VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning using PDDL

Richard Howey, Derek Long and Maria Fox
Department of Computer and Information Systems, University of Strathclyde, Glasgow, UK
firstname.lastname@cis.strath.ac.uk

Abstract

This paper describes aspects of our plan validation tool, VAL. *The tool was initially developed to support the 3rd International Planning Competition, but has subsequently been extended in order to exploit its capabilities in plan validation and development. In particular, the tool has been extended to include advanced features of* PDDL2.1 *which have proved important in mixed-initiative planning in a space operations project. Amongst these features, treatment of continuous effects is the most significant, with important effects on the semantic interpretation of plans. The tool has also been extended to keep abreast of developments in* PDDL, *providing critical support to participants and organisers of the 4th IPC.*

1. Introduction

This paper examines the development of VAL[1], the plan validation tool for PDDL. The tool played an important role in the 3rd International Planning Competition [9], allowing reliable and automatic validation of the several thousand plans produced by the competitors. It also provided competitors with a basis for checking their planners as part of their own development and debugging cycles and an understanding of the semantics of PDDL as described in Fox and Long [5]. We have found that the capabilities of VAL have been critical in understanding the structures of large plans, with its visualisation and reporting facilities. This role of VAL has continued into the 4th IPC, which has included several minor extensions to PDDL and its semantics and consequently to VAL.

The original definition of PDDL2.1 used in the 3rd IPC included features not used, in particular the expression of continuous change. Planning has traditionally been a subject of discrete change; a sequence of well defined discrete changes to a world state model (with a minority of exceptions). Continuous change to numerical values has often been been modelled using discrete changes describing a step function. However this is inadequate in modelling many real world situations with continuously changing values that interact with one another and where (sufficiently) accurate access to the values must be available at all times. For example, consider the battery power model in figure 5 which is given by a non-trivial system of differential equations, where at any given point we must ensure that the battery charge does not cross a critical minimal threshold. The interactions of continuously changing values are impossible to model as a step function and moreover checking the battery has not ran out requires accessing the power level at any given point due to its complex non-linear structure. In VAL we have incorporated the validation of plans with continuous effects, which includes: (i) the development of the semantics of continuous effects in PDDL (section 3.2), and (ii) the analysis of differential equations and continuous functions with respect to the semantics (section 4).

In many real world examples the (initial) planning process can only be carried out by humans. This could be due to the tasks containing complex functions (continuous effects) that are not handled by current planning systems or complex goals that cannot be expressed in any planning language. These goals could be technical, political, ambiguous, prioritized, secretive or changeable — so called *soft goals*. In addition, in certain applications there is resistance to wholesale replacement of human operators with autonomous systems and it is important to build trust through the initial deployment of mixed-initiative systems — systems that support human-machine interaction in the process of plan construction [4, 12, 2]. However, if the essential ingredients can be modelled in a planning language then we can at least validate the plan (using VAL) to see if the plan is executable. VAL can report if the plan is flawed and then the human planners can try to fix their plan. To support this process, VAL has been further developed: if the plan is flawed VAL will give advice on how the plan should be fixed. The human planner can then use this advice to produce a new plan and try again, completing a mixed-initiative planning

[1] Available at http://planning.cis.strath.ac.uk/VAL/

cycle. We are developing the plan advice and are aiming towards a complete plan repair strategy (for certain classes of invalid plans).

In section 2 we review the semantics of the planning language PDDL2.1 used in the IPC series and managed by VAL. An important extension to PDDL is the use of durative actions with continuous effects, discussed in section 3, which includes the semantics of continuous effects within PDDL2.1. The introduction of continuous effects into PDDL not only creates problems in how they are to be integrated semantically, but also the mathematical analysis problems of real valued functions involved in the validation of plans in VAL. These problems are reviewed in section 4. VAL is written to be a useful tool in the planning community (researchers, developers and users), so we discuss the LaTeX report generation facility in section 5. Section 6 describes the plan repair advice generated by VAL for invalid plans. In this facility VAL goes beyond validating a plan and suggests ways to repair an invalid plan. This facility plays an important role in a collaborative project undertaken by the authors in space operations planning [13, 7]. The project is currently concerned with mixed-initiative mode planning, but it is intended to extend this work to include on-board autonomy. In mixed-initiative mode, VAL is used to validate and help to repair invalid plans.

