Hybrid Plan Validation using VAL

Maria Fox[1] Derek Long[1] Richard Howey[2]

[1]King's College London
[2]Newcastle University

Workshop on Model Checking and Automated Planning,
ICAPS 2014

Outline

VAL for PDDL2.1

- Developed in 2002 for validating plans and PDDL2.1 domains
- Very widely used for domain, problem and plan file validation in the PDDL-family planning community since 2002
- Extended in 2004-5 to validate hybrid plans and domains expressed in PDDL+ (process and event models)
- VAL consists of the exact validation methods described in this talk, *and* a stochastic robustness checking method which is not described here (see Fox, Howey and Long, AAAI 2005)

By *Hybrid Planning* we mean that:

- Discrete state changes occur on the continuous time line
- Actions and events cause continuously changing numeric quantities to be updated at arbitrary points on the time line
- Interaction of continuous effects give rise to systems of Differential Equations
- Invariants of the form of comparisons involving continuously changing numeric quantities give rise to functions that must be analysed on a given interval

Validating a hybrid plan breaks down into two main tasks:

- Solving systems of Differential Equations produced by interacting continuous effects
- Checking invariants over intervals, which is equivalent to showing that a function is always non-negative (has no roots) in that interval.

Syntax of continuous effects in PDDL2.1

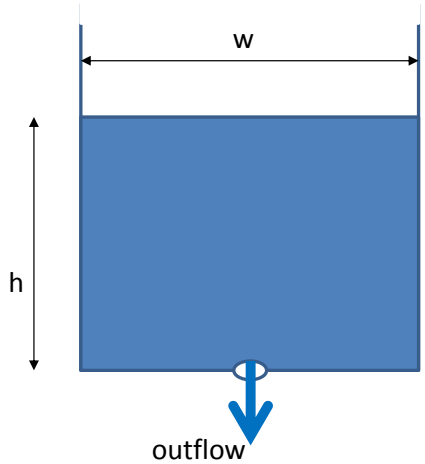
An example of a timed effect is:

```
increase (fuelVolume ?v) (* #t (refuelRate ?p))
```

which models the differential equation:

$$\frac{d}{dt}(\mathit{fuelVolume}\,?v) = (\mathit{refuelRate}\,?p)$$

where `(refuelRate ?p)` is either a constant or another time-dependent function.



Flow rate from the tank depends on the volume of liquid in the tank, and volume depends on the flow rate

