VAL's Progress: The Automatic Validation Tool for PDDL2.1 used in The International Planning Competition

Richard Howey and **Derek Long**
University of Durham, UK
r.a.j.howey@durham.ac.uk d.p.long@durham.ac.uk

Abstract

A key element in the success of the 3rd International Planning Competition was the implementation of an automatic plan validator for PDDL2.1 , VAL. Over 5000 plans were generated by entrants, so it was clearly essential for evaluation of the results alone. VAL was a vital tool in the development process of competitors to develop their planners. This document describes the evolution of VAL to handle more complex types of plans for competitions of the future. In the 3rd IPC the handling of numeric resources was limited in that the values could only be updated discretely. We report on the progress made with extending VAL so that numeric resources can be updated continuously. Also a new reporting option in LaTeX has been implemented including a Gantt chart and graphs of the numerical resources – this greatly increases the usefulness of VAL as a tool for the development of planners using PDDL2.1.

1 Introduction

The 3rd International Planning Competition (Fox, Long, & Committee 2002) was a great success and a cornerstone to this success was the initial definition of a semantics for the language used in the competition, PDDL2.1. This conveyed a general understanding of the semantics of the domains defined using this language and therefore, most importantly, a general understanding of what constitutes a valid plan. With this consensus on what constitutes a valid plan it was possible to implement an automatic plan validator, VAL. Given the domain, problem and plan a definitive answer to whether the plan is valid could be returned, furthermore if the plan fails details of why it failed could be returned. The 3rd IPC produced in excess of 5000 plans, so the availability of an automatic plan checker was essential for evaluation of results alone. However the importance of VAL stretches further than just a competition results evaluation tool – it conveys what is a valid plan in PDDL2.1 to anyone developing a planner using this language. It is therefore an invaluable aid to the planning community in their development of planners using PDDL2.1 for the competition, or any other purpose.

The 3rd IPC only used the discontinuous change of PDDL2.1 which made temporal planning one of the focal objectives, and a number of planners achieved success in handling quite complex metric temporal planning behaviour, including MIPS (Edelkamp & Helmert 2000) and LPG (Gerevini & Serina 2002). Although the introduction of metric temporal reasoning was a considerable challenge, there still remains important challenges. In particular the domains used in the competition were such that all change was modelled using discrete effects. There are features of some domains that cannot be accurately modelled with discrete effects, a more expressive fragment of PDDL2.1 allows the use of continuous effects. This document reports on the progress made in implementing an extension to VAL to include continuous effects, the problems faced in doing this and the semantics of continuous effects in PDDL2.1.

The challenges in validating a plan with continuous effects can be seen to consist of a number of factors: linear effects, non-linear effects, invariants without disjunction and invariants with disjunction. These challenges are addressed in more detail later. Validating plans with continuous effects in a general context has also been considered. Domains that consist solely of effects that are linear can be validated by VAL provided that any expressions appearing in invariant comparisons are polynomial (in time). Non-linear effects present many tougher challenges, nethertheless considerable progress has been made towards validating plans with non-linear effects. Non-linear Effects that are defined as a polynomial function of time can be handled by VAL, again provided that any expressions appearing in invariant comparisons are also polynomial in time.

A new feature to VAL is the option of producing a LaTeX report document as output of validating a plan (or plans). This report greatly enhances the communication of the results from VAL versus simple usage from the command prompt. Of course if the only goal is to simply check a plan and get a *yes or no* answer then basic usage of VAL is appropriate. However the report brings the ability to be able to dissect a plans validation, showing the original plan, the implied *plan to validate*, and a blow by blow account of the validation. The report also includes a Gantt chart which clearly highlights a plans temporal structure showing concurrent activity and dependency. This may make visible any improvements that could be made to a plan or any flaws in the overall planning strategy. Another feature to the report are graphs of the numerical quantities that change throughout the execution of a plan. These show the effects of continuous change, discrete updates, concurrent activity and the interaction of numerical quantities.

2 Actions with Continuous Effects

We assume that time is continuous and real-valued. A numerical quantity that can be changed, a function in PDDL, is called a *primitive numerical expression* (PNE[1]). These PNEs can have continuous change initiated with discrete changes made to the values of their (time) derivatives. Only a durative action may affect the derivative of a PNE, the discrete effect to the derivative starts at the beginning of the durative action and ends at the end of the durative action. The continuous change made by a durative action thus affects a finite interval within the plan structure. The use of derivatives for continuous effects can be seen as instantaneous effects, despite the fact that they lead to continuous change.

In PDDL2.1 any invariant condition must hold over the duration of a durative action. In the absence of continuous change the invariant can be checked by evaluating a set of predicates between the start and end of the durative action. However if an invariant condition depends on any PNEs that are changing continuously it is not possible to guarantee that an invariant holds by a single evaluation of predicates. This is discussed further in section 7.

A continuous effect can only affect metric quantities: it is not possible to change a propositional fluent continuously. A durative action that has a continuous effect on a PNE changes the fluent so that the values taken once the continuous change is activated are described by a continuous function of time. That is if v is changing continuously on an interval $[t_1, t_2]$ then for each $t' \in [t_1, t_2]$ the limit $\lim_{t \to t'} v(t)$ exists and is equal to $v(t')$. It is possible for other actions to affect a PNE during the interval over which a continuous effect is changing the same expression. In this case, the compound continuous effect will be decomposed into segments of continuous behaviour, punctuated by points of discrete change. These points can be either discrete changes in the value of the expression itself, where an action assigns directly to the PNE, so that the values describe a discontinuous behaviour, or can be discrete changes in the rate of change so that the values describe a continuous, but non-differentiable behaviour. The latter case occurs when an action makes an (instantaneous) assignment to the derivative of a PNE.