2. A Brief Review of PDDL and its Semantics

Since PDDL was first proposed as a community standard in 1998 the planning research community has seen incremental extensions and modifications as the language has adapted to various goals. The core of PDDL was a STRIPS language, offering an ADL extension. This language has a semantics that is widely accepted, based on a simple state-transition model. This semantics has few areas of potential ambiguity. Perhaps the most significant issue for which alternative resolutions exist is that of concurrency: classical plans are often considered to be *sequences* of steps, representing state transitions, but partial-order planning [11] and Graphplan [3] both offer alternative models in which some form of parallelism is considered. McDermott developed a simple plan validation tool for PDDL that accepted only sequential plans. However, the question of interpretation for more complex extensions of PDDL is more difficult. There is no prior widely accepted model, so choices must be made that are not necessarily universally accepted. Since the language plays a central role in communication of domains between researchers, it is important that there be a standard by which a common understanding may be developed for the semantics of domains and plans for those domains. A formal semantics is the first component of this. However, a formal semantics is not sufficient by itself, because a formal semantics is notoriously difficult to read. In practice, many formal semantics are read in detail by few and understood in all details by even fewer. To make the semantics accessible, their implementation as a validation tool is an important step. In this form, it is possible to confirm understanding of the semantics by testing various plans and domains with the tool, confirming the behaviour is as expected. VAL supplies a variety of forms of feedback, making it possible to explore quite precisely what might be wrong with a flawed plan and aiding in the interpretation of the more subtle details of the semantics. Importantly, VAL can be used to ensure consistent semantics between different planners - even if those planners have been proven sound, since soundness proofs are relative to the authors' formulation of the semantics.

The introduction of timed actions instead of a sequence of actions is a straightforward extension, this is achieved by an embedding of the activity into a real time line (see [5]). However, this introduces the problem of explaining under what circumstances the end points of actions (when instantaneous change occurs) may coincide. This is resolved by ensuring that for coinciding actions the preconditions and effects of one action do not *interfere* with those of another action. See [5] on *mutex actions* for details.

Discretized durative actions. When *discretized* durative actions are used, the modeller specifies the *local* pre- and post-conditions of the end-points of the interval, as well as (optionally) invariant conditions that must hold throughout the interval. This makes it possible to exclude as invalid precisely those plans that violate the necessary conditions for successful completion of the durative action without allowing the modeller the expressive power required to model complex temporal interactions.

It is straightforward to give formal semantics to discretized durative actions. If no invariant is specified then a plan containing durative actions can be transformed into one containing just instantaneous actions, one for each of the end-points of every durative action in the original plan. When invariants are specified it is necessary to confirm that the invariant remains true after every action that occurs between the start and end-points of the durative action specifying that invariant.

Continuous durative actions. Durative actions in PDDL2.1 may also be defined so that the modeller can access arbitrary time points within the interval of duration using a variable, **#t**, that refers to these points thus defining a continuous function on this interval. This is achieved by specifying the rate of change of a numerical value, for example: **increase (distance ?c) (* #t (speed ?c))**.

3. Continuous Effects

A continuous effect can only affect metric quantities: it is not possible to change a propositional fluent continuously.

A metric variable that can be changed by a continuous effect is called a *Primitive Numerical Expression (PNE)*. A durative action that has a continuous effect on a PNE changes it so that the values taken 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 it. 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 PNE itself, where an action assigns directly to the PNE, so that the value describes a discontinuous behaviour, or can be discrete changes in the rate of change so that the value describes a piece-wise continuous, but non-differentiable behaviour. The latter case occurs when an action modifies (instantaneously) the derivative of a PNE.

3.1. 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 which describe the motion of a car driving.

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

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.