Figure: A problem with nonlinear dynamics

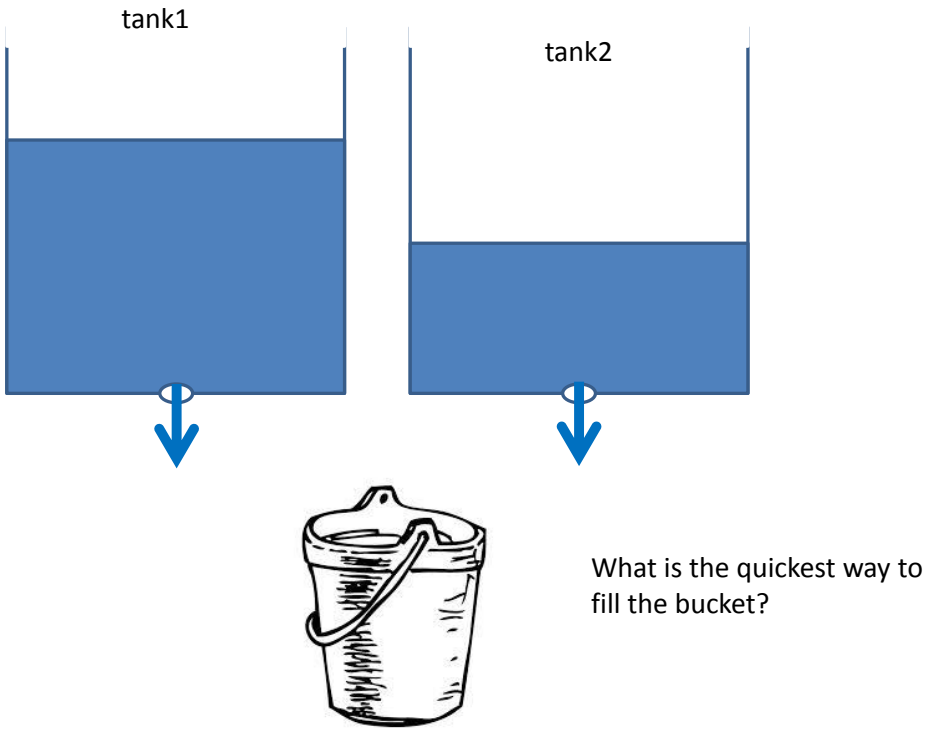


Figure: An extended version

We can model much harder problems than we can solve with planners

- PDDL2.1 allows the two tanks problem to be expressed using a single durative action with an undetermined duration `(> ?duration 0)`
- A correct model of the problem can be constructed, using Torricelli's formula to describe the process of draining a tank
- The problem is to drain each tank for periods determined by the planner, until the bucket is full, taking the minimum time overall
- The solution is a two-step plan that handles the non-linear continuous behaviours
- This problem can be solved by UPMurphi [della Penna, Mercorio and Magazzeni, 2009]

We can validate much more expressive plans than we can generate

- Given a plan and a PDDL model that satisfies certain restrictions on the types of functions expressed, it is possible to determine whether the plan is valid with respect to the model in time polynomial in the input size
- The restrictions are that the differential equations are of the form:

$$\frac{df}{dt} = poly(t)$$

or

$$\frac{df}{dt} = Kf$$

for which the functions obtained are polynomials or simple exponentials.

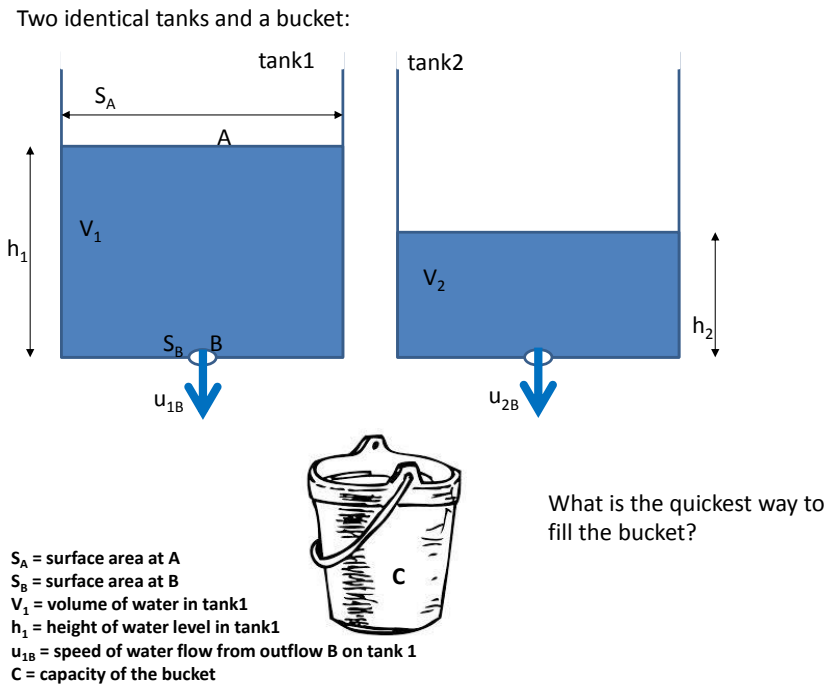


Figure: The problem of two tanks and a bucket

The single tank case

Torricelli: *Water in an open tank will flow out through a small hole in the bottom with the velocity it would acquire by falling freely from the water level to the hole.*
The speed at which water drains from the hole in a tank is therefore:

$$u_B = \sqrt{2gh}$$

where h is the height of the water level.
If S_B is the area of the hole and V is the volume of water in the tank, we have:

$$\frac{dV}{dt} = -S_B\sqrt{2gh}$$

In a straight tank, where S_A is the cross sectional area, $V = S_A h$ so:

$$\frac{dV}{dt} = -S_B\sqrt{\frac{2gV}{S_A}} = -k'\sqrt{V}$$

and $\int \frac{1}{\sqrt{V}} dV = \int -k' dt$.

Then: $V = (-kt + \sqrt{U})^2$, where $k = \frac{k'}{2}$ and U is the volume of water in the tank in the initial state, and:

$$\frac{dV}{dt} = 2k(kt - \sqrt{U})$$

k is called the *flow-constant* of the tank.