3 Syntax of Continuous Effects

A continuous effect of a durative action is written in the following style:

```
(increase (fuel-level ?v) (* #t
      (refuel-rate ?p)))
```

where #t represents, in some sense, the time over which the effect has been active. However, the whole expression must be interpreted as having a special significance that cannot be accurately captured in terms of an instantaneous effect: instead, the expression should be thought of as identifying a rate of change for the primitive numeric expression on its left. In this example the fuel level of `v` is continuously increased at the refuel rate of the fuel pump `p`. If the fuel pump delivers fuel at a constant rate then the fuel level at any point during refueling is given by the time since refueling started, #t, multiplied by the refuel rate. If the refuel rate is itself changing then the behaviour is more complex, described by a differential equation. It might seem more natural to express this effect as an assertion of the form

$$\frac{d}{dt}(\text{fuel-level ?v}) = (\text{refuel-rate ?p})$$

but this would be inappropriate since there might be additional actions that affect the value of the fuel level continuously and concurrently. While it would not be inconsistent for each of those actions to assert that they had the effect of increasing fuel level at some rate, it would be inconsistent for any of them to assert a specific value for the overall rate of change of the fuel level.

Formally continuous effects are written as follows:

`(<assign-op-t> <f-head> <f-exp-t>)`

where

```
<assign-op-t>  ::= increase
<assign-op-t>  ::= decrease
<f-exp-t>      ::= (* <f-exp> #t)
<f-exp-t>      ::= (* #t <f-exp>)
<f-exp-t>      ::= #t
```

and `<f-head>` is a PNE and `<f-exp>` is any expression (for details see PDDL2.1 (Fox & Long 2002)).

4 Semantics of Continuous Effects

A detailed description of the semantics *without* continuous effects of PDDL2.1 can be found in (Fox & Long 2002). The semantics is developed in terms of a reasonably familiar finite-state-machine-transition model, complicated only by the extension of the classical logical states (represented as sets of literals that each hold true in a state) to include a valuation for PNEs (or, in other words, an assignment of numeric values to the PNEs that exist for the problem). All transitions are instantaneous, just as in a classical semantic model for STRIPS-like languages, although actions that bear the same timestamp are required to be pairwise non-mutex and then their effects are applied simultaneously. Durations are simply handled as numeric constraints (in fact, the duration constraints of the durative actions themselves) to be tested in the appropriate state. Thus, durative actions are given semantics by translation into instantaneous actions corresponding to the start and end points of the durative action.

Invariants are handled using special actions (again, instantaneous) that are inserted, between other pairs of time points at which there are any actions, between the start and end points of the durative action to which they apply. Since without continuous effects, all change is constrained to be discrete and to occur at an end point of an action, it is clearly sufficient to check invariants just once between every pair of points at which each invariant might have been affected. We call all the time points at which instantaneous actions occur *happenings*, including those points corresponding to end points of durative actions.

A formal semantics has been developed to include continuous effects based on the semantics mentioned

[1]For historical reasons a PNE is called a FE (functional expression) in the code.

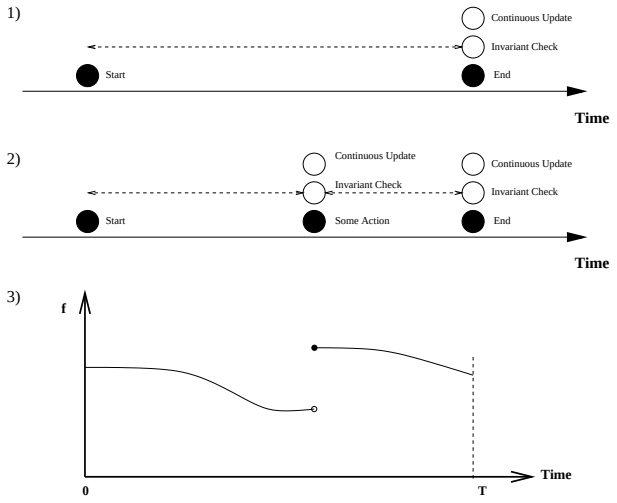

Figure 1: Durative action with continuous effects. Graph shows the values of a PNE, f, which is changing continuously during the execution of the durative action.

above (Howey & Long 2003a). We do not attempt to give a formal semantics in this section due to limited space. The introduction of continuous change creates two further complications:

- Continuous changes can interact with one another
- Invariant conditions may depend on values that are continuously changing

The key extension to the discrete temporal model is that interacting processes are described by systems of differential equations, whose effects can be resolved at the conclusion of each interval over which they act uninterrupted. Invariants can be checked by considering the functions describing the change in variables over the same intervals. This is illustrated in figure 1. In (1) we show how the interval of a durative action effecting continuous change can be handled by updating the continuously changing variables discretely at the end of the interval and checking any invariants at this point (respecting the continuous change). In (2) we show that if another action punctuates the interval then the evaluation of continuous effects and invariant checks are managed at each point of change. Part (3) illustrates how discrete affects can arise, due to parallel activity during the intervals, breaking the continuous change into piece-wise continuous components.

Implementation of Semantics

Given the original plan of timestamped durative actions (with a given duration) and non-durative actions VAL maps them into a sequence of happenings. Each happening is labelled with a time value and contains a list of actions, each action is either a non-durative action or an end point of a durative action. (Two actions with timestamps within a certain tolerance are considered to occur at the same time). For this list of actions to be executed simultaneously some restrictions must be met in order that the result is clearly defined. This is achieved by ensuring that the actions are commutative, that is any order of execution of the actions always yields the same result. This constraint implies that propositions and PNEs are treated like shared memory in multiprocessing operating systems, with actions analogous to separate processes. An action precondition demands *read-access* to all of the atomic propositions and PNEs it includes, while action effects demand *write-access* to all the atomic propositions and PNEs they refer to. VAL refers to an action having *ownership* of an atomic proposition or PNE for deletion or assignment etc. Each action in turn is designated with ownerships for the appropriate atomic propositions and PNEs, these are then compared with existing ownerships of other actions for any conflicts. For example an atomic proposition can support any number of precondition ownerships and PNEs can support any number of additive updates (since they commute, subject to minor restrictions). However one action having a precondition ownership of an atomic proposition and another having a deletion ownership is not acceptable, since the order in which the actions can be applied conflicts. (If two actions conflict in their ownership they are mutex).