3.2. Implementation: Semi-Simple Plans

The semantics of classical actions in terms of state transitions is familiar. Following [5], we call these actions *simple actions* and plans constructed only from simple actions we call *simple plans*. For lack of space we briefly summarise definitions given in full in [5]: a *simple plan* is a collection of pairs (t, a), where t is a time and a is an action name. Each distinct time in a simple plan defines a *happening* at which point a set of simple actions in the simple plan occurs. A *plan* extends a simple plan to include durative action instances, each with an associated duration.

Durative actions with discrete effects can be given a semantics in terms of the semantics of simple plans. This is shown by mapping plans containing durative actions to simple plans (details can be found in [5]), in which the end points of the durative action are treated as simple actions in a simple plan. Invariants of durative actions can also be treated as simple actions with preconditions but no effects. These appear at points in a simple plan corresponding to the critical times at which the invariants must be checked during the interval of the corresponding durative action. It is not possible to give the semantics of durative actions with continuous effects in terms of a simple plan, because the values of PNEs may be required at arbitrary points over an interval on which they are continuously changing.

We therefore define an extension of a simple plan, a *semi-simple plan*.

Definition 3.1 Act *An* act *is a happening labelled with an* act type. *The act type can be one of three values:* invariant, continuous update or regular.

Definition 3.2 Semi-Simple Plan *A* semi-simple plan, *SSP consists of a finite collection of* timed simple actions *which are 3-tuples (t, A, a), where t is a rational-valued time, A is the act type and a is an action name.*

Definition 3.3 Act Sequence for a Semi-Simple Plan *The* act sequence *for a semi-simple plan SSP, $\{(t, A)_i\}_{i=0\ldots k}$, is the lexicographically ordered (by time, then act type), sequence of time and act type pairs appearing in the timed simple actions in SSP. The act types are ordered invariant, continuous update, regular. For all i, $t_i > 0$.*

The definition of ground durative actions [5] is extended to include continuous effects by introducing a simple action to abstract out the continuous effects. A plan containing actions with continuous effects can be mapped to a semi-simple plan in a straightforward way: end points of a durative action are mapped to regular acts, invariant checks are mapped to invariant acts and continuous effects are mapped to continuous update acts. 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 continuously changing PNEs discretely at the end of the interval. Each invariant check is responsible for confirming correctness over the preceding interval of continuous change. In (2) we show that if a simple action occurs between the start

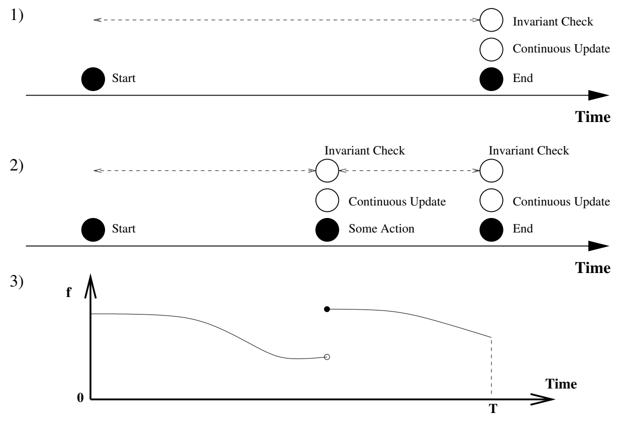

Figure 1. Durative action with continuous effects.

and end points of a durative action then a continuous update and invariant act for that durative action is placed before this simple action. This mapping of P is called the *induced semi-simple plan*, written *semi-simplify(P)*. Part (3) shows how discrete effects can arise, due to parallel activity, breaking the continuous change into piece-wise continuous components.

To execute a regular act, we apply the state transition corresponding to all of its simple actions using the familiar add and delete effect semantics, together with numeric updates in the obvious way. To execute a continuous update act is also straightforward: the continuously changing PNEs are updated according to the functions of time describing their behaviour on the interval from the preceding regular act. Invariant acts are not straightforward and checking invariants is considered in section 4.1.

