therefore given in Appendix E.

4.5.2.1 Representation of Partitioned Interval Resources

We define a "Partitioned Interval Resource" as a sharable, reusable continuous resource[56] with a lower and upper bounds, of which concurrent ***uses*** mean allocation of disjoint subintervals; the allocated subinterval (i.e. position as well as quantity) to an action is preserved during the action.

The encoding given here is based on the assumption that the interval type is *float*, whose initial value is given in the function definition or problem definition. Function defined as interval *float* can be encoded with two variables, $\boldsymbol{l_{start}}(?resource)$ and $\boldsymbol{l_{end}}(?resource)$ to represent two end points of the interval.

In order to deal with interval *fluent*, we need to extend with operations to change the interval boundary (*increase-low*, *decrease-high*). Also, the variables for boundaries should be defined at each time point: $\boldsymbol{l_{start}}(?resource,T_i)$ and $\boldsymbol{l_{end}}(?resource,T_i)$.

[56] The capacity of the resource can be constant or variable. An example of this is main memory space allocated to concurrent processes. In contrast to general sharable, reusable continuous resource (e.g. battery) of which the use means allocation of quantity only, this is a special class of sharable, reusable continuous resources.

Note that the encoding given below is based on the assumption that the resource is shared among either instantaneous actions or durative actions, but not a combination of the two. To share among the actions of the two types, use "*before*" and "*after*" values at each time point, as done with encoding for multi-capacity resource.

Example: RAM allocation to concurrent processes:

```
;; (RAM-space)  : capacity [min,max] – interval float
;; In OS that uses variable length partitions, each job is allocated a consecutive
;; segment of RAM
(:durative-actionExecuteJob
 :parameters     (?p – process  ?memory – real ?time – real)
 :duration       (<= ?duration ?time)
 :condition      (and    (at start (not (active ?p)))
                         (over all (active ?p))
                         (at end (active ?p)))
 :effect         (and    (at start (active ?p))  (at end (not (active ?p)))
                         (use (RAM-space)  ?memory))
)
```

In Instantaneous Action at T_i,

Each ***use*** statement in a ground action is assigned two variables to represent the two end points of the subinterval: $UI_{start}(?r,a,T_i)$, $UI_{end}(?r,a,T_i)$.

(***use*** ?resource ?quantity) is encoded as follows:

(1) $[\ UI_{start}(?r,a,T_i) >= I_{start}(?r)\ ]$

(2) $[\ UI_{end}(?r,a,T_i) <= I_{end}(?r)\ ]$

(3) $[\ UI_{end}(?r,a,T_i) = UI_{start}(?r,a,T_i) = I_{start}(?r)\ ]$

$\oplus\ [\ UI_{end}(?r,a,T_i) - UI_{start}(?r,a,T_i) = ?quantity\ ]$

(4) $Active(a,T_i) \Leftrightarrow [\ UI_{end}(?r,a,T_i) - UI_{start}(?r,a,T_i) = ?quantity\ ]$

- (1) and (2) ensure that the subinterval should be contained in the interval resource
- (3) ensures that at any time point the subinterval for ***use*** is either defined as nonempty, exclusively or zero interval by setting end points to the beginning point of the interval resource
- (4) guarantees that a subinterval for the ***use*** is allocated only if the action containing the statement is active at $\boldsymbol{T_i}$

Constraints to be checked at each time point:

(5) $\forall UI \ \forall UI'$ [UI_{start} (?r,T_i) >= UI'_{end} (?r,T_i)] V [UI_{end} (?r,T_i) <= UI'_{start} (?r,T_i)]

- *UI* and *UI'* are all ***use***s in instantaneous actions at $\boldsymbol{T_i}$
- Pairwise checking if two intervals are not overlapped

In Durative Action over [?start, ?end],

Each ***use*** statement in a grounded durative action[57] is assigned two variables to represent the two end points of the interval at each time point: UI_{start} (?r,da, T_i) , UI_{end}(?r,da, T_i)

(***use*** ?resource ?quantity) is encoded as follows:

(1) [UI_{start} (?r,da, T_i) >= I_{start} (?r)]

(2) [UI_{end} (?r,da, T_i) <= I_{end} (?r)]

[57] It does not seem that encoding the uses of interval type in durative action is attractive in terms of size, which is the consequence of the way durative actions can be handled in SAT-

(3) [UI_{end}(?r,da, T_i) = UI_{start}(?r,da, T_i) = I_{start} (?r)] $\oplus$

[UI_{end} (?r,da, T_i) - UI_{start}(?r,da, T_i) = ?quantity]

(4) ***Active***(***As***,?start) $\Leftrightarrow$ [UI_{end}(?r,da,?start) - UI_{start} (?r,da,?start) = ?quantity]

(5) $\forall Tj \; ? start \prec Tj \prec ? end$,

[***Active***(***Ae***, ?end) $\Leftrightarrow$

[UI_{start}(?r,da,T_j) = UI_{start}(?r,da,T_{j-1}) ^ UI_{end}(?r,da, T_j) = UI_{end}(?r,da, T_{j-1})]]

- (1) and (2) ensure that the subinterval should be contained in the interval of the resource
- (3) ensures that at any time point the subinterval for ***use*** is either defined as nonempty, exclusively zero interval by setting end points to the beginning point of the interval resource
- (4) allocates a subinterval of size of ?quantity if and only if the durative action starts.
- (5) propagates the subinterval assigned at ?start over the time points in (?start, ?end), and the propagation is activated if and only if the durative action is in the plan.
- ***Active***(***As***,*?start*) and ***Active***(***Ae***, ?end) are causally linked in durative action representation ((6) in Section 4.3.3) so that (4) and (5) ensure that the subinterval is preserved during the period of the durative action.

<u>Constraints to be checked at each time point</u>:

---

based planning. The issue, then, is how to optimize (prune) the levels each durative action may be able to start or/and to end.

(6) $\forall UI \quad \forall UI'$,

[UI_{start} (?r,da,T_i) >= UI'_{end} (?r,da',T_i)] V [UI_{end} (?r,da,T_i) <= UI'_{start} (?r,da,T_i)]

- Pair-wise checking if two intervals are overlapped
- UI and UI' comprise all subintervals defined in any action in any time before the current time point, inclusively.

Chapter 5 Implementation of TM-LPSAT

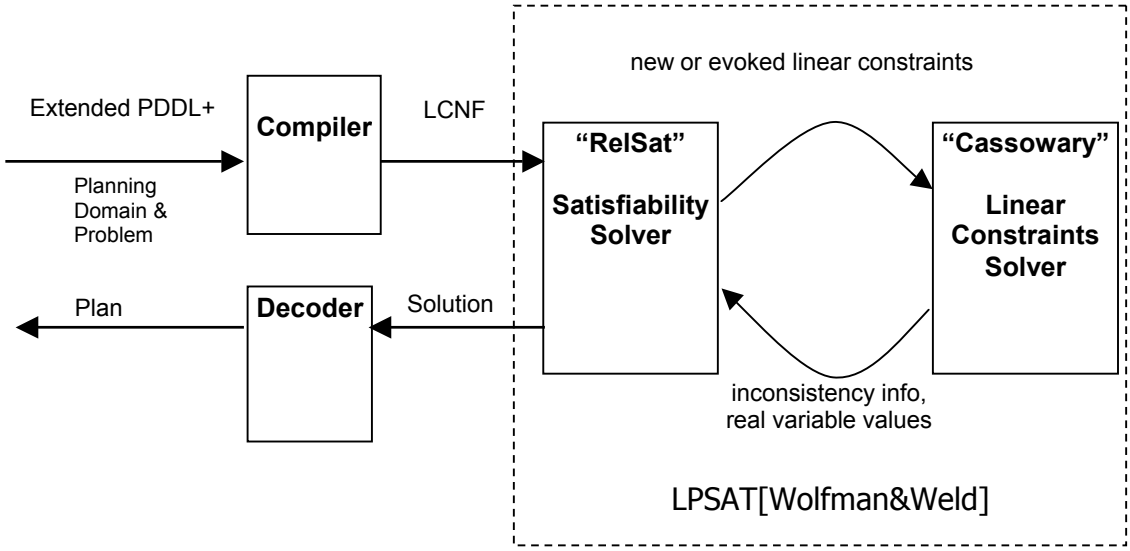

Figure 3: Architecture of the TM-LPSAT Planning Framework[58]

5.1 LCNF Compiler

The TM-LPSAT compiler translates PDDL+ descriptions of a domain and a problem into a CNF formula over linear (in)equalities and propositional fluents. The current implementation of PDDL+ and our encoding schemes can deal with the ADL features, including types, negative preconditions, disjunctive preconditions, equality, quantified-preconditions, and conditional effects.

The objective of current implementation of the TM-LPSAT is to test the feasibility of the LPSAT approach to temporal and metric planning in continuous time.

Except pruning of the search space in a Graphplan style, collecting the time points feasible for goal states in the durative temporal models, and optionally Crawford's COMPACT[59] simplifier over CNF formulas, no other optimization on compiler has been tried.

5.2 LPSAT Engine

Between MathSAT [Sebastiani01] [Audemard02-b] and LPSAT [Wolfman00] [Wolfman99], two LP+SAT engines which were available to us at the time of research, we have chosen (actually, no choice) LPSAT for verification of our encodings, simply because of the expressiveness of linear constraints that the arithmetic constraint solver integrated into the engine can support. The MathSAT can handle only limited forms of linear inequalities (variable multiplied by constant, difference of two variables in constant, etc.) which are not expressive enough for most domains with continuous changes in planning; The LPSAT accepts linear inequalities that fit better for our purpose. MathSAT, however, is robust[60] and supports optimization techniques of state-of-the-art.

LPSAT by Wolfman and Weld

---

[58] This figure is the same as the figure given in page 10.
[59] It includes unit propagation, subsumption and pure literal elimination.
[60] The capacity of the current version, they say, are of 20K Boolean variables, 300K clauses, 1K real variables and 5K atoms.

The LPSAT engine is composed of RelSAT [Bayardo97] and Cassowary [Borning98]: RelSAT is a variant of DPLL-based systematic solver using learning and back jumping techniques, which led the performance of systematic solvers to the level competitive to that of stochastic solvers. Cassowary is a linear arithmetic constraint solver developed for user interface design, which is an implementation of an incremental variant of Simplex method. Techniques to optimize its performance through communication between the two components are explored, such as backjumping and learning based on minimal conflict set calculated when the constraint set is inconsistent and splitting heuristic for triggers for constraints.

The engine works as follows: The SAT solver is responsible for generating assignment of truth values which includes triggers (Boolean variables) for arithmetic constraints. Once an assignment has been found satisfiable propositionally, the set of arithmetic constraints whose triggers are *true* in the assignment is checked for consistency by the LP solver. If the set is consistent, the assignment is a model for the axioms of the given planning problem. If inconsistent, the LP solver gives calculated information of inconsistency (minimal conflict set) back to SAT solver.

In order to integrate their LPSAT into our TM-LPSAT, a number of adjustments had to be made to the CNF formulas generated by the TM-LPSAT compiler:

- The form of linear constraints accepted by Cassowary is equality or non-strict inequality; strict inequality is transformed into non-strict inequality by adding very small ε. In practice, ε needs be tuned to be small enough not to make any change in the value of any quantity used in the given domain.

In Cassowary[61] imbedded in the LPSAT, all numbers are assumed to be real number unless positive constraint is given; a function whose value is be positive needs additional constraints on being positive at every time point.

- Negated Arithmetic Constraint: Propositional Negation vs. Mathematical Negation

  The following triggering mechanism is adopted in LPSAT: A trigger is assigned for each positive constraint. When the trigger is set to *true*, the positive constraint is added to the set of linear constraints to be checked for consistency. When the trigger is set to *false* (theoretically unnecessary[62]), its negated constraint is automatically true. That is, negated arithmetic constraints in the LPSAT are interpreted *propositionally*[63].

  Propositional negation works correctly if only positive forms of a constraint occur in the input formulas. Otherwise, if both negated forms and positive forms occur in the formulas, as is in our encodings, negated constraints should be interpreted as *mathematical negation*[64]. Also, to deal with negative precondition in ADL features, mathematical negation should be adopted.

  not (f(t) = d) is converted into ((f(t) >= d + ε) V (f(t) + ε <= d)
  not (f(t) <= d) is converted into (f(t) >= d + ε)
  not (f(t) < d) is converted into (f(t) >= d)
  not (f(t) >= d) is converted into (f(t) + ε <= d)
  not (f(t) > d) is converted into (f(t) <= d)

[61] In contrast to standard Simplex method, in which variables are assumed to be non-negative real numbers.
[62] The property can be generalized such that for a constraint occurring in only positive form (or, negative form) in the formulas, assignment to *false* (or *true*) is not necessary. It can be useful for pruning search tree, especially in dealing with equality, which requires a disjunction of two strict inequalities [Sebastiani01].
[63] Consequently, the LPSAT cannot handle negated precondition.
[64] In MathSAT, mathematical negation is adopted so that it can deal with negated constraints in the input formulas correctly.