VAL has an iterator that progresses through the happenings that are taken directly from the durative action end points and the non-durative actions. These happenings are referred to as *regular* happenings. On an interval of continuous activity the iterator also returns two other types of happenings before each regular happening, these happenings have the same timestamp as the regular happenings. (This is illustrated in figure 1 part (2)). The first is a *continuous* happening in which the functions of time describing the continuously changing PNEs are calculated, see section 6. Crucially the continuous effects are all considered together, not on an action by action basis since continuous effects may interact with one another. These functions are then used to update the values of the PNEs correctly at this time point. The second happening is an *invariant* happening in which all the active invariants are checked on the open interval from the last regular happening with respect to any continuously changing PNEs, see section 7. This happening has no effects and only verifies invariant conditions so either the conditions are true and the plan continues, or an invariant condition fails and the plan fails. This happening may also require the functions of time describing the continuously changing PNEs, and so they are only calculated once at this time.

5 Implementation of Plan Verification

In figure 3, we show the dimensions that affect the problem of validating a plan with continuous effects and summarize the interactions between them. The first dimension is the complexity of the functions describing the continuous change. These range through constant (no continuous change), linear, polynomial and other (which includes exponential functions and transcendental functions). The second and third dimensions are defined by the invariant structure, with complexity affected by the structure of functions used in the comparisons and by the existence (or not) of disjunctions. In the following sections we consider this categoriza-

Invariant function order (in PNEs): Continuous Effects	None	Linear	Polynomial
None	The invariant collapses to a boolean value in these cases.		
Linear	1	2	3
Polynomial	1	3	3
Other Functions	4	For all entries in this row, see note 5.	

1. These cases involve solving simple differential equations and evaluating the results at a point.
2. This case can be resolved by checking the condition at the end points of the interval $(0, T)$ (this follows from linearity).
3. These cases reduce to checking whether a polynomial has roots within the interval $(0, T)$.
4. This case can be handled by numeric integration, since the roots are not required to check invariant comparisons.
5. All of these cases involve solving differential equations which could yield solutions in exponential, logarithmic, trigonometric or hyperbolic functions. These introduce problems that cannot be guaranteed to be solved.

Figure 3: Interactions between continuous effects and forms of invariants.

Time	**Action**
1:	(generate generator) [100]
20:	(refuel generator tank) [10]

Time	**Happening**
1:	(generate generator) - *start*
20:	*Invariant for* (generate generator)
20:	*Update of continuously changing Primitive Numerical Expressions*
20:	(refuel generator tank) - *start*
30:	*Invariant for* (generate generator)
	Invariant for (refuel generator tank)
30:	*Update of continuously changing Primitive Numerical Expressions*
30:	(refuel generator tank) - *end*
101:	*Invariant for* (generate generator)
101:	*Update of continuously changing Primitive Numerical Expressions*
101:	(generate generator) - *end*

Figure 2: Durative actions in a plan and the corresponding simple actions to be validated.

tion and the interactions between them in more detail.

6 Interacting Continuous Effects

There may be a number of continuous effects active at one time each of which additively modifies the derivative of a PNE. If a PNE has its derivative modified more than once then the derivative is given by the sum of the contributions. The rate of change of a PNE may also depend on the value of other PNEs which may themselves be continuously changing. The values of all the changing PNEs are thus given by a system of differential equations:

$$\frac{df_i}{dt} = g_i(f_1, f_2, \cdots, f_n) \qquad i \in \{1, 2, \cdots, n\},$$

where the f_i are the PNEs and the g_i are some functions depending on the set of continuously changing PNEs. PNEs that are not changing continuously are treated as constants. For example consider the following continuous effects

```
increase (distance ?c) (* #t (speed ?c))
increase (speed ?c) (* #t (acceln ?c))
```

which describe the motion of a car driving. The rate of change of the PNE for the distance of the car is given by the PNE for the speed of the car. The PNE for the speed of the car is in turn given by the PNE for the acceleration of the car. To solve these differential equations to give the functions of time describing the motion of the car we must firstly determine the acceleration, then the speed, and lastly the distance of the car. (See figures 8, 9 and 10 for graphs of these effects).

A PNE can change as a linear function of time, a polynomial function of time or even a more complex function of time, due to more complex dependencies arising in the expression on the right-hand-side of the differential equation governing its evolution. If the dependencies of the PNEs are such that there are no loops, that is the rate of change of any PNE does not depend on itself either directly or indirectly, then it can be shown that the PNEs will be described by polynomials in t, the time over which the action is executing.

Determination of the structure of the dependency sets can be carried out automatically, using syntactic analysis of the expression parse trees. A graph is constructed using PNEs as vertices and with directed edges between the expressions on the left of continuous updates and those on the right of the same expression. If this graph is acyclic then the differential equations can all be solved with polynomials. The only limitation is that this dependency analysis carried out purely syntactically can be conservative: it can be the case that dependencies actually simplify away if expressions can be symbolically manipulated to cancel terms. Since this kind of manipulation is non-trivial, we must assume that the dependencies discovered by syntactic analysis could be more restrictive than the true dependencies.

The complexity of the differential equations that can be expressed far exceeds the practical possibility of solving them analytically and even numerically. It is therefore necessary to impose certain restrictions on the differential equations to guarantee that they can be solved.

7 Invariants

Continuous effects have their most significant effect on the verification of plans when they interact with invariants. An invariant condition must be evaluated on an interval by checking that the continuously changing PNEs that appear within it do not assume values that lead to a violation of the invariant.