Definition 3.4 Executability of a Semi-Simple Plan *A semi-simple plan, SSP, is executable if it defines an act sequence, $\{(t, A)_i\}_{i=0...k}$ with states, $\{S_i\}_{i=0...k+1}$. S_0 is the initial state and for each $i = 0 \ldots k$, S_{i+1} is the result of executing the act, Act_i, for $(t, A)_i$:*

- *If A_i is regular then the preconditions of Act_i must hold in S_i and S_{i+1} is the result of removing delete effects, adding add effects and applying numeric effects.*
- *If A_i is invariant then the conditions of Act_i must hold over the interval between the preceding regular act and t_i (taking into account any continuous change).*
- *If A_i is continuous update then the effects of Act_i are applied at time t_i for the continuous effects over the interval between the preceding regular act and t_i.*

The state S_{k+1} is called the final state *produced by SSP and the state sequence $\{S_i\}_{i=0...k+1}$ is called the* trace *of SSP. Note that an executable plan produces a unique trace.*

Definition 3.5 Validity of a Semi-Simple Plan and of a Plan *A semi-simple plan is* valid *if it is executable and produces a final state S, such that the goal specification is satisfied in S. A plan, P is valid if semi-simplify(P) is valid.*

4. Plan Validation Challenges from Semantics

4.1. Invariants

Continuous effects have their most significant effect on the validation of plans when they interact with 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 greater than or equal to zero). For example

$$t^4 - 3t + 1 > 0 \qquad \text{for } t \in (0, 3)$$

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, when checking an invariant condition with a non-linear expression in time it is no longer sufficient to check the condition at end points only. *The key to the problem of checking invariants that are comparisons with non-linear expressions in time is 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 restrictions on the invariants that may be expressed to guarantee that they can be validated on a given interval. 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.

Invariants with disjunctions provide an extra complication when the disjuncts depend on continuously changing PNEs. For example consider the following invariant with a disjunction

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

for t in $(0, 10)$. We must find the values of t in $(0, 10)$ for which each disjunct is satisfied, then take their union and see if the result covers $(0, 10)$. In general it is necessary to find all the roots of all the continuous functions involved: these points can be used as the end points of the sub-intervals that each disjunct is satisfied on.

4.2. Differential Equations

The complexity of the differential equations that can be expressed far exceeds the practicality and feasibility of solving them. It is therefore necessary to impose certain restrictions to guarantee that they can be solved. The following proposition shows that the values taken by PNEs are given by polynomials if certain restrictions are imposed.

Proposition 4.1 *Let $F = \{f_1, \ldots, f_n\}$ be a finite set of PNEs changing continuously on the interval $[0, T]$ given by*

$$\frac{df_i}{dt} = g_i(f_1, f_2, \ldots, f_n) \text{ for all } i \in \{1, 2, \ldots, n\}$$

where g_i is some function depending on F. The function g_i is restricted to addition, subtraction, multiplication and division on its terms and division by a PNE in F is not permitted. If the rate of change of no PNE depends on itself (either directly or indirectly) then the value of every PNE on $[0, T]$ is given by a polynomial in t.

Proof. Follows by induction on the dependency structure.

If the conditions in Proposition 4.1 were relaxed to allow division by a functional expression then a PNE could take values given by a natural logarithm. If the dependencies could contain loops then exponential functions could occur, as well as trigonometric functions and so on. So far VAL handles polynomials and certain classes of exponential functions involved in the space operations project we are involved with, see section 6. We are also currently investigating solving certain classes of differential equations numerically. We are using the Runge-Kutta-Fehlberg [10] method and have had very encouraging results.

4.3. Summary of Challenges from Semantics

There are two main challenges in the implementation of validating a plan with continuous effects:

1. Solving a system of differential equations.
2. Finding the roots of continuous functions on a given interval (polynomials in particular).

Both these challenges are far from trivial and are described in some detail in Howey and Long [6].

5. LaTeX Plan Validation Report

One of the benefits of VAL is that it can automatically produce a LaTeX report of the plan validation. The report includes: the original plan, the plan to be validated (semi-simple plan), a step by step account of plan validation, plan repair advice if necessary and graphical diagrams. The advantages of the LaTeX report over a simple text output are numerous, but foremost is its clarity of presentation. It is easy to see each detail of the plan validation with simple use of LaTeX text formatting, which also provides a more formal record. The features of the report may also be used in other documents, for example figures 2, 3 and 5 are taken from reports generated by VAL.