Chapter 6 Experiments

Neither the process of compilation nor encoding is optimized [65] in the current implementation of the TM-LPSAT compiler. Thus, rather than comparing the performance of TM-LPSAT with the state-of-the-art planners[66], we experiment on our encodings with different SAT-based constraint solvers. For the problems of substantial size, the total time is dominated by the time required to search for the solution. Thus, our interest is mainly to observe whether in terms of *the searching time* TM-LPSAT can be comparable to the state-of-the-art planners based on different planning frameworks.

In Section 6.1, we give an overview of the different constraint solvers dealing with propositional combinations of Boolean variables and linear arithmetic constraints. In Section 6.2, we present and discuss experimental results.

6.1 SAT-Based Arithmetic Constraint Solvers: LPSAT and MathSAT

LPSAT [Wolfman99] [Wolfman00] and two versions of MathSAT were available for testing our encodings. The detail of LPSAT was dealt in Chapter 5. We give a brief description of the MathSAT family.

[65] Refer Section 5.1 for what can be optimized.
[66] Almost all domain-independent planners can handle neither planning with continuous change nor real-time temporal planning with, which are the main features of our TM-LPSAT.

MathSAT [Sebastiani01] [Audemard02-b]

Two versions of the MathSAT family are available for our testing: one, we call it "MathSAT03", is the version dealing with difference constraints, that is, it cannot handle more than two variables in a linear equation and any arithmetic operation except the difference of two variables. The other, we call it "MathSAT04", is an extended version of MathSAT03 dealing with any arithmetic operation except division and more than two variables in an equation. The form of negated equalities in the extended linear constraints is under development; specifically, back-jumping on arithmetic constraints and learning do not work at the moment.

The architecture of MathSAT is more sophisticated than LPSAT: it is composed of five stratifying layers:

- L_0: The DPLL-based Boolean solver

  It takes care of only propositional connectives; arithmetic constraints are abstracted as propositional ones.
- L_1: Elimination of equalities

  It propagates equalities and clusters variables.
- L_2: Minimal path with negative cycle detection

  If only atoms of "$V_i - V_j$ {>,<.>=,<=} *Constant*" remain, the problem is solved by a minimum path algorithm with cycle detection, a variant of Bellman-Ford algorithm.
- L_3: Linear Programming simplex method
- L_4: Handling negated equalities

A layer L_i is called by L_{i-1} to refine a partial solution of the problem. This decomposed architecture allows exploiting specialized efficient algorithms to deal with each layer and as well as allowing lazy evaluation so that difficult parts of problems (such as LP solver) are not called until it is necessary.

The MathSAT family based on a SIM SAT solver and lpsolver support the following features:

- Dependency among variables can be utilized: only a subset of independent variables is used at branching points.
- It supports heuristics at branching points which are proven to be effective in DPLL-based SAT solvers [Armando02]: ten options of heuristics[67].
- It supports several optimization techniques for arithmetic constraints and Boolean formulas, such as early pruning and static learning making similar effects as learning on the run.
- The known techniques such as back-jumping and learning on arithmetic constraints as well as propositional variables are included in “MathSAT03”. On the other hand, as mentioned before, only back-jumping based on conflicts among numeric variables and learning does works for “MathSAT04” for now.

[67] JWHeur (Jeroslow and Wang heuristics), 2JWHeur (2 side Jeroslow and Wang heuristics), SatoHeur (heuristics used in Sato 3.2 solver), SatzHeur (heuristics used in Satz solver), BoehmHeur (heuristics used in Boehm), MomsHeur (Maximum Occurrences in clause of Minimum Size heuristics), RelsatHeur (heuristics used in Relsat 2.0), UnitieHeur (heuristics

6.2 Test Results and Discussion

We have separately tested *metric planning*, *temporal planning*, and *temporal metric planning*, since characteristics of constraints specific to different planning may make a difference in performance with features of different constraint solvers. We use these terms in the following sense: *Metric planning* is reasoning about discrete changes of numeric or propositional fluents that are made by actions over discrete time. *Temporal planning* is reasoning about changes of propositional fluents that are made by actions or durative actions over real-valued time. *Temporal metric planning* is reasoning about changes of propositional fluents and discrete continuous changes of numeric fluents that are made by operators (actions, events, durative actions, or processes) over continuous time.

Preparation of Problem Domains and Instances

For the metric planning with discrete changes and temporal planning, domains and problems from IPC3 are used. These are originally defined as optimization problems. However, TM-LPSAT cannot optimize plans, the problems are converted to satisfaction problems in a way that an optimized value[68] (*plan quality*) generated by

based on unit propagation), RndHeur (it randomly selects the next proposition to assign), and UsrHeur (it asks the next proposition to assign to user).

[68] All domains and Problems used in our testing are to minimize the given plan metric.

LPG planner[69] [Gerevini03], one of the best planners competed in the IPC3, is used as a condition of the goal state.

The problem domain used for temporal metric planning is a variation of Bathtub domain introduced in Chapter 2: there are processes of "filling with a hot tap" and "filling with a cold tap" and a process of "draining", which can determine the level of the bathtub at the same time. Also it includes an event of "Flood" when the bathtub overflows, which in turn triggers an event "AutoTurnOff a tap". 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. Tested are seven problem instances varying in the number of bathtubs and the number of (hot/cold) taps and in the constraints given in a goal.

The encoding in a CNF formula generated by the TM-LPSAT compiler is simplified by the Crawford's Compaction algorithm, and then the simplified CNF formula is fed into the constraint solvers: The compaction algorithm includes unit propagation, subsumption, and pure literal elimination.

Set-ups of SAT-based Arithmetic Constraint Solvers

The LPSAT is set-up with options of backtracking and learning.

Options on heuristics in MathSAT are used to select the best performance. "MathSAT03" and "MathSAT04" with Relsat heuristic utilize dependency feature (only actions and starting/ending actions of durative actions are independent.). *Static*

[69] Since the TM-LPSAT compiler is not optimized, it would not be meaningful at this point the performance of TM-LPSAT is compared with the performance of LPG: LPG solves these problems used in our testing less than a couple of seconds.

learning feature is turned on, which may make similar effects on learning on the run in "MathSAT04". Also, *early pruning* and *lpl* (LP in the last) option are turned on with "MathSAT04".

The temporal planning is tested with "MathSAT03" supporting mathematical back-jumping; metric planning and temporal metric planning requiring general forms of linear constraints are tested with "MathSAT04".

Platform of Testing

The times are measured on Linux 1.8 GHz Pentium IV processor with 512 MB RAM. For the algorithms with nondeterministic choice points, such as LPSAT and MathSAT with Relsat Heuristics, searching time averaged over 20 runs is presented along with deviation.

6.2.1 Metric Planning

Actions are defined as independent variables for "MathSAT04".

Table 1 shows the performances of the decision procedures with our metric encoding. The domains and problems from "Numeric Category" of IPC3 are tested. Table 2 shows the performances of MathSAT04 with different heuristics.

The constraints of encoding generated in metric planning are either mostly in the form V_i = *Constant*[70] (direct influence by discrete change), or in a general form of

[70] The direct influences by discrete change are constant in particular for the instances used for tests.

linear constraints. The V_i = *Constant* (and its complementary form V_i = 0 from the direct influence) is dealt by static learning and equality processing routine in "MathSAT03" so that it will not call a LP solver. On the other hand, LPSAT should get benefits from back-jumping based on conflicts of numeric constraints. The results do not show which feature is more effective in this kind of planning.

(Unit:Seconds)

Domain	DriverLog			Zeno		
Problem	#2	#3	#4	#2	# 3	#4
Plan Quality	981	911	707	6784	4506	19964
No. of Time Points	10	8	8	6	5	5
Before Compaction:						
No. of Clauses	14681	9791	18351	4614	8739	9534
No. of Boolean Variables	3201	2242	3018	731	1283	1403
No. of Linear Constraints	796	582	760	477	845	905
No. of Real Variables	415	305	394	227	398	428
After Compaction:						
No. of Clauses	8729	5808	10688	2701	5199	5806
No. of Boolean Variables	1755	1256	1713	671	1197	1302
No. of Linear Constraints	784	572	743	459	822	883
No. of Real Variables	415	305	394	220	392	422
Compaction Time	92.29	40.71	139.79	8.13	28.97	35.57
LPSAT						
Searching Time	149.82	29.28	296.7	1.15	16.6	3.41
(Deviation)	125.18	17.53	139.79	0.24	9.37	0.76
MathSAT04						
Searching Time	213.14	37.59	231	0.552	2.23	5.55

Table 1. Performances of Metric Planning with IPC3 Numeric Category

Heuristics	JW	2JW	Sato	Satz	Boehm	Moms	Relsat	Unitie
ZENO								
Problem #2	8.816	7.252	36.032	**0.552**	6.582	35.914	9.118	2.957
Problem #3	461.46	> 600	> 600	**2.23**	252.71	> 600	> 600	289
Problem #4	410.63	328.93	> 600	**5.55**	189.03	> 600	> 600	74.59
DriverLog								
Problem #2	> 600	> 600	> 600	> 600	> 600	> 600	**213.14**	> 600
Problem #3	> 600	> 600	> 600	164.81	> 600	> 600	102.65	**37.59**
Problem #4	> 600	> 600	> 600	**231**	> 600	> 600	> 600	> 600

Table 2. Heuristics in MathSAT04: Metric Planning with IPC3 Numeric Category

6.2.2 Temporal Planning

Starting actions and ending actions of durative actions are defined as independent variables for “MathSAT03”.

Table 3 shows the performances of the decision procedures with our metric encoding. Domains and problems are from “Simple Time” category of IPC03 domains. The additional table shows LPGP’s [LongFox03] performance for the same domains and problems. Table 4 shows the performances of “MathSAT03” with different heuristics.

(Unit:Seconds)

Domain	Satellite	Zeno			DriverLog
Problem	#1	#1	#2	# 3	#1
Plan Quality	41	173	599	280	91
No. of Durative Actions	9	2	8	6	8
No. of Time Points	8	4	7	6	7
Before Compaction:					
No. of Clauses	108310	2060	287532	321102	62934
No. of Boolean Variables	4873	603	2706	4371	4498
No. of Linear Constraints	1864	112	975	1676	1582
No. of Real Variables	428	51	283	555	429
After Compaction:					
No. of Clauses	13287	957	26121	45931	21743
No. of Boolean Variables	3602	253	1939	3368	2886
No. of Linear Constraints	1708	110	913	1592	1342
No. of Real Variables	428	51	283	555	449
Compaction Time	2109.93	1.67	7711.22	15026.3	574.59
LPSAT					
Searching Time	3.267	0.07	1.53	27.13	48.36
(Deviation)	0.92	0.01	0.51	15.43	62.8
MathSAT03					
Searching Time	0.15	0.00	0.08	0.16	0.3

Plan Quality	41	180	633	430	91
No. of Steps	9	1	6	9	8
LPGP					
Total Time	0.166	2.667	5.498	13.233	0.33

Table 3. Performances of Temporal Planning from IPC3 Simple Time Category

The total time consumed by LPGP [LongFox03] includes graph construction time and plan searching time.

The linear constraints in the temporal encoding are of either (i) $V_i - V_j$ {>,<.>=,<=} *Constant*, or (ii) V_i {>,<.>=,<=} *Constant*. These types of constraints are extensively explored in the reasoning of the timed systems [Audemard02-a] and good heuristics are known. "MathSAT03" is particularly effective in dealing with constraints of these forms, which it solves using a variant of the Bellman-Ford algorithm, rather than calling a LP solver as in LPSAT The results show that "MathSAT03" outperforms in this kind of planning.

In temporal planning with durative actions, it is worthy to compare TM-LPSAT with LPGP: The temporal encoding of TM-LPSAT is similar to Graphplan temporal modeling adopted in LPGP. Like TM-LPSAT, consistency on temporal constraints imposed by actions included in the plan is checked using a LP solver. The difference is that TM-LPSAT searches for a plan non-directionally, and the LPGP searches backward. Consequently, TM-LPSAT does not suffer from 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.

Heuristics	JW	2JW	Sato	Satz	Boehm	Moms	Relsat	Unitie
Satellite Problem #1	> 300	> 300	> 300	**0.15**	> 300	> 300	> 300	> 300
ZENO								
Problem #1	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.01
Problem #2	0.51	0.49	18.46	**0.08**	0.45	18.26	18.91	5.25
Problem #3	1.85	1.84	> 300	**0.16**	1.75	>300	>300	14.71
DriverLog Problem #1	> 300	0.36	> 300	1.34	**0.3**	> 300	> 300	> 300

Table 4. Heuristics in MathSAT03: Temporal Planning with IPC3 Simple Time Category

6.2.3 Temporal Planning with Continuous Changes

Only actions are defined as independent variables, but events or processes are not.