One-Clause Invariants

An invariant comparison containing PNEs that are continuously changing can always be expressed as a function of time, t, that must be greater than zero (or perhaps greater then or equal to zero. If equality is used then the difference between the left hand side and the right hand side cannot vary in time for the invariant to hold). For example

$$t^2 + 2t + 2 > 0 \qquad \text{for } t \in (0, 5)$$

may be an invariant condition to check. If the invariant expression is linear in time we can simply evaluate the expression at the end points of the interval to confirm the condition holds. However checking an invariant condition with a non-linear expression in time it is no longer sufficient to check the condition at end points only. An invariant comparison $F(t) > 0$ on $(0, T)$, where F is some continuous function in time, t, can be checked by one of the following methods:

1. Check that $F(0) \geq 0$ and $F(T) \geq 0$ and also check that the value at any stationary points in $(0, T)$ is greater than zero.
2. Check that $F(0) \geq 0$ and $F(T) \geq 0$ and also check to see if any roots exist in $(0, T)$. (If the inequality is non-strict then care is needed in case there is a stationary point).

Method 2 is the method chosen to check non-linear invariants in VAL. The key to the problem of checking invariants that are comparisons with non-linear expressions in t is one of finding the roots of a non-linear function. This problem is, in general, non-trivial, even in the case of polynomials. There are many algorithms to find the roots of equations but we need to be sure of finding all the roots in a given interval in every possible case. It is therefore necessary to impose certain restrictions on the invariants that may be expressed to guarantee that they may be verified on a given interval.

We are initially only considering invariant comparisons which depend on continuously changing PNEs that are given by polynomials in t. For one-clause invariant comparisons which are given by an inequality that is strict we are in fact only interested in the existence of real roots on a given open interval. This can be determined by methods based on Sturm's Theorem or Descartes' Rule of Signs Theorem, with the provision that there are no repeated roots. Fortunately for any given polynomial we can be obtain another polynomial with the same roots but without any repeated roots. This is achieved by dividing the polynomial by the greatest common divisor of itself and its derivative. To find the greatest common divisor of two polynomials the Euclidean (a.k.a. Euclid's or divisional) algorithm can be used, although the accuracy of the coefficients must be handled carefully to avoid any spurious results due to rounding in the calculations.

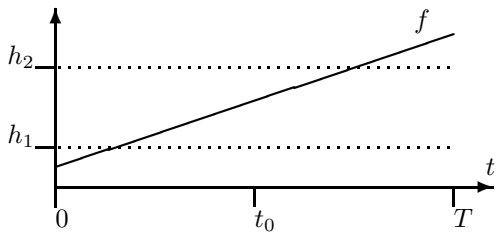

Figure 4: Example of h_1, h_2 and f. If f is required to be above h_2 *or* below h_1 across $(0, T)$, then the value at t_0 breaks the constraint.

For one-clause *non-strict* invariant comparisons to determine the existence of real roots on a given interval alone is not sufficient to check the invariant – since we may have a repeated root so that the invariant condition is not broken. Fortunately the invariant can be checked by applying an algorithm based on Sturm's Theorem or Descartes' Rule of Signs Theorem to obtain a set of intervals that isolate the roots (if there are any). Then checking the sign of the polynomial between the roots will confirm that the invariant has not been violated. Therefore the size of the intervals is not important provided that they do not overlap.

Invariants with Disjunctions

Let A and B be two atoms that depend on time then consider the two conditions

$$A(t) \vee B(t) \text{ for all } t \in (0, T), \tag{1}$$

$$(A(t) \text{ for all } t \in (0, T)) \vee (B(t) \text{ for all } t \in (0, T)). \tag{2}$$

It could be the case that $A(t)$ is only satisfied on $(0, \frac{3}{4}T]$ and $B(t)$ on $[\frac{1}{4}T, T)$ so that condition (1) is satisfied but condition (2) is not. Clearly the two interpretations are not equivalent. Suppose a durative action continuously updates a PNE, f, and there is a concurrent action (possibly the same action) with an invariant condition of the form:

$$f(t) < h_1(t) \wedge f(t) > h_2(t) \text{ for all } t \in (0, T),$$

where h_1 and h_2 are some functions that may depend on other functional expressions. The condition can then be checked by checking each comparison separately. However suppose the condition is of the form:

$$f(t) < h_1(t) \vee f(t) > h_2(t) \text{ for all } t \in (0, T),$$

then each comparison cannot be considered separately.

In the simple case where h_1 and h_2 are constants and f is linear we can no longer simply check the end points of $h_1 - f$ and $f - h_2$ to be greater than zero. This is because we could have $(h_1 - f)(0) > 0$ and $(f - h_2)(T) > 0$ which satisfies the condition at 0 and T but there could exist a point $t_0 \in (0, T)$ such that $(h_1 - f)(t_0) < 0$ and $(f - h_2)(t_0) < 0$ (see figure 4).

In general an invariant condition can be considered as a proposition in DNF that must hold over an interval, say

$(0, T)$. To confirm the invariant, we must then find a set of intervals, C, covering $(0, T)$, such that a disjunct is satisfied in each interval in C. This is simplified if it is possible to find all the roots of all the continuous functions involved, since these points can be used as the end points of the sub-intervals forming the cover. We are initially only considering polynomials and to find the real roots efficiently, infallibly and to within a degree of accuracy we can use an algorithm based on Descartes' Rule of Signs Theorem (Johnson & Krandick 1997). We have recently written an implementation of a polynomial root finder based on these theorems, and in particular algorithms based on research at INRIA (Rouillier & Zimmermann 2001). Using this approach means that our validation of plans cannot be more accurate than the degree of precision used in the solver. However in practice, all numerical testing, even for linear functions, is subject to the degree of precision supported by our machines (always finite), so the problem of plan verification must always be qualified by an observation of the limitation on the accuracy with which numeric constraints can be checked. Therefore, entries labelled 3 in figure 3 are solvable.