The tank model in PDDL2.1

```
(:durative-action fill-bucket
:parameters (?b ?t)
:duration (>= ?duration 0)
: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 amount drained from a tank in time t is

$$U - (-kt + \sqrt{U})^2 = 2kt\sqrt{U} - k^2t^2$$

and for this to equal the capacity of the bucket, D, we would need that

$$k^2t^2 - 2kt\sqrt{U} + D = 0$$

This is therefore the function whose roots we will be interested in when checking the invariant of the `fill-bucket` action.

Initial State

```
(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 goal is to have the bucket filled as quickly as possible.
- We have a bucket of capacity 60. We must now decide how long to drain water from each tank.
- The average rate of flow must be maximised, which is achieved by making the rates of flow from each tank equal after some time draining from each.
- The order in which we drain from the tanks does not matter as we only care about the total time being minimised.

The optimal plan to fill the bucket would then look like:

```
0.001:   (fill-bucket bucket tank1) [d1]
0.001+ d1 + ϵ:   (fill-bucket bucket tank2) [d2]
```

where `d1` and `d2` are the times for which the bucket should be filled from each tank in order to minimise total filling time.

These values can be obtained by finding the time at which the average rate of flow is maximised, which would be the task of the planner.

- VAL's task is to validate a plan against a given model, expressed in PDDL2.1 or PDDL+.
- VAL doesn't care where the plan came from or how it was obtained

We will consider VAL's behaviour on validating the following plan, found by UPMurphi, for the two tanks problem:

```
0.001:   (fill-bucket bucket tank1) [2.6]
2.602:   (fill-bucket bucket tank2) [1.5]
```

- What does VAL do?
- It recognises the forms of DEs in the effects of the actions and processes, looks up the general solutions to the managed forms, then uses the initial conditions to find the particular solution functions.
- Why do we have to code up the DEs so that VALcan decode them?
- That is just a reflection of where we are with the modelling languages we use (suppose a richer language where we could just specify that Torricelli's formula applies).

The simple plan

- VAL interprets a temporal plan using the PDDL2.1 semantics
- The timeline is punctuated by Happenings at which change occurs and continuous effects can interact with invariants
- A durative action, a, is decomposed into two *Regular Happenings*, a_{start} and a_{end}, an *Invariant Happening* and a *Continuous Update Happening*
- These are arranged on the timeline, with the invariant check and updates immediately prior to a_{end}.
- If another Happening intervenes, the invariants of a must be checked and continuous updates performed at that point, splitting the durative interval of a

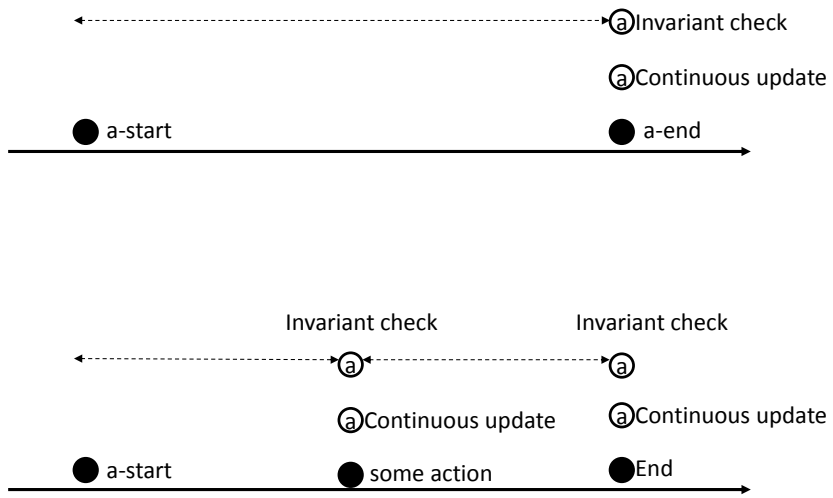


Figure: Checking invariants and updating continuous variables is done at the end of the durative interval of the action *a*..