Table 5 shows performance of a variation of Bathtub domain. Table 6 shows the performances of "MathSAT04" with different heuristics.

(Unit:Seconds)

Problem Instances	Prob #0	Prob #1	Prob #2	Prob #3	Prob #4	Prob #5	Prob #6
Size (# of Bath x # of Taps)	1 x 2	1 x 4	1 x 8	1 x 16	2 x 4	2 x 8	2 x 16
Before Compaction:							
No. of Clauses	398	638	1118	2078	1093	1868	3422
No. of Boolean Variables	273	373	573	973	658	976	1616
No. of Linear Constraints	166	198	262	390	323	386	514
No. of Real Variables	68	84	116	180	129	161	225
After Compaction:							
No. of Clauses	376	568	952	1720	974	1586	2812
No. of Boolean Variables	242	316	464	760	543	758	1191
No. of Linear Constraints	166	198	262	390	323	386	514
No. of Real Variables	68	84	116	180	129	161	225
Compaction Time:	0.11	0.26	0.72	2.43	0.73	2.01	6.55
LPSAT							
Searching Time	0.167	0.318	3.753	1.104	2.079	18.296	87.997
(Deviation)	0.023	0.149	7.023	0.998	1.286	2.83	160.92
MathSAT04							
Searching Time	0.03	0.18	0.12	0.26	1.55	0.52	0.89

Table 5: Performances of Temporal Planning with Continuous Changes: Bathtub Domain

The number of steps was set to 5 for each instance; the plans for problem 4, 5 and 6 should be parallel as much as possible to achieve a goal within 5 steps.

The patterns of linear constraints are mostly in a general form of linear constraints of 3 variables or V_i = *Constant* (from direct influence by discrete change or continuous change). Temporal metric encoding should be more constrained such that metric constraints are intertwined with temporal constraints. Also generally the proportion of numeric constraints and variables is larger than in any other planning.

The results show at least with this domain that (i) “MathSAT04” still get benefits from layered structure and choice of good heuristics at branching points, compared to LPSAT, (ii) the choice of heuristic at branching point makes a big difference in performance with “MahSAT04”. However, those results should be tested with different domains and problems.

Heuristics	JW	2JW	Sato	Satz	Boehm	Moms	Relsat	Unitie
Prob #0	0.09	**0.03**	0.05	0.2	0.12	0.06	0.682	1.19
Prob #1	0.23	**0.18**	125.21	0.9	0.54	124.87	0.45	0.28
Prob #2	2.36	**0.12**	0.15	> 600	30.79	0.17	62.037	0.2
Prob #3	85.39	**0.26**	0.35	> 600	> 600	0.36	> 600	0.53
Prob #4	> 600	200.11	> 600	**1.55**	> 600	> 600	243.7	> 600
Prob #5	> 600	1.21	**0.52**	> 600	> 600	**0.52**	> 600	> 600
Prob #6	> 600	2.58	**0.9**	> 600	> 600	**0.89**	> 600	> 600

Table 6: Heuristics and “total” time in MathSAT04: Bathtub Domain

Chapter 7 Extensions of TM-LPSAT

In this Chapter, we suggest possible extensions of TM-LPSAT in continuing research on this thesis. In Section 7.1, we list the possible directions of extensions of TM-LPSAT. In Section 7.2, we discuss further techniques of optimizing encoding. In Section 7.3, we present the known techniques to optimize the LPSAT engine based on a systematic solver, particularly utilizing structure of planning domains.

7.1 Possible Extensions

Optimization of Encoding and LPSAT Engine

The optimization of encoding is the first and most necessary step in making TM-LPSAT scalable and practical to a certain degree. We discuss these optimization techniques further in Section 7.2.

Although it has been known that general SAT solvers work well with planning domains, there is room to optimize a SAT solver when combined with an arithmetic constraints solver. Also, the structure specific to planning domains can be utilized to optimize the combined decision procedure. These optimization techniques are discussed in Section 7.3.

Adaptation to our Encoding into Planning Framework based on Plan Graph

There are (at least) two approaches to project a model of time[71] into a plan graph. One[72] is to model the graph as a "uniform" flow of time so that each fact layer has fixed duration and each action layer corresponds to an absolute time. The other[73] is to model the graph to represent the "logical" structure of plans.

The model of time adopted in TM-LPSAT naturally corresponds to the second temporal projection: action layers are mapped to time points in our encoding and the durations of fact layers are resolved by temporal metric constraints imposed by actions happening at the end points of the fact layer. Thus, the adaptation of our temporal metric encoding into the plan graph does not appear so difficult.

In addition to being utilizable for pruning[74] SAT encoding[75], CSP encoding[76], or MILP encoding[77], the plan graph (constructed with metric temporal information and characteristics of temporal models) opens up various search methods for plans. Apart from the backward search[78] adopted in Graphplan, these can include non-directional searches by general SAT solvers: (i) *Simulated Systematic SAT Engine* in DPPlan [Baioletti02] [Baioletti03], which simulates on a plan graph the strength of SAT solvers (i.e. non-directional propagation and search) and the search techniques of DPLL (i.e. if stuck with a certain value, flip it for a correct plan). Moreover, in the selection of variables and their values, it can freely utilize heuristics specific to planning domains

[71] [Coddington02] and {longFox03} deal with extensive analysis of these two temporal projections on plan graph.
[72] This temporal projection was adopted in TGP [Smith99], TP4 [HaslumGeffner01] and Sapa [Do03-b].
[73] This temporal projection is adopted in LPGP [LongFox03].
[74] The current implementation of TM-LPSAT prunes search space in a Graphplan style; however, the encoding stays in Sate-Space encoding, rather than in Graphplan encoding.
[75] It's called "Graphplan encoding" [KautzMcAllester96] and adopted in BlackBox SAT-based planner [KautzSelman99].
[76] it was adopted in GP-CSP [DoKambhampati00].
[77] It was adopted in MILP planner [DimopoulosGerevini02].

as well as problem domains. (ii) *Simulated Stochastic SAT Solver* in LPG temporal metric planner [GereviniSerina03] [GereviniSerina03], which uses a local search on the plan graph along with domain independent heuristics. (iii) *Branch and Bound on Plan Graph* in BBG [HoffmannGeffner03], in which the search is done non-directionally along with heuristic functions to prune the search space.

Almost all the work done in temporal metric planning, including optimization, is based on the assumption that actions have constant durations and known numeric resource usages. But we are chiefly interested in dealing with richer expressiveness of temporal, metric constraints, such as uncertain durations, unknown resource usages, numeric parameters and, eventually, continuous changes. We believe that once our encoding is adopted on a plan graph, planning architectures searching on the plan graph in the manner of a systematic or stochastic SAT solver (i.e. simulating SAT engine) may be promising and robust in terms of (i) dealing with the richer temporal, metric constraints mentioned above, (ii) overcoming SAT-based architecture's inability to utilize plan metric, (iii) and integrating other heuristics specific to planning domains into the architectures. Those architectures can ultimately exploit all benefits from Graphplan, SAT-based framework, and heuristic-based framework.

Currently we are working on mapping our temporal, metric encoding into the plan graph.

LPSAT Engine Based on Stochastic SAT Solver

[78] It is known that backward search is not an effective means for metric planning.

All the known SAT-based decision engines are based on variants of the DPLL systematic solver. As a related work, ILP-Plan metric planner [KautzWalser00] uses integer local search based on WalkSat [Selman93] to solve mixed integer linear programming problems in planning domains; LPG temporal metric planner [GereviniSerina03] uses a local search similar to WalkSat on a plan graph, along with domain-independent heuristics, which shows the best performance in IPC3 planning competition.

Encoding Schemes

It has been proven empirically that different encoding schemes can make a big difference in performance [Giunchiglia98] [Kautz96]. Another natural extension of TM-LPSAT is the application of other encoding schemes. Considering that a partial order–based approach is known to be good at handling continuous time, (lifted) causal link encoding [KautzMcAllester96] is something worthy to explore. There are some works [Wolfman] [Do03-a] on causal link encoding for temporal planning in semi-continuous time, but at this point it is unclear to us whether it would be feasible to represent concurrent continuous and/or discrete changes. Also, as mentioned above, a plan graph extended with our encoding can be compiled into Graphplan encoding [KautzMcAllester96] or CSP [DoKambhampati00].

Representation and Reasoning with Qualitative/Incomplete Information

Reasoning with qualitative, incomplete properties and relationships among the continuously changing quantities (parameters) is extensively explored in Qualitative Reasoning community [Forbus84] [Weld90] [Kuipers01]. However, they have not yet been incorporated into planning frameworks as well as planning domain definition languages. In related work, Davis [Davis92] gives a logical analysis of Qualitative Process Theory [Forbus84], while Miller and Shanahan [Miller96-b] speculate on incorporating qualitative information about parameter behavior into their logical formalism based on event calculus dealing with both continuous and discrete changes.

7.2 Optimization of Encoding

Techniques to Reduce Encoding Size

It has been shown empirically that smaller encodings can generally be solved faster, although a smaller encoding is not always easier to solve [Kautz96] [Ernst97].

- Different representation techniques for the same axioms to reduce encoding size have been extensively explored [Ernst97]: factoring, operator splitting, bit representation, etc.
- Brafman [Brafman01-a] reports that a substantial percentage (more than 50% on average) of the clauses in SAT-encodings of planning problems (clearly from the action representation and frame axioms, for instance) are binary clauses. His simplification algorithm for binary clauses has proven very beneficial, especially with systematic solvers. We tested his 2-simplifier with

our encoding simplified by Crawford's COMPACT; it shows a more than 20% reduction in the number of clauses.

- Simplification techniques for arithmetic constraints include (i) constant folding and expression folding as used in compilers, (ii) recognition of arithmetic equivalence classes, (iii) and simplification algorithms for linear inequalities connected by Boolean operators (done in the hardware/circuit verification community [Strichman02] [Amon00]).
- The most popular technique for pruning search space is using a plan graph built in Graphplan-style, as applied in Blackbox system (Graphplan + SAT solver) [KautzSelman99], DPPLAN (Graphplan + simulated SAT engine) [Baioletti02], LPGP temporal planner (Graphplan + LP solver) [LongFox03], MILP temporal planner (Graphplan + ILP solver) [DimopoulosGerevini02], LPG temporal metric planner (Graphplan + local search & repair) [Gerevini03], BBG (Graphplan + branch and bound) [HoffmannGeffner03], Metric-FF [Hoffmann03] or Sapa metric temporal planner [Kambhampati03] [Do03-b] (to extract heuristics).

  Many heuristics for a Graphplan-based framework have been explored to prune search space. In particular, r*eachability* (forward mutex propagation), *relevance,* and *inseparability* (backward mutex propagation) can be utilized on a plan graph: [Do00] and [Brafman01-b] present reachability and relevance-based pruning techniques in a plan graph based SAT encoding.

  The current implementation of TM-LPSAT prunes search space in a Graphplan style. As shown in Chapter 6, the degree of pruned actions and fluents varies drastically with domains and problems. Also the possible goal

states can be collected while pruning. This is useful for the durative temporal models (Level 3 & 4) in which the goal state may be in any time point because the semantics impose all durative actions started in the plan to be finished in the plan.

- Another technique to reduce encoding size would be to utilize domain structure inferred from a domain analysis tool, such as TIM [FoxLong99] [FoxLong00] or DISCOPLAN [Gerevini00]. This can be applied either on the plan graph (to prune) or on the encoding level (to encode domain knowledge). The MILP temporal planner [DimopoulosGerevini02] uses single-valuedness (On(X,*Y)) and binary XOR constraints (ON(X,Y) and CLEAR(Y)) extracted by a domain analysis tool to infer pairs of actions that cannot be concurrent. This can be used to reduce the number of temporal constraints as well as the number of temporal variables. Notice that this kind of mutex relation cannot be identified by mutex rules imposed by the semantics.

Linearization Techniques

It may be possible to extend the power of TM-LPSAT by including techniques for making originally nonlinear constraints linear.

- The information given in the initial and goal conditions could be used to reduce an apparently nonlinear constraint into linear one by using simplification techniques such as constant folding.

- A linearization technique such as *clock translation* [Henzinger97] [Henzinger98] used in hybrid systems can be applied to reduce nonlinearity into linearity.

7.3 Optimization of LPSAT Engine

A systematic SAT solver is assumed for this section, as general SAT-based decision procedures are based on DPLL. Optimizing a stochastic SAT solver would be a different line of work.