More complex non-linear functions remain difficult: it is not possible to guarantee that there is an algorithm to confirm that an arbitrary non-linear function does or does not have roots in a specified range. This makes checking all situations labelled 5 in the table in figure 3 problematic.

As an example of an invariant with disjunction consider the following:

$$(t^2 - 9t + 14 \geq 0) \vee ((t - 1 > 0) \wedge (-t + 8 \geq 0))$$

for t in $(0, 10)$. We must find the values of t in $(0, 10)$ each disjunct is satisfied on then take the union of the two and see if the result covers $(0, 10)$, which implies the result is in fact equal to $(0, 10)$. The first disjunct, $(t^2 - 9t + 14 \geq 0)$, is satisfied for values of t in $(0, 2] \cup [7, 10)$. The second disjunct is not so straightforward, the condition $(t - 1 > 0)$ is satisfied on $(1, 10)$ and $(-t + 8 \geq 0)$ is satisfied on $(0, 8]$ then taking the intersection of the two we have the disjunct being satisfied for values of t in $(1, 8]$. Now taking the union of the values that the two disjuncts are satisfied on gives $(0, 10)$ which indeed covers $(0, 10)$ showing that the invariant condition holds.

Two aspects of the syntactic form of invariants affect the complexity of checking them: one is the complexity of the functions that appear in the invariant proposition itself and the other is whether or not there is more than one disjunct in the proposition.

8 Validation Example: Tanks of Water

To demonstrate some of the capabilities of VAL it is useful to consider an example. So let us consider the problem of filling a bucket of water as quickly as possible from two tanks of water. Firstly we must model the problem sufficiently accurately. The rate of the flow of the water out of the bottom of a tank is governed by Torricelli's Law: *Water in an open tank will flow out through a small hole in the bottom with the velocity it would acquire in falling freely from the water level to the hole*. The volume of water, V, in a tank with the tap left on is then shown to be given by

$$V = (-kt + \sqrt{U})^2 \qquad t \in \left[0, \frac{\sqrt{U}}{k}\right],$$

where k is the flow constant (which depends on gravity, the size of the tap and the cross section of the tank) and U is the initial volume of water in the tank. This then gives the rate at which the water level of the tank is changing as

$$\frac{dV}{dt} = 2k(kt - \sqrt{U}) \qquad t \in \left[0, \frac{\sqrt{U}}{k}\right].$$

At time 0 the tank has volume U and starts to drain then by time $\frac{\sqrt{U}}{k}$ the tank is empty so that these equations are only valid for time values, t, in $[0, \frac{\sqrt{U}}{k}]$. A possible encoding in PDDL2.1 is given as follows:

```
(define (domain tank-domain)
(:requirements :fluents :durative-actions
                  :duration-inequalities)
(:predicates (draining ?t) (filling ?b))
(:functions (volume ?t) (drain-time ?t)
          (flow-constant ?t) (capacity ?b)
          (sqrtvolinit ?t) (sqrtvol ?t))

(:durative-action fill-bucket
 :parameters (?b ?t)
 :duration (<= ?duration
   (/ (sqrtvolinit ?t) (flow-constant ?t)))
 :condition (and
   (over all (<= (volume ?b) (capacity ?b)))
   (at start (not (draining ?t)))
   (at start (not (filling ?b))) )
 :effect (and
   (at start (assign (drain-time ?t) 0))
   (at start (assign (sqrtvol ?t)
                     (sqrtvolinit ?t)))
   (at start (draining ?t))
   (at start (filling ?b)
   (increase (drain-time ?t) (* #t 1))
   (decrease (volume ?t) (* #t
     (* (* 2 (flow-constant ?t))
        (- (sqrtvolinit ?t)
           (* (flow-constant ?t)
              (drain-time ?t)))) ))
   (decrease (sqrtvol ?t)
             (* #t (flow-constant ?t)))
   (increase (volume ?b) (* #t
     (* (* 2 (flow-constant ?t))
        (- (sqrtvolinit ?t)
           (* (flow-constant ?t)
              (drain-time ?t)))) ))
   (at end (assign (sqrtvolinit ?t)
                   (sqrtvol ?t)) )
   (at end (not (draining ?t)))
   (at end (not (filling ?b))))))
```

The domain consists of one durative action which models the filling of one bucket from one tank for a fixed time, along with the condition that the bucket must not overflow. The initial square root of the volume of the tank, `sqrtvolinit`, is required for the equation describing the flow of water out of the tank. This must be given in the first instance since

PDDL2.1 does not handle the use of square roots (yet?). The square root of the volume of the tank, `sqrtvol`, can then be tracked by a linear function of time which then can be used to supply the initial square root of the volume of the tank in the case that the tank is partially drained and then drained later.

Now consider the problem of filling one bucket from two tanks defined in PDDL2.1 as follows:

```
(define (problem tank-problem)
    (:domain tank-domain)
    (:objects tank1 tank2 bucket)
    (:init (= (volume bucket) 0)
           (= (capacity bucket) 60)
           (= (volume tank1) 100)
           (= (sqrtvolinit tank1) 10)
           (= (flow-constant tank1) 0.8)
           (= (volume tank2) 64)
           (= (sqrtvolinit tank2) 8)
           (= (flow-constant tank2) 1))
     (:goal (> (volume bucket)
               (- (capacity bucket) 2)))
     (:metric minimize (total-time)))
```

The two tanks are such that the initial flow from each is equal so the best plan is not simply a matter of choosing the best tank to use. The fastest plan must use each tank in turn, but how long should each tank be used? Calculating the best plan turns out to be non-trivial and would pose a challenge to planners trying to handle non-linear effects. The solution is in fact given by the solution of a quadratic equation which may not be surprising considering the volume of the tanks of water are given by quadratic equations. But what is the correct quadratic equation to solve? How could a planner be expected to deduce the correct equation to solve? And how should a planner interpret the solution?