Checking Invariants

- When invariants are comparisons involving numeric expressions, checking that they hold over an interval is equivalent to showing that a function is always non-negative in that interval
- VAL implements an efficient and infallible polynomial root finder, based on Descartes rule of signs and the Newton-Raphson method
- For non-polynomial functions other methods are used (polynomial approximation or direct solution)

Validating a plan

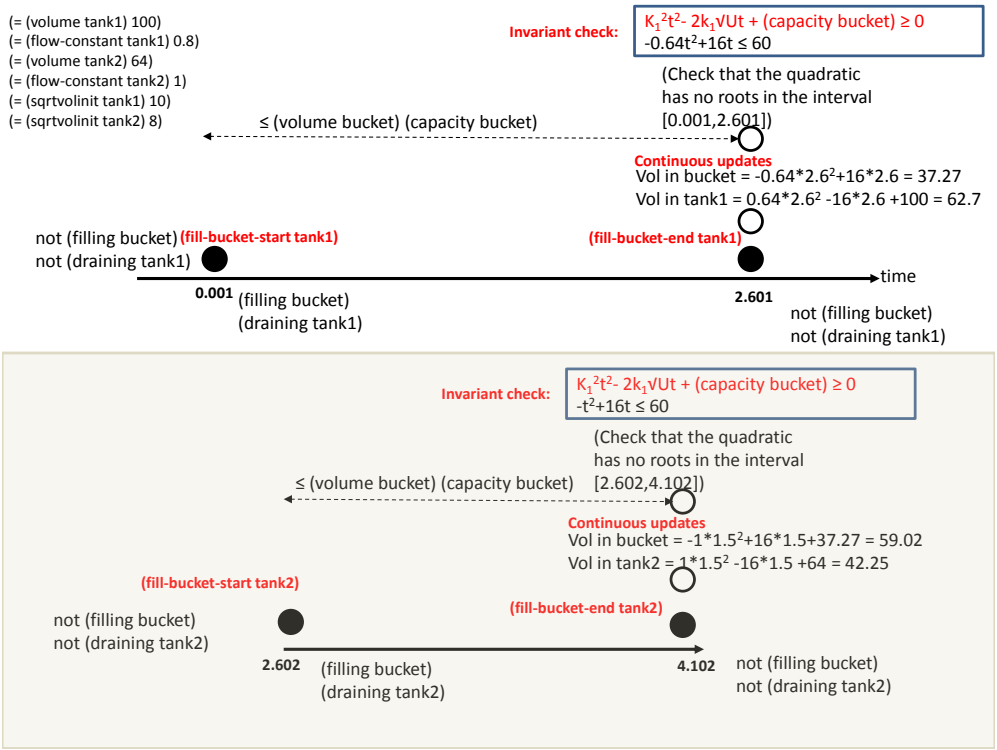


Figure: Validating the 2-step plan

The VAL report

Time	Details
2.601:	Checking Happening... ...OK! **(volume bucket)**$(t) = -0.64t^2 + 16t$ **(volume tank1)**$(t) = 0.64t^2 - 16t + 100$ **(sqrtvol tank1)**$(t) = -0.8t + 10$ **(drain-time tank1)**$(t) = t$ Updating **(volume bucket)** (0) by 37.2736 for continuous update. Updating **(volume tank1)** (100) by 62.7264 for continuous update. Updating **(sqrtvol tank1)** (10) by 7.92 for continuous update. Updating **(drain-time tank1)** (0) by 2.6 for continuous update.
2.601:	Checking Happening... ...OK! Deleting (draining tank1) Deleting (filling bucket) Updating **(sqrtvolinit tank1)** (10) by 7.92 assignment.
2.602:	Checking Happening... ...OK! Adding (draining tank2) Adding (filling bucket) Updating **(drain-time tank2)** (0) by 0 assignment. Updating **(sqrtvol tank2)** (0) by 8 assignment.
4.102:	Checking Happening... ...OK!

- What happens if a tank runs dry?
- If we make the capacity of the bucket very large, and have very long actions, the tanks would run dry and VAL must detect this.