Optimization of a SAT Solver Integrated with Arithmetic Constraints Solver

In the SAT-based framework integrated with an arithmetic constraint solver, the main bottleneck in performance is the time consumed by arithmetic constraint solver [Alessandro01] [Wolfman00]. This will probably become more true as the performance of SAT solvers [Zhang02] improve. In order to reduce the time consumed by an arithmetic constraint solver, the two solvers should interact. The general techniques used in systematic solvers are as follows: (i) the SAT solver utilizes information from the arithmetic solver on the inconsistent constraints set so that it can prune the search tree (called back-jumping and learning from a minimal subset of inconsistent constraints) [Wolfman00] [Castellini01]; (ii) the inconsistent arithmetic constraint sets are identified in a preprocessing step, and then clauses are added to prevent any combinations of those constraints from being considered active in a truth assignment. These techniques are applied to many applications [Audemard02-b] [Audemard02-a]

[Wolfman00] and have proved to be effective. However, this technique is computationally expensive, so the justification for its use comes from greater efficiency gain. Or, at the encoding stage, inconsistent pairs of arithmetic constraints can be prevented from being triggered simultaneously as done in our encoding of direct influence[79].

Optimization of a General SAT Solver for Planning Domains

It is possible to exploit the structure specific to the planning domains, such as the variable dependency that is lost by converting to CNF. The variable dependency can be used to select variables at the branching points in order to prune the search space (in particular, to reduce calls to arithmetic solvers).

In [Giunchiglia98], observing that the values of fluents at a certain time point derive deterministically from the initial states and the sequence of actions performed until that point, their planner utilized the nondeterminism in the planning domain, that is, the choice of actions to be performed at each time point. It showed a dramatic reduction in the search space. Similar ideas are utilized in Rintanen's planner [Rintanen98] [Rintanen99], DPPlan [Baioletti02] [Baioletti03] and LPG [GereviniSerina03] [Gerevini03].

Mali [Mali02-b] extensively experimented on the effects of directional search on SAT encodings of planning domains using Satz systematic solver [Li97], where values are assigned to action and/or fluent variables in forward/backward directions within/without intermittent manner. It is reported that directionality does matter in

[79] See axioms (3) and (4) and the attached footnote in Section 4.2.2.

solving SAT encodings of planning domains: forward and bidirectional searches perform better than the backward search.

Worthy of investigation is how both of these observations work with metric/ temporal domains and the identification of the structure specific to temporal and/or metric domains.

Chapter 8 Conclusion

The dimension of time is inherently involved in any domain with change. Many complex real-world domains involve continuous, metric time, resources, metric quantities, and concurrent actions. However, until very recent years, continuous time had not been studied by the planning community, mainly because planning techniques have not caught up to cope with continuous time and subsequent complications with resources and concurrency.

We have developed a SAT-based temporal planner that can reason about continuous and/or discrete changes. As far as we know, our TM-LPSAT is the first SAT-based planner that can reason in continuous time.

It was claimed that the SAT-based approach to continuous time would not be feasible [Smith00-b] [LongFox00]: Allowing a metric quantity to be a continuous function of time raises the possibility that there could be an infinite number of possible actions, such as a refueling action in which the level of fuel added is a function of the duration of the action. The way this problem is resolved in TM-LPSAT is to encode in terms of the time points at which "interesting activities" happen, rather than every possible action point as encoding in discrete time. It can be interpreted in such a way that time points in TM-LPSAT are an abstraction of all possible refueling actions: If refueling action at a certain metric time is "interesting" from the point of goal achievement, the action at the metric time is bound to one of the time points. A time point in TM-LPSAT is bound to the metric time value over **R*** (nonnegative real numbers). Number of time points is determined by the interesting activities happening

at distinct times, which are needed to achieve the goal. It is possible that more than one activity happens at a time point, i.e. it supports parallelism.

The TM-LPSAT generates plans based on the following assumptions:

- The world is closed.
- Actions are deterministic.
- There is a single agent in the world.
- Time is isomorphic to nonnegative real numbers.

The TM-LPSAT has the following features:

- It accepts planning problems described in PDDL+ (but does not include the plan metric feature of PDDL+).
- It generates a parallel plan that contains concurrent, asynchronous actions.
- It can reason about concurrent continuous and/or discrete changes on numeric-valued fluents.
- Actions may depend on or make changes on piecewise linear metric constraints.
- It can reason about durative actions that occur over extended intervals of time; the intervals may be static (constant), dynamic (variable) or uncertain (inequality relation).
- It can reason about continuous changes captured within durative actions.
- It can reason about autonomous processes carrying continuous changes and external events.
- It supports a numeric-valued or interval-valued parameter to action.

- It supports interval type as a fluent type. In addition to Allen's 13 Interval relations, extended relational operations among interval-valued fluents and operations of updating interval-valued fluents are supported.
- It supports sharable, reusable resources: multiple-capacity resources and interval-partitioned resources. The resource usages can be variable.
- It can deal with the ADL[80] subset of PDDL+, including such features as typing, negative preconditions, disjunctive preconditions, equality, quantified preconditions, and conditional effects.

We have developed encoding schemes to support these features. Based on the LPSAT engine [Wolfman99], the TM-LPSAT temporal metric planner has been implemented. Also we have experimented on our metric temporal encodings with the MathSAT decision procedure solving propositional combinations over Boolean variables and linear arithmetic constraints. The test results show that the solving times of TM-LPSAT can be comparable to state-of-the-art planners based on other planning frameworks. The issues are to find and utilize the structure specific to the planning domains or problems to guide the search in decision procedures.

The TM-LPSAT has the following limitations:

- It restricts arithmetic constraints and continuous changes to be piecewise linear. The encoding for triggering and terminating events/processes *with no time slip* (also constraining as the ending point of a durative action when invariants become ***FALSE*** from ***TRUE***) is based on the assumptions that all

[80] In the SAT-based framework like TM-LPSAT, these advanced features can be handled without any extra complication.

arithmetic constraints in the conditions are piecewise linear and that it is restricted to non-strict equalities.

- A crucial limitation of the SAT-based approach to the temporal metric planning is its inability to use optimization functions. In general, plan quality in temporal metric domains is very likely to be inherently multi-dimensional, which may be composed of a temporal quality (such as makespan) and a plan cost (such as resource consumption or cumulative action cost). Moreover, there may be interdependencies between different quality metrics. In the planning architecture of a current state-of-the-art SAT solver integrated with an arithmetic solver, it is not feasible to find an optimal plan according to plan metrics [Wolfman99] [Smith00], even within bounded length; there may be other truth assignments that satisfy the formula and optimize the objective functions better.
- Another fundamental limitation of TM-LPSAT, as well as the SAT-based approach itself, is the size of the encoding, since it needs Boolean variables for fluents and ground actions to be defined at every time point and the size blows up when transformed into a CNF formula. Although there is plenty of room to optimize the encoding size as elaborated in Section 7.2, the approach ultimately would end up with memory explosion.

As discussed in Section 3.1, there are some planning systems that can reason about continuous changes: the extent of concurrency and expressiveness of temporal metric constraints that these systems can handle are very limited and the systems scale up poorly. The current state-of-the-art domain independent temporal planners

dealing with durative actions are at an early stage of development in terms of the expressiveness they can handle (e.g., constant durations and STRIPS-style operators). There are a few temporal and metric planners, but they can deal mostly with discrete changes within a durative action; LPG [Gerevini03] and Sapa [Do03-b] temporal and metric planners claim to handle continuous changes within durative actions, but the extent of concurrency they can deal with and their scalability are unclear.

McDermott's Optop planner, which deals with durative actions and autonomous processes [McDermott03-b], is at the stage of feasibility testing for the Estimated Regression approach, as is our planner. However, the Optop can optimize plans and is open to deal with nonlinearity of arithmetic constraints.

There exists neither a SAT-based temporal planner in continuous time nor a SAT-based temporal and metric planner.

We claim that our TM-LPSAT shows that the SAT-based framework is feasible for planning in continuous time. Based on experiments with domains and problems used in IPC3, we can see that in terms of the searching time the SAT-based approach to metric temporal planning is quite comparable to other approaches and there remains plenty of room to push the limits of the SAT-based approach.

Appendix A: Notations

A.1 Sets

T	A set of time points
PF	A set of propositional fluents
NF	A set of numeric-valued fluents
PR	A set of processes
E	A set of events
A	A set of instantaneous actions
DA	A set of durative actions
PosCond(pf)	The set of actions and events with positive literal ***pf*** in precondition
NegCond(pf)	The set of actions and events with negative literal $\neg$ ***pf*** in precondition
Ref(nf)	The set of actions and events which refer numeric-valued fluent ***nf*** in precondition or effect.
PosEffect(pf)	The set of actions and events with positive literal ***pf*** in effect
NegEffect(pf)	The set of actions and events with negative literal $\neg$ ***pf*** in effect
SetEffect(nf)	The set of actions and events with *assign* statement on numeric-valued fluent ***nf***
AddEffect(nf)	The set of actions and events with *increase* or *decrease* statement on numeric-valued fluent ***nf***

A.2 Predicates

Active(a,T_i) (***DA*** $\cup$ ***A*** $\cup$ ***E*** $\cup$ ***PR***) * ***T*** $\rightarrow$ { True, False }

- Is action (process or event) active at time point ***T_i***?

Proposition(pf,T_i) ***PF*** * ***T*** $\rightarrow$ { True, False }

- Is a propositional proposition ***pf*** true at at time ***T_i***?

Continues(a,T_i) (***DA*** $\cup$ ***PR***) * ***T*** $\rightarrow$ { True, False }

- Is there any durative action (process) ***a*** continuing at time ***T_i***? (remind that it is possible for the same durative actions started at different time points to be concurrent.)

A.3 Functions

RateOfChange(a,nf,T_i) (***DA*** $\cup$ ***PR***) * ***NF*** * ***T*** $\rightarrow$ ***R***

- Rate of change of a numeric-valued fluent ***nf*** by a durative action (process) ***a*** at time ***T_i***

NetContiChange(a,nf,T_{i-1},T_i) (***DA*** $\cup$ ***PR***) * ***NF*** * ***T*** * ***T*** $\rightarrow$ ***R***

- Continuous change in a numeric-valued fluent ***nf*** made by a durative action (process) ***a*** over the interval (***T_{i-1}***,***T_i***).
- Since the same durative actions but started at different time points can be concurrent over the interval (***T_{i-1}***,***T_i***), ***a*** distinguishes those instantiations of the durative action.

DiscreteChange(a,nf,T_i) (***DA*** $\cup$ ***A*** $\cup$ ***E***) * ***NF*** * ***T*** $\rightarrow$ ***R***

- Discrete change on a numeric-valued fluent ***nf*** by an action (event) ***a*** at time ***T_i***

- ***a*** distinguishes different instantiations of the same durative action started at different time points.

Appendix B: Time-Labeling Convention

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

<LogicalExpression>[***Ti-***] (*read as the logical expression is time-labeled by* ***Ti-***)

- The logical expression is time-labeled with the values of fluents before the discrete changes made at the time point T_i.

  e.g. Precondition of an action or event happening at T_i
- Each propositional fluent is time-labeled by T_{i-1}.
- Each numeric-valued fluent is time-labeled by $Value_{before}(nf, T_i)$.

<LogicalExpression>[***Ti+***] (*read as the logical expression is time-labeled by* ***Ti+***)

- The logical expression is time-labeled with the values of fluents right after the discrete changes made at the time point T_i.

  e.g. Postcondition of an action or event happening at T_i

  Precondition of a process triggered by discrete changes at T_i
- Each propositional fluent is time-labeled by T_i
- Each numeric-valued fluent in the right-hand side of an arithmetic statement is time-labeled by $Value_{before}(nf, T_i)$.
- Each numeric-valued fluent in the left-hand side of an assignment statement or precondition (of a process) is time-labeled by $Value_{after}(nf, T_i)$.

Appendix C: Encoding Temporal Model of Durative Actions

First, we define a simplified bathtub domain and a problem in extended PDDL+ level 4, and then, show its encoding in the meta-level notation.

C.1 Bathtub Domain

This is a simplified variation of Bathtub domain introduced in Section 1.1. There are more than one bathtub in the bathroom; there are more than one tap, each of which flows at its own flow rate when it is turned on. We may plan to fill the bathtub to a certain level. Also, we may want to add some bubble while running water. Observe that this domain shows that (i) concurrent continuous changes on the level of the bathtub may occur by turning on multiple taps, and (ii) discrete change on the water level of bathtub can occur while continuous changes go on the level of bathtub. Here we model this domain in extended PDDL+ Level 4 (using ADL features) and show a problem.