Now let us consider the following plan to fill the bucket in some detail:

```
0.001: (fill-bucket bucket tank1) [4.5]
```

The first step to validating the plan is to map the plan to the implied plan to validate which is as follows:

Time	**Happening**
0.001:	(fill-bucket bucket tank1) - *start*
4.501:	*Invariant for* (fill-bucket bucket tank1)
4.501:	*Update of continuously changing Primitive Numerical Expressions*
4.501:	(fill-bucket bucket tank1) - *end*

Firstly the start of durative action is applied which checks all '`at start`' conditions and applies all '`at start`' effects. The duration condition is also checked at the start, which in this case ensures that the duration is no greater than the time it takes the tank to empty. Next the invariant conditions are checked on the open interval from the start of the durative action to the end of the durative action. There is only one invariant condition to check which is that the bucket must not overflow. The volume of water in the bucket is given by $-0.64t^2 + 16t$ so that the condition which must be checked is $-0.64t^2 + 16t \leq 60$ for t in $(0, 4.5)$. This can be shown to hold by checking that the inequality holds at the end points and the equation $0.64t^2 - 16t + 60 = 0$ has no roots (except perhaps repeated roots) in the interval $(0, 4.5)$. Next the continuously changing PNEs are updated, so for example the volume of the bucket is given by $-0.64 \times 4.5^2 + 16 \times 4.5 = 59.04$. Finally the end of the durative action is applied checking all '`at end`' conditions and applying all '`at end`' effects.

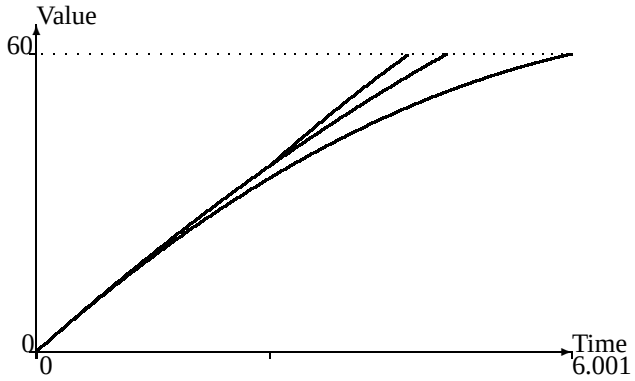

Figure 5: Graph of (volume bucket) for three separate plans.

Using the second tank to fill the bucket is even slower taking 6 time units to fill it to the brim. However if both tanks are used for the correct amount of time we can obtain an optimal plan.

```
0.001: (fill-bucket bucket tank1) [2.61]
2.612: (fill-bucket bucket tank2) [1.5]
```

Figure 5 shows the volume of the bucket of water for three plans, from left to right: filling the bucket using both tanks optimally, using tank 1 and using tank 2.

9 LaTeX Plan Validation Report

The LaTeX report option (`-l`) produces a report which is structured as follows:

Section	Topic
1	Domain and problem details
2	Validation Report for the first plan
2.1	Original Plan given by the user for validation
2.2	Plan to be validated by VAL
2.3	Blow by blow account of plan validation
2.4	Gantt chart
2.5	PNE graphs
3...	Validation reports of subsequent plans
n...	Outcome of validation attempts

The report allows better analysis of a plans validation, as well as providing a more formal record. Included is a Gantt chart which is useful to visualize a plans temporal structure, there are also graphs of the values taken by PNEs during a plans execution. The features of the report may also be useful for use in other documents, for example figures 2, 5, 7, 6, 8, 9 and 10 are taken from reports generated by VAL.

Report Styles

The style of the LaTeX document, (fonts, use of bold face and italics etc.) is given by a number of commands in the report preamble, for example:

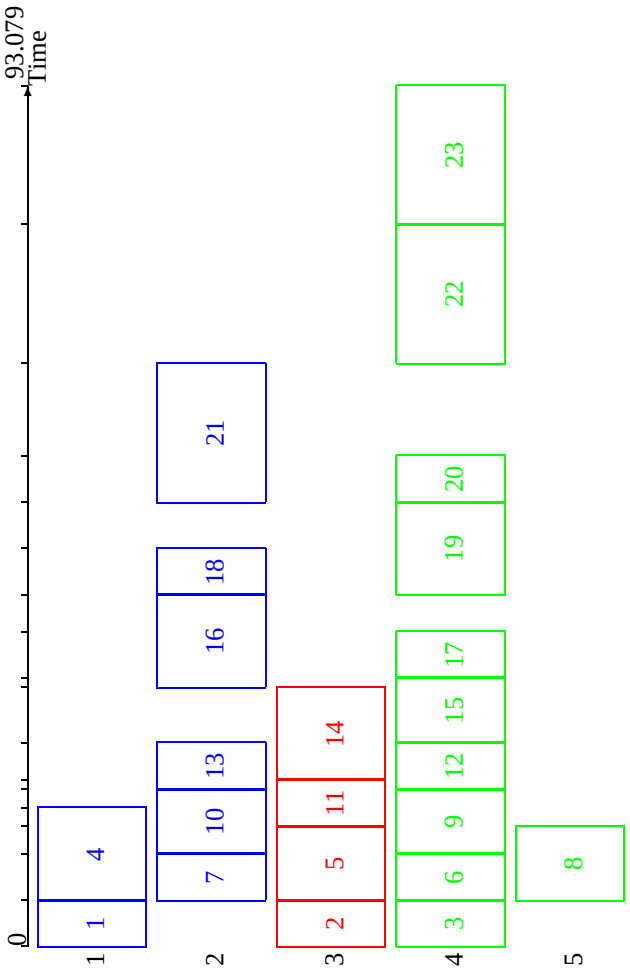

Figure 6: Gantt Chart

```
\newcommand{\action}[1]{{\sf #1}}
\newcommand{\exprn}[1]{{\sf #1}}
```

To change all the action names and expressions in the report, say to bold face, change the lines accordingly.

```
\newcommand{\action}[1]{{\bf #1}}
\newcommand{\exprn}[1]{{\bf #1}}
```