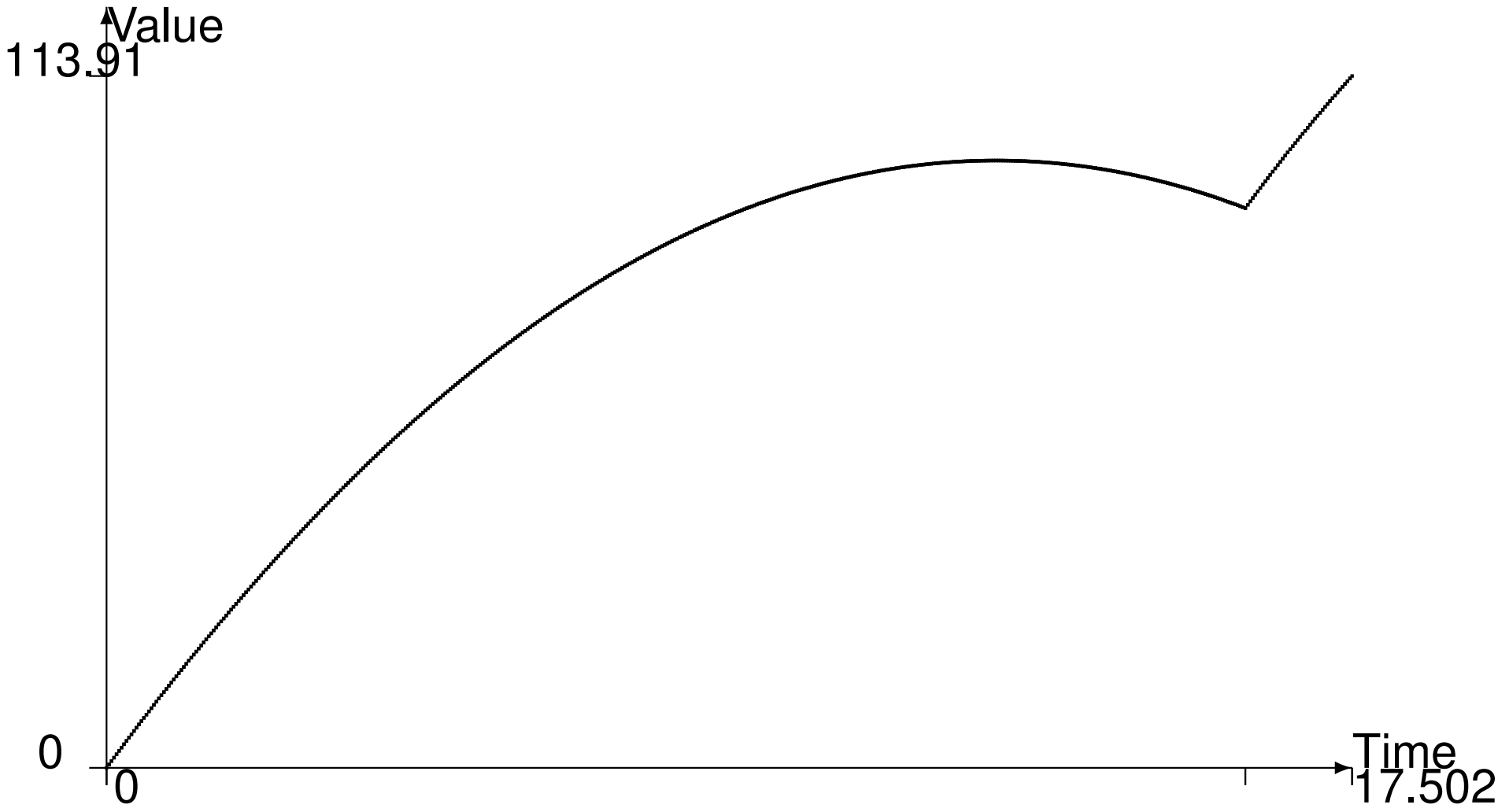


Figure: Graph of (volume bucket).