Problem definition:

```
(:problem              A-Problem-Simplified-Bathtub
 :objects              (HOT – tap  COLD – tap  MYBATH – bath)
 :init                 (plug_in  MYBATH)
                       (level MYBATH 0)
 :goal          (and   (bubble_added  MYBATH)
                       (>  (level MYBATH)   (/ (capacity MYBATH) 2 ))
                       (<= (level MYBATH)  (* 0.75 (capacity MYBATH))
                )
)
```

Domain definition:

```
(:domain                Simplified-Bathtub
 :requirements          (:types  :durative-actions  :adl)
 :types                 (bath  tab)
 :predicates            (  (tap_on  ?b – bath  ?t – tap)
                           (plug_in  ?b – bath)
                           (bubble_added ?b – bath)
                        )
:fluents                (  (level  ?b – bath)  - fluent
                           (hot-flows  ?b – bath) - fluent
                           (cold-flows  ?b – bath) – fluent
                           (flow  ?b – bath  ?t – tap)  - float
                           (capacity ?b – bath) – float
                        )

(:durative-action       fillBath
 :parameters            (?b – bath  ?t - tab)
 :duration              ( )
 :condition      (and   (at start  (plug_in  ?b))  (at start  (not (tap_on  ?b ?t)))
                        (over all   (<=  (level ?b)  (capacity ?b)))
                        (at end    (tap_on  ?b ?t))   (at end    (plug_in ?b)))
  :effect        (and   (at start  (tap_on ?b ?t))
                        (at end  (not (tap_on ?b ?t)))
                 (increase  (level ?b)  (* #t  (flow ?b ?t) )))
)

(:action                addBubble
 :parameters            (?b – bath)
 :precondition  (and    (not (bubble_added ?b))
                        (exists (?t – tab)  (tap_on ?b ?t))
                        (<=  (level ?b)  (/ (capacity ?b) 2)))
 :effect        (and    (bubble_added ?b)
                        (increase  (level ?b)  (/  (capacity ?b) 30))

 )

(:action                turnOn
 :parameters            (?b –bath   ?t –tab)
 :precondition          (not (tap_on ?b ?t))
 :effect                (tap_on ?b ?t)
 )

(:action                turnOff
 :parameters            (?b –bath   ?t –tab)
 :precondition          (tap_on ?b ?t)
 :effect                (not (tap_on ?b ?t))
 )
```

C.2 Encoding in Meta-Level

Notations (with examples):

- A ground durative action parameterized with starting time point and ending time point:

  fillBath(BATH0,HOT,T_i,T_j)

  Or, a ground action parameterized by starting time point

  fillBath(BATH0,HOT,T_i)
- A starting/ending action of a ground durative action parameterized by the starting time point T_i:

  As(fillBath(BATH0,HOT,T_i)), Ae(fillBath(BATH0,HOT,T_i))
- A ground instantaneous action is active at a given time point or not:

  Active(Ae(fillBath(BATH0,HOT,T_i)),T_j)
- Each ground propositional fluent is parameterized by the time point, meaning that it is true or false at the time point:

  tap_on(BATH0,HOT,T_i)
- Each numeric-valued fluent is represented as follows:

  $Value_{before}$(level(BATH0),T_j)), $Value_{after}$(level(BATH0),T_j)),
- Each numeric-valued float is represented as follows:

  Value(flow(BATH0,HOT))
- The duration of a ground durative action is represented as follows:

  Value(duration(fillBath(BATH0,HOT,T_i)))
- At least one instance of the ground durative action is active at the time point:

  Continues(fillBath(BATH0,HOT), T_k)
- A direct influence (continuous change) on a numeric-valued fluent by a ground durative action started at T_i and ended in T_j over the interval [T_{k-1},T_k]:

  NetContiChange(fillBath(BATH0,HOT,T_i,T_j), level(BATH0),T_{k-1},T_k)
- A discrete change on a numeric-valued fluent by a instantaneous action

  DiscreteChange(addBubble(BATH0,T_k), level(BATH0),T_k)

Representation of Durative Actions:

(1) $Active(As(fillBath(BATH0,HOT,T_i)),T_i) \Rightarrow$

$plug\_in(BATH0,T_{i-1}) \wedge \neg tap\_on(BATH0,HOT,T_{i-1}) \wedge tap\_on(BATH0,HOT,T_i)$

(2) $Active(Ae(fillBath(BATH0,HOT,T_i)),T_j) \Rightarrow$

$tap\_on(BATH0,HOT,T_{j-1}) \wedge plug\_in(BATH0,HOT,T_{j-1}) \wedge \neg tap\_on(BATH0,HOT,T_j)$

(3) $Active(Ae(fillBath(BATH0,HOT,T_i)),T_j) \Rightarrow Active(As(fillBath(BATH0,HOT,T_i)),T_i)$

$\wedge [T_i + Value(duration(fillBath(BATH0,HOT,T_i))) = T_j]$

(4) $Active(As(fillBath(BATH0,HOT,T_i)),T_i) \Rightarrow$

$[Active(Ae(fillBath(BATH0,HOT,T_i)),T_{i+1}) \vee \ldots \vee Active(Ae(fillBath(BATH0,HOT,T_i)),T_n)]$

(5) $Active(Ae(fillBath(BATH0,HOT,T_i)), T_j) \Rightarrow$

$[\forall T_k \; T_i < T_k < T_j \; Continues(fillBath(BATH0,HOT), T_k)]$

(6) $Continues(fillBath(BATH0,HOT), T_k) \Rightarrow$

$[Value_{before}(level(BATH0),T_k) <= Value(capacity(BATH0))] \wedge$

$[Value_{after}(level(BATH0),T_k) <= Value(capacity(BATH0))]$

(7) $\neg Continues(fillBath(BATH0,HOT), T_{k-1}) \wedge Continues(fillBath(BATH0,HOT), T_k) \Rightarrow$

$[Value_{after}(level(BATH0),T_{k-1}) <= Value(capacity(BATH0))] \wedge$

(8) $\forall T_k, \; T_i < T_k <= T_j ,$

$Active(As(fillBath(BATH0,HOT,T_i)),T_i) \Rightarrow$

$[NetContiChange(fillBath(BATH0,HOT,T_i),level(BATH0),T_{k-1},T_k) =$

$(T_k - T_{k-1}) * Value(flow(BATH0,HOT))]$

$\neg Active(As(fillBath(BATH0,HOT,T_i)),T_i) \Rightarrow$

$[NetContiChange(fillBath(BATH0,HOT,T_i),level(BATH0),T_{k-1},T_k) = 0]$

Representation of Constraints on Duration

The "fillBath(BATH0,HOT)" action in the Bathtub domain defined in Example2-1, starts at T_i and ends at T_j has follwing constraint on its duration:

(:duration (at end (<= ?duration (/ (- (capacity ?b) (level ?b)) (flow ?b ?t))))

is encoded as

$$\text{Active}(\text{Ae}(\text{fillBath}(\text{BATH0},\text{HOT},T_i)), T_j) \Rightarrow$$
$$[\ \text{Value}(\text{duration}(\text{fillBath}(\text{BATH0},\text{HOT},T_i))) <=$$
$$(\text{Value}(\text{BATH0}) - \text{Value}_{\text{before}}(\text{level}(\text{BATH0}),T_j)) / \text{Value}(\text{flow}(\text{BATH0},\text{HOT}))\ ]$$

Note that in this case the constraint is time-annotated by "*at end*", which means that the constraint should be held at the ending time point of the action. Likewise, if it is time-annotated by "*at start*", the activation of its starting action implies the constraint at the starting point.

Frame Axioms:

Propositional Fluents:

$$\neg\, \text{tap\_on}(\text{BATH0},\text{HOT},T_{i-1}) \ \wedge\ \text{tap\_on}(\text{BATH0},\text{HOT},T_i) \Rightarrow$$
$$\text{Active}(\text{As}(\text{fillBath}(\text{BATH0},\text{HOT},T_i)),T_i)$$

Numeric-Valued Fluents: "level" of the Bathtub at T_3,

$$\text{Value}_{\text{before}}(\text{level}(\text{BATH0}),T_3) = \text{Value}_{\text{after}}(\text{level}(\text{BATH0}),T_2) +$$
$$\text{NetContiChange}(\text{fillBath}\ (\text{BATH0},\text{HOT},T_1,T_3), \text{level}(\text{BATH0}),T_2,T_3) +$$
$$\text{NetContiChange}(\text{fillBath}(\text{BATH0},\text{HOT},T_2,T_3), \text{level}(\text{BATH0}),T_2,T_3) +$$
$$\text{NetContiChange}(\text{fillBath}(\text{BATH0},\text{COLD},T_1,T_3),\text{level}(\text{BATH0}),T_2,T_3)+$$
$$\text{NetContiChange}(\text{fillBath}(\text{BATH0},\text{COLD},T_2,T_3), \text{level}(\text{BATH0}),T_2,T_3)$$

$$\text{Value}_{\text{after}}(\text{level}(\text{BATH0}),T_3) = \text{Value}_{\text{before}}(\text{level}(\text{BATH0}),T_3) +$$
$$\text{DiscreteChange}(\text{addBubble}(\text{BATH0},T_3), \text{level}(\text{BATH0}),T_3)$$

Mutual Exclusivity:

Propositional Fluents:

for “tap_on(BATH0,HOT)”,

$[\ turnOn(BATH0,HOT,T_i) \ \vee \ tapOff(BATH0,HOT,T_i)\ ] \ \wedge$

$[\ \neg\, turnOn(BATH0,HOT,T_i) \ \vee \ \neg\, tapOff(BATH0,HOT,T_i)\ ]$

Numeric-Valued Fluents:

for the “fillBath” ,a discretized durative action defined in Example 2-1 and “addBubble” ,

$[\ \neg\, Active(Ae(fillBath(BATH0,HOT,T_i)),T_j) \ \vee \neg\, Active(addBubble(BATH0),T_j)\ ]$

By rule 3 (no moving target rule for numeric-valued fluents), “fillBath” can not finish at time T_j at which “addBubble” occurs, since the ending action of “fillBath” refers (updates) the value of the level and “addBubble” updates (refers) the value.

C.3 Conditional Effect

This is “Match” domain [FoxLong03], burning a match to make a location light before picking up an object in the location.

```
(:durative-action burnMatch
 :parameters     (?m – match  ?l – location)
 :duration       (and     (< ?duration 5)  (> ?duration 0))
 :condition      (and     (at start (have ?m))  (at start (at ?l)))
 :effect         (and     (when  (at start (dark ?l))
                                 (and  (at start (not (dark ?l)))  (at start (light ?l))))
                          (at start (not (have ?m)))
                          (at start (burning ?m))
                          (at end (not (burning ?m)))
                          (when  (at start (dark ?l))
                                 (and  (at end (not (light ?l)))  (at end (dark ?l))))
)
(:action pickUp
 :parameters     (?l – location ?o – object)
 :precondition   (and     (at ?l)  (onFloor ?o ?l)  (light ?l))
 :effect         (and     (not (onFloor ?o ?l))  (have ?o))
)
```

Encoding:

(when (at start (dark ?l)) (and (at end (not (light ?l))) (at end (dark ?l)))) in “burnMatch” action over an interval $[T_i,T_j]$ is encoded as

$$Active(Ae(burnMatch(MATCH0,BASEMENT0,T_i)),T_j) \Rightarrow$$

$$[\ dark(BASEMENT0,T_{i-1}) \Rightarrow \neg light(BASEMENT0,T_j) \wedge dark(BASEMENT0,T_j)\ ]$$

Appendix D: Encoding Real-Time Temporal Model

D.1 Bathtub Domain

This is the PDDL+ Level 5 version corresponding to the Bathtub domain as defined in Appendix C.

Domain Definition:

```
(:action          tapOn
 :parameters      (?b – bath  ?t – tab)
 :precondition    (and      (plug_in ?b)  (not (tap_on ?b ?t)))
 :effect                    (tap_on ?b ?t)
)

(:process         fillBath
 :parameters      (?b – bath  ?t  - tab)
 :precondition    (and      (tap_on ?b ?t)  (<= (level ?b)  (capacity ?b))
 :effect                    (increase  (level ?b)  (*  #t  (flow  ?b ?t)))
)

(: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)
 :precondition    (and      (not  (bubble_added ?b))
                            (exists (?t – tab)  (tap_on ?b ?t))
                            (<=  (level ?b)  (/ (capacity ?b) 2))
  :effect         (and      (bubble_added ?b)
                            (increase  (level ?b)  (/  (capacity ?b) 30))

)
```

Assume that BATH0 is an object of bath type; HOT and COLD are objects of tap type.

D.2 Encoding in Meta-Level

(The same notational conventions are used as defined in Appendix 3.1.)