Gantt Chart

The LaTeX report includes a Gantt chart of the original plan given to VAL to validate, this shows the times over which actions are active highlighting duration, concurrent activity and dependency. The chart is drawn irrespective to whether the plan satisfies the final goal, is executable, or indeed makes any sense at all. Provided the plan can be parsed into actions from the plan file then the chart will be drawn. The chart consists of a number of rows against a time line axis, a durative action is shown as a bar with the start of the bar representing the start of the action and the end representing the end of the action. A non-durative action is represented as a line with a boxed number to label the action on the chart. The actions are sorted into rows by considering each action

Row 1 : rover2

No	Time	Action
1	0.001	(navigate rover2 waypoint7 waypoint0) [5]
4	5.006	(sample_soil rover2 rover2store waypoint0) [10]

Row 2 : rover2

No	Time	Action
7	5.006	(calibrate rover2 camera1 objective1 waypoint0) [5]
10	10.011	(take_image rover2 waypoint0 objective1 camera1 high_res) [7]
13	17.018	(navigate rover2 waypoint0 waypoint7) [5]
16	28.029	(communicate_soil_data rover2 general waypoint0 waypoint7 waypoint2) [10]
18	38.039	(navigate rover2 waypoint7 waypoint0) [5]
21	48.049	(communicate_image_data rover2 general objective1 high_res waypoint0 waypoint2) [15]

Row 3 : rover1

No	Time	Action
2	0.001	(navigate rover1 waypoint4 waypoint6) [5]
5	5.006	(sample_rock rover1 rover1store waypoint6) [8]
11	13.014	(navigate rover1 waypoint6 waypoint4) [5]
14	18.019	(communicate_rock_data rover1 general waypoint6 waypoint4 waypoint2) [10]

Row 4 : rover3

No	Time	Action
3	0.001	(navigate rover3 waypoint7 waypoint3) [5]
6	5.006	(calibrate rover3 camera3 objective3 waypoint3) [5]
9	10.011	(take_image rover3 waypoint3 objective2 camera3 low_res) [7]
12	17.018	(calibrate rover3 camera3 objective3 waypoint3) [5]
15	22.023	(take_image rover3 waypoint3 objective3 camera3 low_res) [7]
17	29.03	(navigate rover3 waypoint3 waypoint7) [5]
19	38.039	(communicate_rock_data rover3 general waypoint3 waypoint7 waypoint2) [10]
20	48.049	(navigate rover3 waypoint7 waypoint0) [5]
22	63.064	(communicate_image_data rover3 general objective3 low_res waypoint0 waypoint2) [15]
23	78.079	(communicate_image_data rover3 general objective2 low_res waypoint0 waypoint2) [15]

Row 5 : rover3

No	Time	Action
8	5.006	(sample_rock rover3 rover3store waypoint3) [8]

Figure 7: Gantt Chart Key

in turn, as given in the original PDDL2.1 plan given to VAL, using the following rules:

1. An action is placed into the row where the last action terminated most recently. (If there is more than one such row it is placed into the first of these rows.)
2. If each row has an active action at the start time of the

action to be assigned a row then the action is placed in a new row. (The first action to be considered is obviously placed in a new row.)

A domain may contain a set of objects that represent executives, it may then be desirable to highlight those actions that affect these executives. It is possible to group actions into rows for each executive, which may have one or many rows depending on concurrent activity. There may also be other motives to group the actions with the same object as a parameter. The extra following rule is then followed if actions with the same special object are to be grouped together:

3. If an action has a special object then it can only be placed in a row where the actions in this row also have this object as a parameter.

The list of special objects can specified using the option '`-o obj1 obj2... -o`', where the list consists of object names or types of objects. Figures 7 and 6 show an example of a plan's Gantt chart using VAL, given from the 2002 planning competition using the simple time rover domain, problem number 12 and the solution given by LPG (Gerevini & Serina 2002). Rows 1 and 2 (coloured blue) show the actions for `rover2`, row 3 the actions for `rover1` (coloured red) and rows 4 and 5 the actions for `rover3` (coloured green).

If the Gantt chart appears to be too 'cramped' VAL will split the chart across the time axis by a number of pages. Also if there are too many rows the chart will be split across the rows by a number of pages. This page splitting can be overridden to give the desired appearance using the option '`-p <n> <m>`', where `n` is the number of pages to split the time axis over and `m` is the number of pages to split the rows across. To obtain the default value for the number of pages for either parameter set the value to zero.

Primitive Numerical Expressions

The LaTeX report contains graphs showing the values taken by PNEs over the duration of a plan. These graphs show discrete changes in value (see figure 10), linear changes in value (see figure 9), and non-linear changes in value (see figure 8). The interaction between PNEs may also be clearly observed as shown by the three graphs showing the acceleration (see figure 10), speed (see figure 9) and distance (see figure 8) of a car. These graphs were taken from the output of VAL for a simple domain modelling the behaviour of driving a car.

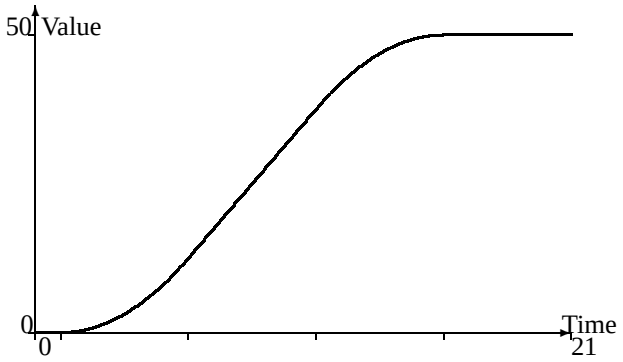

Figure 8: Graph of (distance car).