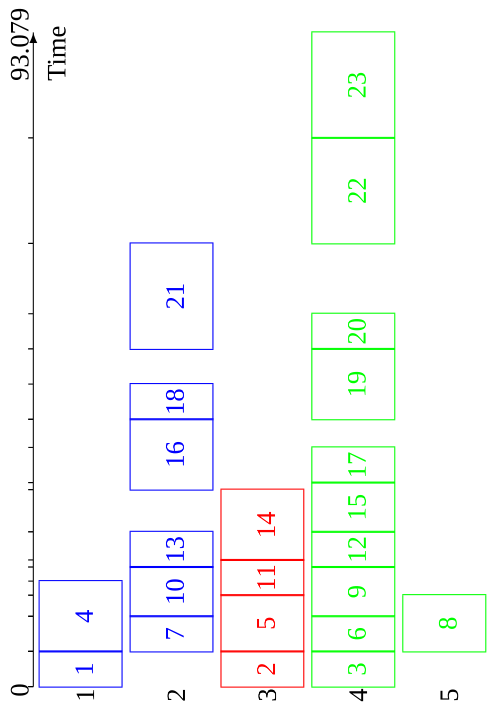

Figure 2. Gantt Chart

5.1. 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 consists of a number of rows against a time line axis, a durative action is shown as a bar, a non-durative action as a line. The actions are sorted into rows by considering each action in turn, as given in the original plan, 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 is placed in a new row.)

Plans are usually structured so that dependent actions immediately follow one another, thus rule 1. gives a sensible way of arranging the actions. A domain may contain a set of objects that represent executives, it may then be desirable to highlight those actions that affect these executives separately. Therefore, it is possible to group the actions with the same *tracked* object as a parameter. An extra rule is then followed:

3. If an action has a tracked object as a parameter then it

can only be placed in a row where the actions in this row also have this tracked object as a parameter.

Figure 2 shows an example of a plan's Gantt chart using VAL (a key is included in the LaTeX report), given from the 2002 planning competition using the simple time rover domain. Each tracked object has its rows coloured the same colour and grouped together. 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).

5.2. 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, linear changes, and non-linear changes in value. The interaction between PNEs can be clearly observed as shown by the two graphs showing the distance and speed respectively of a car driving (figure 3). Graphs of PNEs can be useful when trying to repair an invalid plan, for example when refuelling a fuel tank which overflows. It could be seen from the graph when is a more suitable time to refuel.

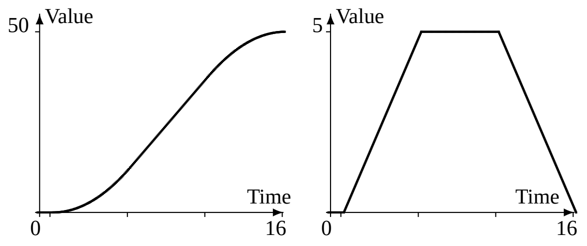

Figure 3. Graphs of the distance and speed of a car from VAL's LaTeX report.

6. Mixed Initiative Planning

6.1. Mixed-Initiative Planning and Plan Repair

When VAL is used in its simplest form, without any parameters, in the case of plan failure it reports only that the plan has failed. An option is available for verbose output in which the system generates a report explaining which actions in a plan have failed. However, this is still of limited use since no indication is given of why an action precondition is unsatisfied. The action precondition might be very complex, but only have failed due to one literal with the incorrect truth value. For example, a large factory machine may have an action for starting processing with a complicated precondition, but an instance of the action in a plan might fail simply because the machine is not switched on prior to planned execution of the start action. Feedback

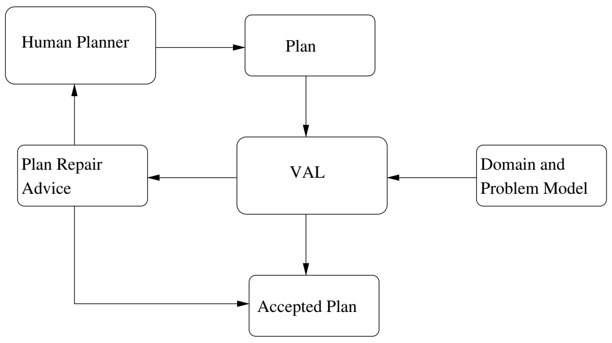

Figure 4. Mixed-initiative planning with validation-repair component.

from the plan validation reporting that the machine needs to be switched on would be invaluable advice on how to fix the plan. In complex plans identifying even simple failures such as this can be difficult due to the obscuring effects of the actions surrounding the failure.