(1) Axioms for “flood” event,

$$\begin{aligned}
&\text{Active}(\text{flood}(\text{BATH0}),T_i) \Leftrightarrow \\
&\quad [\ \text{tap_on}(\text{BATH0},\text{HOT},T_{i-1}) \vee \text{tap_on}(\text{BATH0},\text{COLD},T_{i-1})\ ] \wedge \\
&\quad [\ \text{Value}_{\text{before}}(\text{level}(\text{BATH0}),T_i) >= \text{Value}(\text{capacity}(\text{BATH0}))\ ] \wedge \\
&\quad \text{dry_floor}(\text{BATH0},T_{i-1})
\end{aligned}$$

$$\text{Active}(\text{flood}(\text{BATH0}),T_i) \Rightarrow \neg\, \text{dry\_floor}(\text{BATH0},T_i)$$

(2) Axioms for “addBubble” action,

$$\begin{aligned}
&\text{Active}(\text{addBubble}(\text{BATH0}),T_i) \Rightarrow \\
&\quad [\ \neg\, \text{bubble_added}(\text{BATH0},T_{i-1})\ ] \wedge \\
&\quad [\ \text{tap_on}(\text{BATH0},\text{HOT},T_{i-1}) \vee \text{tap_on}(\text{BATH0},\text{COLD},T_{i-1})\ ] \wedge \\
&\quad [\ \text{Value}_{\text{before}}(\text{level}(\text{BATH0}),T_i) <= \text{Value}(\text{capacity}(\text{BATH0})) / 2\ ] \wedge \\
&\quad [\ \text{bubble_added}(\text{BATH0},T_i)\ ]
\end{aligned}$$

$$\begin{aligned}
&\text{Active}(\text{addBubble}(\text{BATH0}),T_i) \Rightarrow \\
&\quad [\ \text{DiscreteChange}(\text{addBubble}(\text{BATH0},T_i),\text{level}(\text{BATH0}),T_i) = \\
&\qquad\qquad \text{Value}(\text{capacity}(\text{BATH0})) / 30\ ]
\end{aligned}$$

$$\begin{aligned}
&\neg\, \text{Active}(\text{addBubble}(\text{BATH0}),T_i) \Rightarrow \\
&\quad [\ \text{DiscreteChange}(\text{addBubble}(\text{BATH0},T_i),\text{level}(\text{BATH0}),T_i) = 0\ ]
\end{aligned}$$

(3) Axioms for “fillBath” process,

$$\begin{aligned}
&\text{Active}(\text{fillBath}(\text{BATH0},\text{HOT}),T_i) \Leftrightarrow \\
&\quad [\ \text{tap_on}(\text{BATH0},\text{HOT},T_{i-1}) \wedge \\
&\qquad [\ \text{Value}_{\text{after}}(\text{level}(\text{BATH0}),\ T_i) <= \text{Value}(\text{capacity}(\text{BATH0}))\ ]\ ] \\
&\quad \wedge \\
&\quad [\ \text{tap_on}(\text{BATH0},\text{HOT},T_i) \wedge \\
&\qquad [\ \text{Value}_{\text{before}}(\text{level}(\text{BATH0}),\ T_i) <= \text{Value}(\text{capacity}(\text{BATH0}))\ ]\ ]
\end{aligned}$$

$$Active(fillBath(BATH0,HOT),T_i) \Rightarrow$$
$$[\ NetContiChange(fillBath(BATH0,HOT,T_i),level(BATH0),T_{i,}T_{i+1}) =$$
$$(T_{i+1} - T_i) * Value(flow(BATH0,HOT))\ ]$$

$$\neg\ Active(fillBath(BATH0,HOT),T_i) \Rightarrow$$
$$[\ NetContiChange(fillBath(BATH0,HOT,T_i),level(BATH0),T_{i,}T_{i+1}) = 0\ ]$$

(4) The axiom for Triggering “flood” event with no time slip:

$$[\ tap\_on(BATH0,HOT,T_{i-1}) \wedge dry\_floor(BATH0,T_{i-1}) \wedge$$
$$Value_{after}(level(BATH0),T_{i-1}) > Value(capacity(BATH0))\ ]$$
$$\Rightarrow [\ Value_{before}(level(BATH0),T_i) >= Value(capacity(BATH0))\ ]$$

$$[\ tap\_on(BATH0,COLD,T_{i-1}) \wedge dry\_floor(BATH0,T_{i-1}) \wedge$$
$$[\ Value_{after}(level(BATH0),T_{i-1}) > Value(capacity(BATH0))\ ]$$
$$\Rightarrow [\ Value_{before}(level(BATH0),T_i) >= Value(capacity(BATH0))\ ]$$

(5) The axiom for terminating “fillBath” process with no time slip:

$$[\ Value_{after}(level(BATH0), T_i) < Value(capacity(BATH0))\ ] \Rightarrow$$
$$[\ Value_{before}(level(BATH0), T_{i+1}) <= Value(capacity(BATH0))\ ]$$

Appendix E: Encoding Interval-Valued Fluents

We define operations on interval-valued fluents and present how the operations can be encoded in LCNF forms.

Operations with Intervals [Davis90] [Allen83]

To reason with intervals, we extend PDDL+ with the following operations among intervals and points:

<IntervalUpdateStmt> = **(** *set-I* <FunctionOfIntervalType> <IntervalExpr> **)**

- Assign evaluated <IntervalExpr> to <FunctionOfIntervalType>

<IntervalExpr> = <Interval> |
(<CompositonOp> <IntervalExpr> <IntervalExpr> **)**

<Interval> = <FunctionOfIntervalType> | <ConstantInterval>

<CompositonOp> = *overlap_of* | *join*

(*overlap_of I J*) $\equiv$ { [K_{start}, K_{end}] = *K* | $\forall x \in I \cap J \quad x \in K$ }

if I is overlapped with J, or contained in J, or vice versa
undefined otherwise

(*join I J*) $\equiv$ { [K_{start}, K_{end}] = *K* | $\forall x \in I \cup J \quad x \in K$ }

if I is contained in J or overlaps or meets with J, or vice versa
undefined otherwise

<IIRelationExpr> = **(** <RelationOp> <IntervalExpr> <IntervalExpr> **)**

<IIRelationOp> = *before* | *meets* | *overlaps* | *starts* | *during* | *finishes* | *equal* | *after* | *meet_by* | *overlapped_by* | *started_by* | contained_by | *finished_by* | *contained*

- <IIRelationOp> is Allen's 13 possible order relations between a pair of intervals, except "*contained*"
- *"contained"* is *"equal"*, *"starts"*, *"during"*, or *"finishes"*

<PIRelationExpr> = **(** <PIRelationOp> <Number> <IntervalExpr> **)**

<PIRelationOp> = *before-pi* | *after-pi* | *contained-pi*

<Number> = <ConstNumber> | <FunctionOfNumericlType>

Encoding of Interval Relations and Operations

First of all, we need to define the identity of interval, Empty Interval (EI), which will be used to represent undefined interval value in linear constraints:

EI $\equiv$ [$\boldsymbol{I_{start}}(EI) = \boldsymbol{I_{end}}(EI) = \text{Max}$],

where, Max is the maximum number representable

All undefined variables generated during evaluation of interval expression or operation is equal to this empty interval.

The translation of interval relations defined above into linear constraints in terms of end points of intervals (and points) is quite straightforward [Davis90], with additional care that I or J can be empty interval:

(*before* I J) is encoded as

$[[I_{end} < J_{start}] \wedge \neg [I_{end} = \text{Max}] \wedge \neg [J_{start} = \text{Max}]]$

(*meets* I J) is encoded as

$[[I_{end} = J_{start}] \wedge \neg [I_{end} = \text{Max}] \wedge \neg [J_{start} = \text{Max}]]$

(*overlaps* I J) is encoded as

$[[I_{start} < J_{start}] \wedge [J_{start} < I_{end}] \wedge [I_{end} < J_{end}]]$

(*starts* I J) is encoded as $[[I_{start} = J_{start}] \wedge [I_{end} < J_{end}]]$

(*equals* I J)[81] is encoded as

$[[I_{start} = J_{start}] \wedge [I_{end} = J_{end}] \wedge \neg [I_{end} = \text{Max}] \wedge \neg [J_{start} = \text{Max}]]$

(*during* I J) is encoded as

$[[J_{start} < I_{start}] \wedge [I_{start} < I_{end}] \wedge [I_{end} < J_{end}]]$

(*finishes* I J) is encoded as $[[J_{start} < I_{start}] \wedge [I_{end} = J_{end}]]$

Interval Composition Operations

Note that I or J can be empty interval.

(overlap_of I J) = TI ≡

$$[\,[\,I_{start}(I,T_i) >= I_{end}(J,T_i)\,] \vee [\,I_{start}(J,T_i) >= I_{end}(I,T_i)\,] \Leftrightarrow [\,I_{start}(TI,T_i) = \text{Max}\,] \wedge [\,I_{end}(TI,T_i) = \text{Max}\,]\,]$$

$$\wedge\ [\,[\,I_{start}(I,T_i) >= I_{start}(J,T_i)\,] \Leftrightarrow [\,I_{start}(TI,T_i) = I_{start}(I,T_i)\,]\,]$$

$$\wedge\ [\,[\,I_{start}(J,T_i) > I_{start}(I,T_i)\,] \Leftrightarrow [\,I_{start}(TI,T_i) = I_{start}(J,T_i)\,]\,]$$

$$\wedge\ [\,[\,I_{end}(I,T_i) <= I_{end}(J,T_i)\,] \Leftrightarrow [\,I_{end}(TI,T_i) = I_{end}(I,T_i)\,]\,]$$

$$\wedge\ [\,[\,I_{end}(J,T_i) < I_{end}(I,T_i)\,] \Leftrightarrow [\,I_{start}(TI,T_i) = I_{start}(J,T_i)\,]\,]$$

- The 1st conjunct ensures that if I is before or after J, it returns EI(Empty Interval).
- The 2nd and 3rd conjuncts ensure that the maximum of $I_{start}(I,T_i)$ and $I_{start}(J,T_i)$ is $I_{start}(TI,T_i)$.
- The 4th and 5th conjuncts ensure that the minimum of $I_{end}(I,T_i)$ and $I_{end}(J,T_i)$ is $I_{end}(TI,T_i)$.

(join I J) = TI ≡

$$[\,[\,[\,I_{start}(I,T_i) > I_{end}(J,T_i)\,] \wedge \neg[\,I_{start}(I,T_i) = \text{Max}\,]\,] \vee [\,[\,I_{start}(J,T_i) > I_{end}(I,T_i)\,] \wedge \neg[\,I_{start}(J,T_i) = \text{Max}\,]\,] \Leftrightarrow [\,I_{start}(TI,T_i) = \text{Max}\,] \wedge [\,I_{end}(TI,T_i) = \text{Max}\,]\,]$$

$$\wedge\ [\,[\,I_{start}(I,T_i) = \text{Max} \wedge I_{end}(I,T_i) = \text{Max}\,] \Leftrightarrow [\,I_{start}(TI,T_i) = I_{start}(J,T_i)\,] \wedge [\,I_{end}(TI,T_i) = I_{end}(J,T_i)\,]\,]$$

$$\wedge\ [\,[\,I_{start}(J,T_i) = \text{Max} \wedge I_{end}(J,T_i) = \text{Max}\,] \Leftrightarrow [\,I_{start}(TI,T_i) = I_{start}(I,T_i)\,] \wedge [\,I_{end}(TI,T_i) = I_{end}(I,T_i)\,]\,]$$

$$\wedge\ [\,[\,I_{start}(J,T_i) < I_{start}(I,T_i)\,] \Leftrightarrow [\,I_{start}(TI,T_i) = I_{start}(J,T_i)\,]\,]$$

$$\wedge\ [\,[\,I_{start}(I,T_i) <= I_{start}(J,T_i)\,] \Leftrightarrow [\,I_{start}(TI,T_i) = I_{start}(I,T_i)\,]\,]$$

---

[81] The equality of two interval variables with empty value (EI) is defined as *false*.

$\wedge$ [[$I_{start}(J,T_i)$ < $I_{start}(I,T_i)$] $\Leftrightarrow$ [$I_{start}(TI,T_i)$ = $I_{start}(J,T_i)$]]

$\wedge$ [[$I_{end}(I,T_i)$ >= $I_{end}(J,T_i)$] $\Leftrightarrow$ [$I_{end}(TI,T_i)$ = $I_{end}(I,T_i)$]]

$\wedge$ [[$I_{end}(J,T_i)$ > $I_{end}(I,T_i)$] $\Leftrightarrow$ [$I_{start}(TI,T_i)$ = $I_{start}(J,T_i)$]]

- The 1st conjunct ensures that if I and J are not empty intervals and I is before or after J, it returns EI(Empty Interval).
- The 2nd conjunct ensure that if I is empty interval, TI = J.
- The 3rd conjunct ensure that if J is empty interval, TI = I.
- The 4th and 5th conjuncts ensure that the minimum of $I_{start}(I,T_i)$ and $I_{start}(J,T_i)$ is $I_{start}(TI,T_i)$.
- The 6th and 7th conjuncts ensure that the maximum of $I_{end}(I,T_i)$ and $I_{end}(J,T_i)$ is $I_{end}(TI,T_i)$.

Interval Expression: each interval operation introduces an interval variable, on which next operation is applied. For instance,

(*join* (*overlap_of* I_1 I_2) (*overlap_of* I_3 I_4))

In this interval expression, 3 new intervals are introduced to translate it: the interval, TI_1, generated by (overlap_of I_1 I_2), the interval, TI_2, generated by (*overlap_of* I_3 I_4), and the interval, TI_3, generated by (*join* TI_1 TI_2). Using the new variables, *overlap_of* and *join* are encoded as above. It returns TI_3.