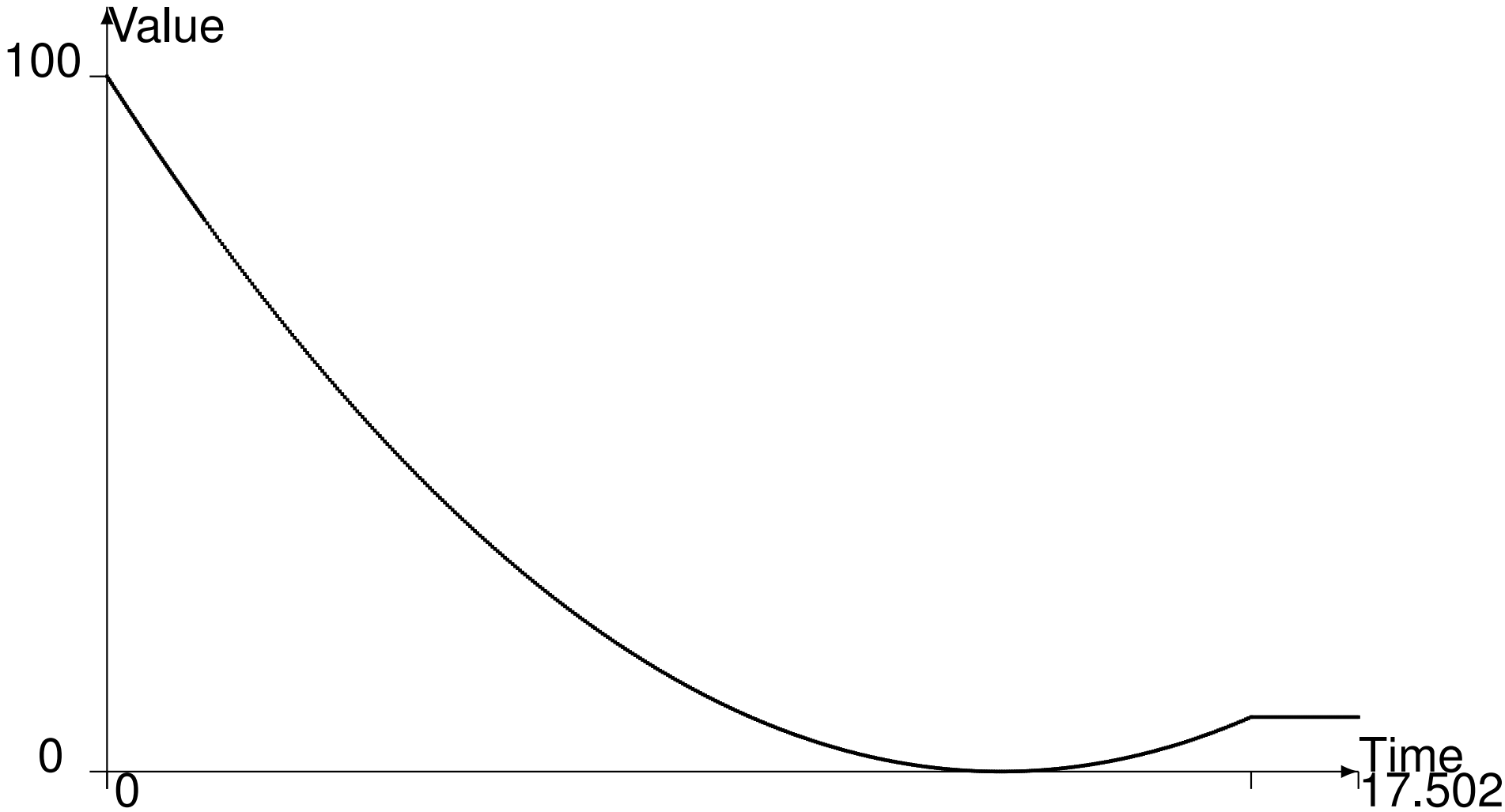


Figure: Graph of (volume tank1).

- We derive Torricelli's formula from Bernoulli's equation:

$$\frac{P_A}{\rho} + \frac{u_A^2}{2} + gz_A = \frac{P_B}{\rho} + \frac{u_B^2}{2} + gz_B$$

  where A is the water level and B is the height of the hole (zero in our case).
- If the tank is being replenished at the draining rate then the surface velocity u_A is zero.
- In our case there is no replenishing, but if the area of the hole is very small relative to the surface area of the tank, then Torricelli's formula produces a good abstract model.

Validating Plans with Events (PDDL+)

```
(:event tank-drained
:parameters (?t)
:precondition (and (<= (volume ?t) 0.001)
                                (draining ?t))
:effect (and (not (draining ?t))
                    (assign (flow-constant ?t) 0))
)
```

Time **Details**

12.4615: Checking Happening... ...OK!
(volume tank1)$(t) = 0.64t^2 - 16t + 100$
(sqrtvol tank1)$(t) = -0.8t + 10$
(drain-time tank1)$(t) = t$
(volume bucket)$(t) = -0.64t^2 + 16t$
Updating **(volume tank1)** (100) by 0.001 for continuous update.
Updating **(sqrtvol tank1)** (10) by 0.0316228 for continuous update.
Updating **(drain-time tank1)** (0) by 12.4605 for continuous update.
Updating **(volume bucket)** (0) by 99.999 for continuous update.

12.4615: **Event triggered!**
Triggered event (tank-drained tank1)
Deleting (draining tank1)
Updating **(flow-constant tank1)** (0.8) by 0 assignment.