With the intention of supplying more informed feedback we have developed in VAL a detailed advice sub-system indicating how to satisfy unsatisfied preconditions in an invalid plan. The advice can be used in a *mixed-initiative planning* cycle in which the human planner firstly produces a plan either by hand or with the aid of software before VAL simulates execution of the plan giving detailed advice on how to repair the plan for each unsatisfied action precondition (or invariant condition or goal). The advice can then be used by the human planner to produce a new plan correcting the errors, or at least some of them. The new plan can then be executed using VAL which produces new plan repair advice, and so on. The process is illustrated in figure 4.

In general, the advice offered by VAL indicates why a given plan failed and what conditions must be achieved in order to repair it. It does not indicate which actions might be applied to achieve those conditions or explore the interactions they might introduce into the plan if they are added to it. Therefore, the advice from VAL must be seen as the first stage in the repair or reconstruction of a flawed plan: other components are necessary to decide how best to act on the advice if this decision is to be made automatically.

Structure of plan repair advice. The advice given for a failed precondition is derived from a PDDL precondition expression and stored in a structure called an *advice proposition*.

Definition 6.1 Advice Proposition *For a given* PDDL *precondition of an action in a plan the* advice proposition *provides instructions on how the state, S, must be altered at this point in the plan in order to satisfy the precondition. An advice proposition (AP) is one of the following:*

- *Instructions to set A to true, for some literal A.*

• *Instructions to set A to false, for some literal A.*

• *Instructions to satisfy a comparison consisting of numerical expressions where each PNE has its current value reported.*

• *A list of APs where* all *must be followed (conjunction AP).*

• *A list of APs where* at least one *must be followed (disjunction AP).*

• *No advice (the empty advice case).*

VAL produces an advice proposition for each unsatisfied precondition given by a mapping of a PDDL precondition and state to an advice proposition.

Definition 6.2 *Let ϕ be the mapping from a* PDDL *precondition, P, and a state, S, to an advice proposition defined as follows if P is a literal, comparison, conjunction, disjunction and implication respectively.*

$$\phi(P,S) := \textit{if } S \models P \textit{ then } \text{no advice } \textit{else } \text{set } P \text{ to true}$$
$$\phi(P,S) := \textit{if } P \textit{ is an unsatisfied comparison then } \text{satisfy } P$$
$$\phi(\wedge_i X_i, S) := \wedge_i \phi(X_i, S), \textit{ for each unsatisfied } X_i \textit{ in } S$$
$$\phi(\vee_i X_i, S) := \vee_i \phi(X_i, S), \textit{ for each unsatisfied } X_i \textit{ in } S$$
$$\phi(X \rightarrow Y, S) := \phi(\neg X \vee Y, S)$$

If P is a negation, $P = \neg Q$, then $\phi(P) = \psi(Q)$ where ψ is defined as below if Q is a literal, comparison, conjunction, disjunction, implication and negation respectively.

$$\psi(Q,S) := \textit{if } Q \not\models S \textit{ then } \text{no advice } \textit{else } \text{set } Q \text{ to false}$$
$$\psi(Q,S) := \textit{if } Q \textit{ is a satisfied comparison then } \text{do not satisfy } Q$$
$$\psi(\wedge_i X_i, S) := \vee_i \phi(\neg X_i, S), \textit{ for each satisfied } X_i \textit{ in } S$$
$$\psi(\vee_i X_i, S) := \wedge_i \phi(\neg X_i, S), \textit{ for each satisfied } X_i \textit{ in } S$$
$$\psi(X \rightarrow Y, S) := \phi(X \wedge \neg Y, S)$$
$$\psi(Q,S) := \phi(Q', S), \textit{ if } Q = \neg Q'$$

Notice that the map ϕ is well defined since PDDL preconditions and states are finite, and that starting from a PDDL precondition that is not satisfied always yields a non-empty advice proposition. The advice will take the form of lists of advice which must either all be followed or one of which must be followed, further advice lists may then be nested. The actual conditions that need to be changed in the state will be the truth value of predicates and the numerical values of PNEs.