Interval Update Operation

(*set-I* ?IF ?IE) is encoded as

Active(a,T_i) $\Leftrightarrow$ [$I_{start}(?IF,T_i)$ = $I_{start}(?IE,T_i)$] $\wedge$ [$I_{end}(IE,T_i)$ = $I_{end}(IE,T_i)$]

$$[\forall a \in SetEffect(?IF)\ \forall a' \in SetEffect(?IF) \quad \neg \textbf{\textit{Active}}(\textbf{\textit{a}}, \textbf{\textit{T}}_i) \quad \vee \quad \neg \textbf{\textit{Active}}(\textbf{\textit{a'}}, \textbf{\textit{T}}_i)\ ]$$

- The set operations on the same interval fluent at the same time should be mutually exclusive.

Appendix F: Non-Interference Rules in PDDL+

Define the sets [FoxLong03] :

GPre(a) the set of ground atoms that appear in the precondition

Add(a) the set of ground atoms that are asserted as positive literals in the post-condition

Del(a) the set of ground atoms that are asserted as negative literals in the post-condition

L(a) the set of ground functions that appear as lvalue in a

R(a) the set of ground functions that appear as rvalue in a

L*(a) the set of ground functions that appear as lvalue in an additive assignment in a

Non-Interference Rules in PDDL+

Rule 1: $\boldsymbol{GPre}(a) \cap (\boldsymbol{Add}(b) \cup \boldsymbol{Del}(b)) = \boldsymbol{GPre}(b) \cap (\boldsymbol{Add}(a) \cup \boldsymbol{Del}(a)) = \phi$

- *No moving targets rule* for propositional fluent

Rule 2: $\boldsymbol{Add}(a) \cap \boldsymbol{Del}(b) = \boldsymbol{Add}(b) \cap \boldsymbol{Del}(a) = \phi$

Rule 3: $\boldsymbol{L}(a) \cap \boldsymbol{R}(b) = \boldsymbol{R}(a) \cap \boldsymbol{L}(b) = \phi$

- *No moving targets rule* for numeric-valued fluents

Rule 4: $\boldsymbol{L}(a) \cap \boldsymbol{L}(b) \subseteq \boldsymbol{L^*}(a) \cap \boldsymbol{L^*}(b)$

- It means concurrent actions can only update the same numeric-valued fluents if they both do so by additive assignment effects
- An action with *assign* and all other actions with *assign*, *increase* or *decrease* are in mutex relation

Bibliography

[Armando02] Armando, A., Castellini, C., Giunchiglia, E., Giunchiglia, F., and Tacchella, A.. SAT-Based Decision Procedures for Automated Reasoning: a Unifying Perspective, Journal of Automated Reasoning, Vol. 28, no. 2, 2002.

[Allen83] Allen J. F. Maintaining Knowledge about Temporal Intervals, CACM, 26(11):832-843, 1983.

[Allen94] Allen, J., and Ferguson, G., Actions and Events in Interval Temporal Logic, Journal of Logic and Computation, Special Issue on Actions and Processes, 1994.

[Alur93] Alur, R., Courcoubetis, C., Henzinger, T., and Ho, P-H., Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems, Hybrid Systems, Lecture Notes in Computer Science 736, Springer-Verlag, 1993.

[Amon00] Amon, T., Simplifying Formulas of Linear Inequalities with Boolean Connectives, Southern Utah University CS Technical Report 03292000, 2000.

[Armando99] Armando, A., Castellini, C., and Giunchiglia, E., SAT-based procedures for temporal reasoning, In lecture notes in Computer Science, Volume 1809, 1999.

[Audemard02-a] Audemard, G., Cimatti, A., Kornilowicz., A, and Sebastiani, R., Bounded Model Checking for Timed Systems, Proceedings of 22nd Joint International Conference on Formal Techniques for Networked and Distributed Systems(FORTE02), 2002.

[Audemard02-b] Audemard, G., Bertoli, P., Cimatti, A., Kornilowicz, A., and Sebastiani, R., A SAT Based Approach for Solving Formulas over Boolean and Linear

Mathematical Propositions, Proceedings of 18th International Conference of Automated Deduction, CADE'02, 2002.

[Audemard03] Audemard, G., Bozzano, M., Cimatti, A., and Sebastiani, R., Verifying Industrial Hybrid Systems with MathSAT, Workshop on Pragmatics of Decision Procedures in Automated Reasoning - Affiliated to CADE19, 2003.

[Baioletti02] Baioletti, M, Marcugini, S., and Milani, A. DPPLAN: an Algorithm for Fast Solutions Extraction from a Planning Graph. AIPS, 2000.

[Baioletti03] Baioletti, Milani, A., and Poggioni, V. Planning with Fuzzy Resources. Italian AI conference, 2003.

[Bayardo97] Bayardo, R., and Schrag, R., Using CSP Look-Back Techniques to Solve Real-World SAT Instances, Proceedings of National Conferences on Artificial Intelligence, 1997.

[Bedrax03] Bedrax-Weiss, T., McGann, C., and Ramakrishnan, S., Formalizing Resources for Planning, Workshop on PDDL, ICAPS '03, 2003.

[Blum97] Blum, A. and Furst, M. Fast Planning through Planning Graph Analysis, Artificial Intelligence 90, 1997.

[Bockmayr98] Bockmayr, A., and Dimopouos, Y., Mixed Integer Programming Models for Planning Problems, Workshop on Constraint Problem Reformulation in Constraints Programming '98, 1998.

[Bockmayr99] Bockmayr, A., and Dimopouos, Y., Interger Progams and Valid Ineqaulities for Planning Problems, ?, 1999.

[Borning98] Borning, A., and Badros, G., The Cassowary Linear Arithmetic Constraint Solving Algorithm: Interface and Implementation, Technical Report UW-CSE-98-06-04, 1998.

[Brafman01-a] Brafman, R., A Simplifier for Propositional Formulas with Many binary Clauses, Proceedings of International Joint Conference on Artificial Intelligence, 2001.

[Brafman01-b] Brafman, R., On Reachability, Relevance, and Resolution in the Planning as Satisfiability Approach, Journal of Artificial Intelligence Research 14, 2001.

[Brenner01] Brenner, M., A Formal Model for Planning with Time and Resources in Concurrent Domains, Proceedings of International Joint Conference of Artificial Intelligence, 2001.

[Castellini01] Castellini, C., Giunchiglia, E., and Tachella, A., Improvements to SAT-based Conformant Planning, Proceedings of European Conference on Planning, 2001.

[Chittaro02] Chittaro, L, Montanari, A., Temporal Representation and Reasoning in Artificial Intelligence: Issues and Approaches, Baltzer Journals, July 2002.

[Coddington02] Coddington, A., Fox, M., Long, D., Handling Durative Actions in Classical Planning Frameworks, Proceedings of the AIPS 2002 workshop on Planning for Temporal Domains, 2002.

[Davis62] Davis., M., Longermann, G., and Loveland, D., A machine program for theorem proving, Journal of the ACM, 5(7), 1962.

[Davis90] Davis, E. Representations of Common Sense Knowledge, Morgan Kaufmann, 1990.

[Davis92] Davis, E. Axiomatizing Qualitative Process Theory, KR '92, 1992.

[Davis94] Davis, E. Branching Continuous Time and the Semantics of Continuous Action, AIPS '94, 1994.

[Dean88] Dean, T., Firby, J., and Miller, D., hierarchical Planning involving deadlines, travel times and Resources, Computational Intelligence, 4(4), 1988.

[Dechter91] Dechter R., I. Meiri, and J. Pearl, Temporal Constraints Networks, AI, 49:61-95, 1991.

[DimopoulosGerevini02] Dimopoulos, Y., and Gerevini, A., Temporal Planning through Mixed Integer Programming: A Preliminary Report, Proceedings of 8th Conf. on Principle and Practice of Constraint Programming (CP 2002), 2002.

[Do00] Do, M.B., Srivastava, B., and Kambhampati, S. Investigating the Effect of Relevance and Reachibility Consytraint on SAT Encoding of Planning,?,2000(?)

[Do03-a] Do, M.B., and Kambhampati, S., Improving the Temporal Flexibility of Position Constrained Metric Temporal Plans, ICAPS '03, 2003.

[Do03-b] Do, M.B., and Kambhampati, S., Sapa: A Scalable Multi-Objective Metric Temporal Planner, Journal of Artificial Intelligence Research, To appear, 2003.

[DoKambhampati00] Do, M.G. and Kambhampati, S. Solving Planning-Graph by Compliing it into CSP. AIPS, 2000.

[Drabble93] Drabble, B., EXCALIBUR: A Program for Planning and Reasoning with Processes, Artificial Intelligence, 1993.

[Ernst97] Ernst, M., Millstein, T., and Weld, D., Automatic SAT-Compilation of Plannning Problems, Proceedings of International Joint Conference Artificial Intelligence, 1997.

[Farquhar94] Farquhar, A., A Qualitative Physics Complier, Proceedings of 12th AAAI, 1994.

[Forbus89] Forbus., K., Introducing Actions into Qualitative Simulation, Proceedings of International Joint Conference of Artificial Intelligence, 1989.

[Forbus96] Forbus, K., Qualitative Reasoning, In A.B. Tucker, editor, The Computer Science and Engineering Handbook, 715--733. CRC Press, 1996.

[FoxLong99] Fox, M., and Long, D., The Detection and Exploitation of Symmetry in Planning Domains, Proceedings of the 15th International Joint Conference on AI (IJCAI), Morgan Kaufmann, 1999.

[FoxLong00] Fox, M., and Long, D., Utilizing Automatically Inferred Invariants in Graph Construction and Search, AIPS '00, 2000.

[FoxLong01] Fox, M., Long, D., PDDL+ level 5: An Extension to PDDL2.1 for Modelling Planning Domains with Continuous Time-dependent Effects, Unpublished manuscript, 2001.

[FoxLong02-a] Fox, M., and Long, D, PDDL+ : Modelling Continuous Time-dependent Effects, Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space, 2002.

[FoxLong02-b] Fox, M., Long, D., PDDL2.1 : An Extension to PDDL for Expressing Temporal Planning Domains, International Planning Competition , 2002.

[FoxLong02-c] Fox, M., Long, D., The 3rd International Planning Competition: Temporal and Metric Planning, Fifth International Conference on AI Planning and Scheduling, 2002.

[FoxLong02-d] Fox, M., Long, D., Bradley, B., and McKinna, J., Using Model Checking for Pre-Planning Anaysis, Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space, 2002.

[FoxLong02-e] Fox, M., and Long, D., Extending the Exploitation of Symmetry Analysis in Planning, Fifth International Conference on AI Planning and Scheduling, 2002.

[FoxLong03] Fox, M., Long, D., PDDL2.1 : An Extension to PDDL for Expressing Temporal Planning Domains, Journal of Artificial Intelligence Research 20, 2003.

[Frank03] Frank, M., Golden, K., and Jonsson, A., The Loyal Opposition Comments on Plan Domain Description Languages, Workshop on PDDL, ICAPS '03, 2003.

[Funge99] Funge, J., Representing Knowledge within the Situation Calculus using Interval-valued Epitemic Fluents, Journal of Reliable Computing, 1(5), 1999.

[Garrido02] Garrido, A., Fox, M., and Long, D., A Temporal Planning System for Durative Actions of PDDL2.1, Proceedings of European Conference on Artificial Intelligence, 2002.

[Gent99] Gent, I., and Walsh, T., The Search for Satisfaction, Internal Report, Dept. of Computer Science, University of Strathclyde, 1999.

[Gerevini00] Gerevini, A., and Schubert, L.K., Discovering State Constraints in DISCOPLAN: Some New Results, Proceedings of the Seventeenth National Conference on Artificial Intelligence, 2000.

[Gerevini03] Gerevini, A., Saetti, A. and Serina, I., Planning through Stochastic Local Search and Temporal Action Graphs, to appear in Jounal of Artificiall Intelligence Research (JAIR), 2003.

[GereviniSerina03] Gerevini, A., Saetti, A. and Serina, I., Planning as Propositional CSP: From Walksat to Local Search Techniques for Action Graphs . Constraints, 8, 2003.

[Ghallab00] Ghallab, M., Planning and Scheduling with Time and Resources, PLANET Summer School, 2000.

[Ghallab94] Ghallab, M. and Laruelle, H., Representation and Control in IxTeT, a Temporal Planner, Proceedings of 2nd International Conference of AI Planning Systems, 1994.

[Giunchiglia00] Giunchiglia, E., Planning as satisfiability with expressive action language: Concurrency, constraints, and nondeterminism, 7th International Conference on Principles of Knowledge Representation and Reasoning (KR '00), 2000.

[Giunchiglia98] Giunchiglia, E., Massarotto, A., and Sebastiani, R., Act, and the Rest Will Follow: Exploiting Determinism in Planning as Satisfiability, AAAI'98, 1998.