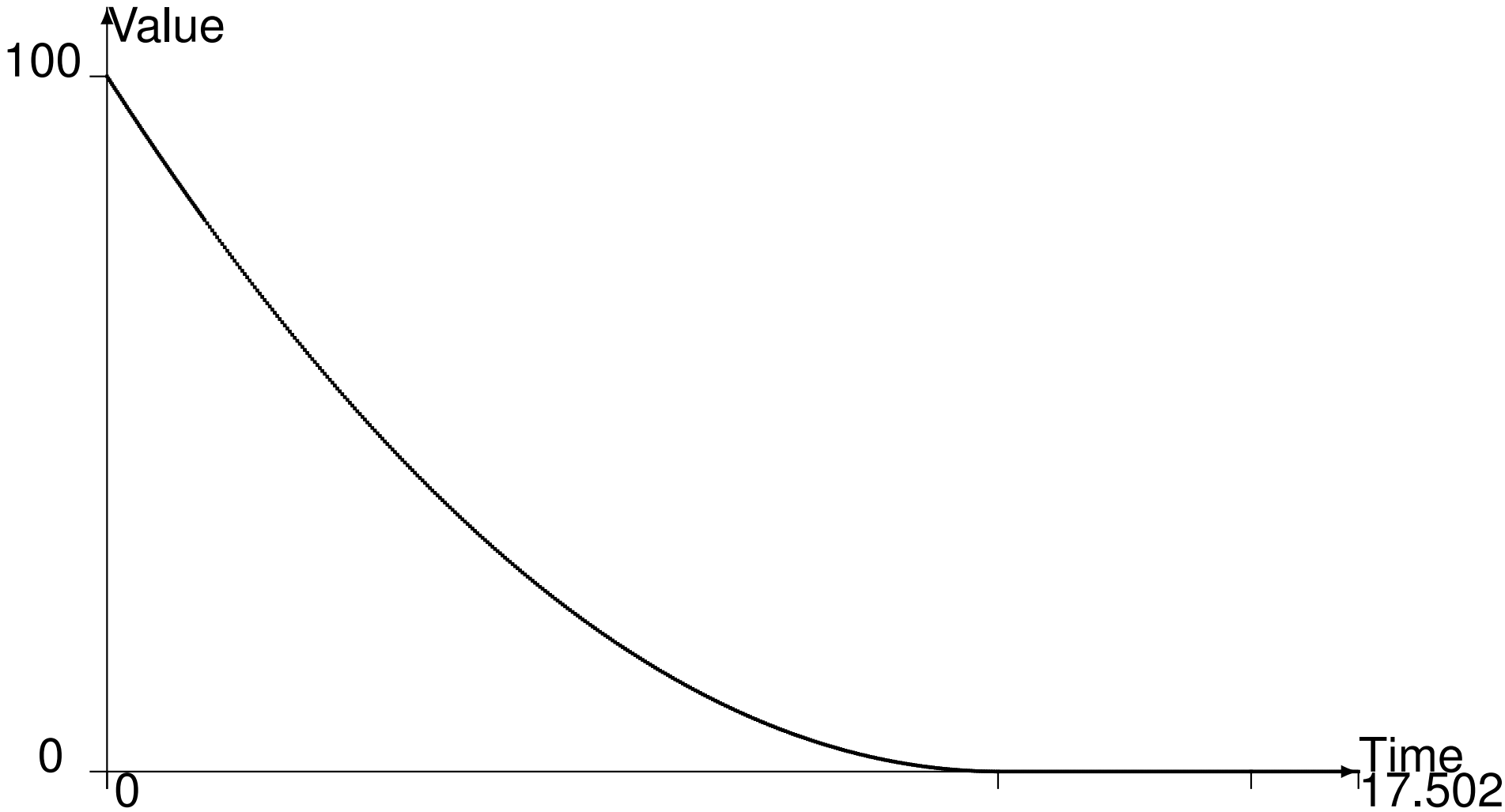


Figure: Graph of (volume tank1).

Summary

- VAL is a well-tested and maintained plan-validation tool for PDDL2.1 and PDDL+ domains
- Plans containing a variety of complex non-linear dynamics can be validated
- VAL copes with a wide range of functions, as long as they are polynomials or simple exponentials of the form $\frac{df}{dt} = f$.
- Outlook
  - We are extending VAL to cope with even more complex functions
  - We are investigating ways to integrate VAL into the plan generation process