Advice on invariants depending on continuous effects The introduction of continuous effects into a plan further complicates the validation of an invariant over a given interval, as discussed in section 4.1. There is a natural extension to the plan repair advice given by ϕ to invariant conditions depending on continuously changing PNEs. An invariant condition must hold for all values on a given interval, this further consideration only changes the advice given by ϕ for comparisons that depend on continuously changing PNEs. Instead of considering just one state the advice for satisfying an invariant must consider: one logical state (for the predicates), and a continuously changing numerical state on the interval in question for comparisons depending on continuous effects. The advice for such a comparison is that it needs to be satisfied on an interval with a report of the subset of values of the interval on which the comparison is satisfied.

For a disjunctive advice proposition which states that one of the several disjuncts be satisfied the meaning should be interpreted appropriately when referring to invariants conditions. That is, for each time value in the invariant interval one of the advice propositions must be followed. The advice proposition that is followed need not be the same advice proposition for each time value. See Howey and Long [6] section 7.2 for more details on disjunctive invariants.

Extending advice into plan repairs. The advice generated by VAL identifies the flaws in a plan and what conditions must be achieved to repair the plan. To actually construct a repair requires further identification of the means of achieving these conditions. In the worst case, this is equivalent to planning, but in many cases there are limited choices that can be used and that represent simple repairs to the flaws. In particular, where there is an achieving action already present in the plan, or where only one action could be used to achieve the condition, the repair is clear. We identify these cases and enact the repairs directly. Where there are choices we report these to the human in order to make a useful selection between them.

Repairing plans with continuous effects (example). Consider a battery power model, as shown in figure 5. The graph shows how power consumption by a system varies over a temporal interval in which the system is both recharging (from solar-power) and is engaged in various power-consuming activities. This example is taken from our work on operations planning for Beagle 2 (for more details see [7]). Interacting activities can cause the power level to dip below a minimum threshhold. In such a case, VAL observes that the invariant (that the charge remain above the critical level) is unsatisfied over certain intervals. The curve that

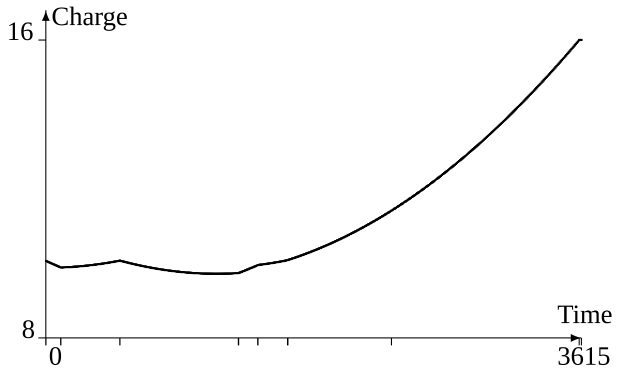

Figure 5. Graph of Charge.

VAL produces shows that the charge dips low early in the interval when powerdraw exceeds solar-power generation. The shape is complex due to the changing demands of the activities across the interval. VAL is able to recognise this flaw and propose advice about how to resolve the conflict.

The process of repair depends on a rich plan representation, capturing the dependency structure between the actions and possible external events. This temporal aspect of the structure is an important additional element, along with the effect of continuous change, that is not considered in the otherwise closely related work of Lemai and Ingrand [8] on plan repair. In their work they consider plans as partial order structures and build repairs using traditional partial-order planning flaw repair strategies. This is a valuable approach to handling plan flaws and can be generalised to handle metric conditions to some extent. However, they do not consider continuous change or its impact on invariants and this is an important aspect of the current work.