[Giunchiglia00] Giunchiglia, E., Giunchiglia, F., and Tacchella, A., SAT-based Decision Procedures for Classical Modal Logics, Journal of Automated Reasoning, 2000.

[Gomes98] Gomes, C.P., Selman, B., and Kautz, B, Boosting combinatorial search through randomization, Proceedings of 15th National Conference on Artificial Intelligence, 1998.

[Green69] Green, C., Application of Theorem Proving to Problem Solving, Proceedings of IJCAI-69, 1969.

[HaslumGeffner01] Haslum, P., and Geffner, H., Heuristic Planning with Time and Resources, Proceedings of International Joint Conference on Artificial Intelligence, 2001.

[Hendrix73] Hendrix, G., Modeling Simultaneous Actions and Continuous Changes, Artificial Intelligence, 4:145-180, 1973.

[Henzinger96] Henzinger, T., The Theory of Hybrid Automata, Proceedings of the 11th Annual Symposium on Logic in Computer Science(LICS), 278-292, IEEE Computer Society Press, 1996.

[Henzinger97] Henzinger, T., Ho, P., and Wong-Toi, H., HYTECH: a model checker for hybrid systems, International Journal of STTT 1:110-122, 1997.

[Henzinger98] Henzinger, T., Ho, P., and Wong-Toi, H., Algorithmic Analysis of Nonlinear Hybrid Systems, IEEE Transactions on Automatic Control, 1998.

[HoffmannGeffner03] Hoffmann, J. and Geffner, H. Branching Matters: Alternative Branching in Graphplan. ICAPS, 2003.

[IPC3] International Planning Competition 2002 web site:
http://planning.cis.strath.ac.uk/competition/

[IPC4] International Planning Competition 2004 web site:
http://www.informatik.uni-freiburg.de/~hoffmann/ipc-4/

[Hoffmann03] Hoffmann, J., The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numerical State Variables, to appear in Jounal of Artificiall Intelligence Research (JAIR), 2003.

[Kambhampati03] Kambhampati, S., 1001 ways to skin a planning graph for heuristic fun and profit. ICAPS '03 Invited Talk., 2003.

[Kautz00] Kautz, H., and Walser, J., Interger Optimization Models of AI Planning Problems, Knowledge Engineering Review, 15(1), 2000.

[Kautz92] Kautz, H., and Selman, B., Planning as Satisfiability, Proceedings of the 10th European Conference on Artificial Intelligence, 1992.

[KautzMcAllester96] Kautz, H., McAllester, D., and Selman, B., Encoding plans in Propositional Logic, Proceedings of Knowledge Representation and Reasoning, 1996

[Kautz96] Kautz, H., and Selman, Pushing the Envelope: Planning, Propositional Logic and Stochastic Seach, Proceedings of the 13th National Conference on Artificial Intelligence, 1996.

[KautzSelman99] Kautz, H., and Selman, Unifying SAT-based and Graph-based Planning, Proc. 16th Intl. Joint Conf. on AI (IJCAI'99), 1999.

[KautzWalser00] Kautz, H., and Walser, J., Integer Optimization Models of AI Planning Problems, Knowledge Engineering Review, 15(1), 2000.

[KautzWalser99] Kautz, H., and Walser, J., State-space Planning by Integer Optimization, Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99), 1999.

[Koehler98] Koehler, J., Planning under Resource Constraints, ECAI-98, 1998.

[Kuipers01] Kuipers, B., Qualitative Simulation, ?, 2001.

[Laborie02] Laborie, P., Planning with resources, PLANET Summer school, 2002.

[Laborie95] Laborie, P. and Ghallab, M., Planning with sharable resource constraints, Proceedings of 14th International Joint Conference on AI, 1995.

[Larsen97] Larsen, K., Steffen, B., and Weise, C., Continuous Modeling of Real-Time and Hybrid Systems: from concepts to tools, Special Section on Timed and Hybrid Systems, International Journal of STTT 1, 64-85, 1997.

[Li97] Li, C.M., and Anbulagan, Heuristics based on Unit propagation for satisfiability problems, in Proceedings of International Joint Conference on Artificial Intelligence, 1997.

[LongFox00] Long, D, Fox, M., Sebastia, L., Coddington, A., An examination of resources in planning, UK Planning and Scheduling SIG Workshop, December 2000.

[LongFox01] Long, D, Fox, M., Encoding Temporal Planning Domains and Validating Temporal Plans, UK Planning and Scheduling SIG, Edinburgh, 2001.

[LongFox02] Long, D, Fox, M., Bridging the Modeling Gap: Examining the Expressiveness of Planning Domain Description Language, Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space, 2002.

[LongFox03] Long, D., and Fox, M., Exploiting a Graphplan Framework in Temporal Planning, International Conference on Automated Planning Systems, 2003.

[Malik02] Malik, S., The quest for Efficient Boolean Satisfiability Solvers, Invited talk at CAV-CADE , 2002.

[Mali02-a] Mali, A., Encoding Temporal Planning as CSP, Proceedings of IEEE International Conference on Tools with Artificial Intelligence (ICTAI), 2002.

[Mali02-b] Mali, A., DSatz: A Directional SAT Solver for Planning, ?, 2002.

[Mali02-c] Mali, A., On the Hybrid Propositional Encodings of Planning, Computational Intelligence Journal, Vol. 18, No. 3 2002.

[McAllester91] McAllester, D., and Rosenbiltt, D., Systematic Nonlinear Planning, Proceedings of 9th National Conference on Artificial Intelligence, 1991.

[McDermott00] McDermott, D., The 1998 AI Planning Systems Competition, AI Magazine 21(2), 2000.

[McDermott03-a] McDermott, D., The Formal Semantics of Processes in PDDL, Workshop on PDDL, ICAPS '03, 2003.

[McDermott03-b] McDermott, D., Reasoning about Autonomous Processes in an Estimated-Regression Planner, ICAPS'03, 2003.

[McDermott03-c] McDermott, D., OPT Manual Version 1.5, 2003.

[McDermott03-d] McDermott, D., *Commentary* PDDL2.1 – The Art of the Possible? Commentary on Fox and Long, Journal of Artificial Intelligence Research, 20, 2003.

[McDermott83] McDermott, D., A Temporal Logic for Reasoning about Processes and Plans, Cognitive Science, 1983.

[McDermott98] McDermott, D., and the AIPS '98 Plannning Competition Committee, PDDL – the Planning Domain Definition Language, URL: www.cs.yale.edu/homes/dvm, 1998.

[Miller96-a] Miller, R., A Case Study in Reasoning about Actions and Continuous Change, Proceedings of ECAI, 1996.

[Miller96-b] Miller, R. and Shanahan, M., Reasoning about Discontinuities in the Event Calculus, Proceedings of KR '96, 1996.

[Nelson79] Nelson, G. and Oppen, D. Simplification by cooperating decision procedures. ACM Transaction on Programming Languages and Systems, 1(2):245-57, 1979.

[Nguyen02] Nguyen, X., Kambhampati, S. and Nigenda, R.S. , Panning Graph as the Basis for deriving Heuristics for Plan Synthesis by State Space and CSP search, Artificial Intelligence 135, 2002.

[OlderVellino90] Older, W, and Vellino, A., Extending Prolog with Constraint Arithmetic on Real, In Proceedings of IEEE Canadian conference on Electrical and Computer Engineering, 1990.

[Penberthy92] Penberthy, J., and Weld, D., UCPOP: a sound, complete, partial order planner for ADL, Proceedings of 3rd International Conference on Principles of Knowledge Representation and Reasoning, 1992.

[Penberthy93] Penberthy, J., Planning with Continuous Change, Ph.D. dissertation 93-12-01, Deprt. of Computer and Engineering, U. of Washingtopn, 1993.

[Penberthy94] Penberthy, J., and Weld, D., Temporal Planning with Continuous Change, Proceedings of 12th National Conference on Artificial Intelligence, 1994.

[Pinto98-a] Pinto, J., Concurrency and Action Interaction, Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR'98), 1998.

[Pinto98-b] Pinto, J., Integrating Discrete and Continuous Change in a Logical Framework, Computational Intelligence, 1998.

[Reiter01] Reiter, R. Knowledge in Action,: Logical Foundations for Describing and Implementing Dynamic Systems, MIT Press, 2001.

[Reiter96] Reiter, R., Natural Actions, Concurrency and Continuous Time in Situation Calculus, Proceedings of KR ’96, 2-13, Morgan Kauffman, 1996.

[Rintanen00] Rintanen, J., Lecture Notes on Introduction to Automated Planning, 2000.

[Rintanen98] Rintanen, J., A Planning Algorithm not based on Directional Search, Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR '98), 1998.

[Rintanen99] Rintanen, J., and Jungholt, H., Numeric State Variables in Constraint-Based Planning, Recent Advances in AI Planning: 5th European Conference on Planning, ECP'99, 1999.

[Russell03] Russell, S., Norvig, P., Artificial Intelligence: A Modern Approach, the second edition, Prentice Hall, 2003.

[Sandwell89] Sandwell, E., Combining Logic and Differential Equations for Describing Real-World Systems, Proceedings of KR ’89, Morgan Kauffman, 1989.

[SchwalbVila98] Schwalb E., and L. Vila, Temporal Constraints: a survey, Special issue on Spatial and Temporal Reasoning, Constraints, 3(2-3), 1998.

[Sebastiani01] Sebastiani, R., Integrating SAT Solvers with Math Reasoners: Foundations and Basic Algorithms, Technical Report 0111-22, ITC-IRST, 2001.

[Selman93] Selman, B., Kautz, H., and Cohen, B., Local Search Strategies for Satisfiability Testing. Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenges, 1993.

[Selman94] Selman, B., Kautz, H., and Cohen, B., Noise Strategies for Improving Local Search, Proceedings of 10th National Conference on Artificial Intelligence, 1994.

[Shanahan90] Shanahan, M., Representing Continuous Change in the Event Calculus, Proceedings of ECAI, 1990.

[ShinDavis04] Shin, J. and Davis, E. Continuous Time in a SAT-based Planner. Proceedings of 20th National Conference on Artificial Intelligence, 2004.

[Simmons88] Simmons, R., Combining Associations and Causal Reasoning to Solve Interpretation and Planning Problems, AI-TR-1048, MIT AI Lab, 1988.

[Smith03] Smith, D., The Mystery Talks, PLANET summer school. 2003.

[Smith00-a] Smith, D., Coping with Time and Continuous Quantities, slides of invited talk in AIPS '00, 2000.

[Smith00-b] Smith, D., Frank, J., and Jonsson, A., Bridging the Gap between Planning and Scheduling, Knowledge Engineering Review, 15(1), 2000.

[Smith97] Smith, S.F., and Becker, M., An Ontology fro Constructing Scheduling Systems, AAAI Symposium on Ontological Engineering, 1997.

[Smith99] Smith, D. and Weld, D., Temporal Planning with Mutual Exclusion Reasoning, International Joint Conference on Artificial Intelligence, 1999.

[Strichman02] Strichman, O., Optimizations in Decision Procedures for Propositional Linear Inequalities Technical report (CMU-CS-02-133), 2002.

[Thielscher99] Thielscher, M., Fluent Calculus Planning with Continuous Change, ETAI, 1999, URL: http://www.ep.liu.se/ea/cis/1999/011

[Vere83] Vere, S., Planning in Time: Windows and Durations for Activities and Goals, Pattern Analysis and Machine Intelligence 5, 1983.

[Vila94] Vila L., A Survey on Temporal Reasoning in Artificial Intelligence, Artificial Intelligence, 7(1):4-28, 1994.

[Vossen00] Vossen, T., Ball, M., Lotem, A., and Nau, D., Applying Integer Programming to AI Planning, Knowledge Engineering Review, 15(1), 2000.

[Weld90] Weld, D., and Kleer, J., (ed) Qualitative Reasoning about Physical Systems, 1990.

[Weld94] Weld, D., An Introduction to Least Commitment Planning, AI Magazine, Summer/Fall, 1994

[Weld99] Weld, D., Recent Advances in AI Planning, AI Magazine 15(4), 27-61, 1999.

[Wilkins88] Wilkins, D., Practical Planning: Extending the Classical AI planning Paradigm, Morgan Kaufman, 1988.

[Wolfman] Wolfman, S., unpublished manuscript.

[Wolfman00] Wolfman, S., and Weld, D., Combining Linear Programming and Satisfiability Solving for Resource Planning, Knowledge Engineering Review, 15(1), 2000.

[Wolfman99] Wolfman, S., and Weld, D., The LPSAT Engine and its application to Resource Planning, International Joint Conference of Artificial Intelligence, 1999.

[Zhang02] Zhang, L., and Malik, S., The Quest for Efficient Boolean Satisfiability Solvers, Proceedings of 8th International Conference on Computer Aided Deduction(CADE 2002), 2002.