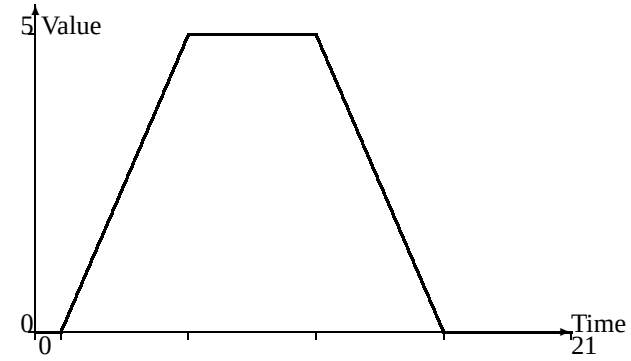

Figure 9: Graph of (speed car).

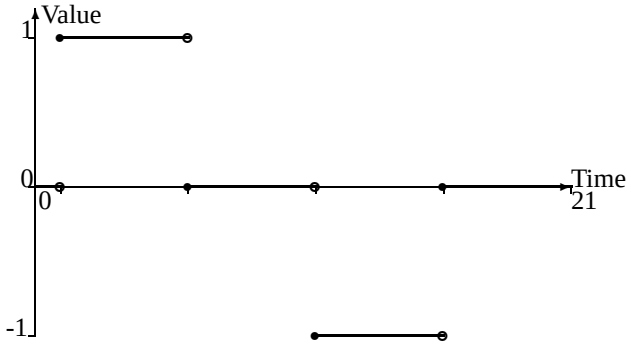

Figure 10: Graph of (acceln car).

10 Overview Of Usage

The code for VAL is written in C++ using the STL. It is known to compile under Linux with Gnu C++ (2.95.3 and 2.96) and the associated STL. We have also built successfully on Sun machines using a Gnu installation (2.95.2). The parser requires flex++ and bison (also Gnu tools). VAL consists of one executable, `validate`, which expects three or more filenames: a domain, a problem and one or more plans. There is a growing number options available:

`-t <n>`

Set the tolerance to the (float) value of `n`, the default value is 0.01.

`-v`

Verbose reporting of plan check progress.

`-h`

Display the help message.

`-g`

Use the graphplan length where no metric is specified.

`-l`

Output a LaTeX report of the plan validation. This includes a Gantt chart and graphs of the values taken by PNEs.

`-p <n> <m>`

This option allows the Gantt chart to be split across a number of pages. The first value `n` is the number of pages to split the time axis over, the second value `m` is the number of pages to split the rows over. Zero obtains the default value.

`-o ... -o`

Specify a list of the objects and/or types of objects to be tracked on the Gantt chart.

`-c`

Continue the plan validation to next happening if the current happening fails. The plan will still, of course, be deemed to have failed. Note that a failed happening may not have all of its effects enacted.

`-i`

An invariant may include comparisons with continuously changing PNEs in it that are too complex to verify. This option allows the validator to assume that the conditions hold and continues to validate the plan. The plan is then valid subject to these conditions holding – which may be trivial to prove (or disprove) to a human. For example consider the invariant

$$t^{100000000000} - t^{1000000000} + 2 > 0 \qquad \text{for } t \in (0, 5),$$

it is not too difficult to prove that this condition holds. However automatic validation of this invariant using a root finding approach may be somewhat problematic. In the future when VAL can handle more complex functions it will be possible to express quite complicated invariants that may however be simple to verify by a human working with these functions. For example the following invariant may arise

$$\frac{e^{-2t} - e^{t}}{t^2 + 2t + 3} + 1 > 0 \qquad \text{for } t \in (0, 2)$$

which could be straightforward to verify to a human, but for VAL to be able to check *any* such non-linear expression that may arise is another matter. The PNE graphs may be a useful indicator to whether an invariant holds or not, but of course these by no means constitute a proof that the invariant holds or not.

As an example of using VAL to validate a plan in the rover domain, and produce a LaTeX report of the results with the rover objects tracked in the Gantt chart, the following could be executed:

```
validate -l -o rover -o STRover.pddl
 pfile12 pfile12.soln
```

11 Conclusion

This document describes the progress of extending VAL used in the 3rd International Planning Competition to validate plans with continuous effects, and the related problems in achieving this. Currently, to guarantee the validation of a plan containing durative actions with continuous effects the restriction that all continuous effects are polynomial must be met. This condition implies that the dependency graph of the rates of change of PNEs has no loops. This condition is automatically checked and VAL identifies plans that it cannot correctly validate.

The validation of plans containing continuous effects is an important first step in making planners capable of planning with languages that express them. The availability of a validation tool is a vital first step for the community in progressing along this path. VAL is available at the competition web-site which is given in the references.

References

Edelkamp, S., and Helmert, M. 2000. On the implementation of mips. In *Proceedings of Workshop on Decision-Theoretic Planning, Artificial Intelligence Planning and Scheduling (AIPS)*, 18–25. AAAI-Press.

Fox, M., and Long, D. 2002. PDDL2.1: An extension to PDDL for expressing temporal planning domains. Technical Report Department of Computer Science, 20/02, Durham University, UK. Available at `www.dur.ac.uk/d.p.long/competition.html`.

Fox, M.; Long, D.; and Committee. 2002. International planning competition. Web-site at `www.dur.ac.uk/d.p.long/competition.html`.

Gerevini, A., and Serina, I. 2002. LPG: A planner based on local search for planning graphs. In *Proc. of 6th International Conference on AI Planning Systems (AIPS'02)*. AAAI Press.

Howey, R., and Long, D. 2003. A formal semantics for PDDL2.1 with continuous change. Technical report, Durham University, UK. Available at `www.dur.ac.uk/computer.science/research/stanstuff/html/dpgpublications.html`.

Johnson, J. R., and Krandick, W. 1997. Polynomial real root isolation using approximate arithmetic. In *International Symposium on Symbolic and Algebraic Computation*, 225–232.

Rouillier, F., and Zimmermann, P. 2001. Efficient isolation of a polynomial real roots. Technical Report 4113, Rinria.