7. Conclusions

Polynomials were the first continuous effects to be handled by VAL [6]. The approach can be extended to continuous effects described by more complex functions, by using polynomials to approximate the functions. In our implementation of exponential functions in VAL we have represented them directly for improved performance. This extension has been an important step in the exploitation of VAL in the context of the Beagle 2 operations, since the power models are sufficiently complex that they cannot easily be modelled using simple polynomials. In fact, it has been necessary to numerically solve certain classes of differential equations.

The mixed initiative aspect of VAL is part of a larger project to explore the transfer of a variety of planning technologies into space operations planning in European space missions. This should be seen in the context of the world-wide efforts in space exploration and the NASA mobile robots missions currently being pursued on Mars. These missions also make extensive use of mixed-initiative planning aids, including MAPGEN [1], which has proved a very successful tool in the Mars Exploratory Rover missions.

As planning technology is applied to real application problems, the need to provide solutions to problems that are often not considered in the pure planning research community becomes more apparent. Mixed-initiative planning has long been considered an important bridging technology in moving towards automatic planning, while also solving some of the difficulties faced by planners in complex and realistic domains. We have used VAL, our plan validation technology, as a tool in mixed-initiative planning for space operations. An important aspect is that it is directly linked to our development of the semantics of PDDL, therefore providing a basis for formal validation of the domain descriptions we are constructing. The importance of continuous change in this domain is an added complication. We have advocated the role of continuous change in planning domain models for some time and this domain is an illustration that it can be of key importance in real problems.

Plan validation and plan repair are important aspects of a mixed-initiative planning system, but they also form the foundations of fully automated planning. The extension of existing planning technology to address continuous resources and complex metric constraints is a challenge that we believe must be addressed in order to apply planning in a wide range of application scenarios.

References

[1] M. Ai-Change, J. Bresina, L. Charest, J. Hsu, A. Jónsson, R. Kanefsy, P. Maldegue, P. Morris, K. Rajan, and J. Yglesias. MAPGEN: Mixed intitive activity planning for the Mars Exploratory Rover mission. In *Proceedings of Demonstration Systems Track, ICAPS'03*, 2003.

[2] J. Allen and G. Ferguson. Human-machine collaborative planning. In *Proceedings of 3rd International NASA Workshop on Planning and Scheduling for Space*, 2002.

[3] A. Blum and M. Furst. Fast Planning through Plan-graph Analysis. In *Proceedings of IJCAI*, 1995.

[4] M. Burstein and D. McDermott. Issues in the development of human-computer mixed-initiative planning. In B. Gorayska and J. Mey, editors, *Cognitive Technology*. Elsevier, 1996.

[5] M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of AI Research*, 20, 2003.

[6] R. Howey and D. Long. Validating plans with continuous effects. In *Proc. of the 22nd Workshop of the UK Planning and Scheduling Special Interest Group*, pages 115–124, 2003.

[7] R. Howey, D. Long, and M. Fox. Plan validation and mixed-initiative planning in space operations. In D. Borrajo, editor, *Proceedings of ECAI Workshop on Planning: Bridging the Gap between Theory and Practice*, 2004.

[8] S. Lemai and F. Ingrand. Interleaving temporal planning and execution in robotics domains. In *Proc. AAAI'04*, 2004.

[9] D. Long and M. Fox. The 3rd International Planning Competition: Results and analysis. *Journal of AI Research*, 20, 2003.

[10] J. H. Mathews and K. K. Fink. *Numerical Methods Using Matlab*. Prentice-Hall Inc., New Jersey, USA, 4th edition, 2004.

[11] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, volume 2, pages 634–639, Anaheim, California, USA, 1991. AAAI Press/MIT Press.

[12] M. Veloso, A. Mulvehill, and M. Cox. Rationale-supported mixed-initiative case-based planning. In *Proceedings of the 14th National Conference on AI and 9th Innovative Applications of Artificial Intelligence Conference*, 1997.

[13] M. Woods, R. Aylett, D. Long, M. Fox, and R. Ward. Assessing planning and scheduling technologies for deep space exploration. In *Proceedings of 4th British Conference on (Mobile) Robotics: Towards Intelligent Mobile Robots*, 2003.