An Investigation into the Expressive Power of PDDL2.1

Maria Fox and **Derek Long** and **Keith Halsey**
Department of Computer and Information Sciences,
University of Strathclyde, Glasgow, UK
maria.fox,derek.long,keith.halsey@cis.strath.ac.uk

Abstract. The planning domain language PDDL2.1, used in the 3rd International Planning Competition, has sparked off some controversy in the planning community as researchers consider its expressive power and the ease with which interesting domain models can be constructed in the language. In this paper we show that the expressive power of PDDL2.1 is much greater than is commonly believed. We demonstrate that PDDL2.1 can model many of the domain features often claimed to lie beyond its modelling capability. In so doing we provide a means by which powerful domain features and language constructs can be given a semantics in terms of the canonical basis of PDDL2.1.

1 Introduction

The Planning Domain Description Language, PDDL, was first developed by a committee led by Drew McDermott [11], for use in the International Planning Competition (IPC) series. For the 3rd IPC, Fox and Long [7] expanded the language to include temporal features and to support the expression of plan metrics. While the core of the original language proposal concentrated on STRIPS and ADL features, with an expressive power that has been explored quite thoroughly [4, 14], the new features of PDDL2.1 enter territory that is not so well understood. In particular, since there are alternatives for the expression of temporal features, it is of interest to understand what can and cannot be expressed in the syntax of PDDL2.1 in order to understand where its limits lie. Helmert showed that the metric aspects of PDDL2.1 [8] make the plan existence problem undecidable. This result is important but it does not imply that one has practical access to huge expressive power. Indeed, whilst it might be theoretically possible to encode any feature using Gödel numbering, this does not help to answer the question of whether a certain family of constraints is efficiently expressible in the language. For example, we might be concerned with whether we can express predetermined exogenous events by means of a polynomial expansion of a PDDL2.1 domain description. The notion of the polynomial expressibility of a family of constraints is defined below. In this paper we restrict out interest to the consideration of temporal constraint families.

Temporal constraints that have been proposed as important for planning, but which are not an explicit part of PDDL2.1, include deadlines, predetermined exogenous events (fully exogenous events, which are unpredictable, cannot be modelled without non-determinism), durative actions that support logical change in state at other times than the start and end point and more complex temporal constraints for plans to satisfy. Constraints might require that certain conditions never become true during the execution of a plan or that certain conditions be always true during the execution of a plan. Constraints might also refer to windows during the plan execution, either tied to absolute times or to triggering activities within the plan.

In this paper we consider four language features: temporally extended goals, durative actions having effects at multiple time points, exogenous events and certain interval temporal constraints. We argue that all of these constraint forms are expressible within PDDL2.1 by means of polynomial transformations. This means that the addition of syntax to the language to support their convenient expression should be seen as "syntactic sugar" mapping into canonical representations in PDDL2.1. Where no polynomial time transformation is possible a feature can be seen as necessitating an extension to the core language. Hoffmann and Thiebaux's axioms [17] and continuous numeric effects (see [12] and [9]) are of this kind. This paper is not concerned with the practicality of these transformations for planning, only in determining the expressive power of PDDL2.1. Exploration of their potential in practical planning is discussed in [5].

2 Definitions and Terminology

Definition 1 *A* planning instance *is a pair $I = \langle D, P\rangle$ where D is a domain description and P is a problem description, in PDDL2.1*

Definition 2 *Given a planning instance $\langle D, P\rangle$ and a set of constraints C, an* original plan *is a plan for $\langle D, P\rangle$ that satisfies C.*

In order to explore whether particular constraints can be expressed within PDDL2.1 we consider the following process. A planning instance is provided, with a collection of constraints. These include arbitrary temporal constraints that might be applied to planning domains, problems or to potential plans for these planning instances. The initial expression of these constraints is obviously not in PDDL2.1, since we want to examine whether they can, in fact, be transformed into an equivalent PDDL2.1 formulation. Thus, we do not need to specify how these constraints are formulated: where we consider specific families of constraints we use commonly accepted representations that we briefly describe. In each case we define a transformation that takes as input the planning instance and the constraints that must be expressed and generates a new planning instance, expressed in pure PDDL2.1. For the transformation to achieve its goal, it is necessary to show that a solution to the new planning instance can be interpreted as a solution to the original planning instance satisfying the constraint set.

Definition 3 *A family of constraints F is* expressible *if, given a planning instance I and a set of constraints C in F that must apply to plans generated for I, it is possible to transform I and C into a converted planning instance I' in such a way that any plan for I' can be interpreted to yield a plan that satisfies I and C.*

Within the context of the previous definition we refer in the following to *original* and *converted* planning instances and *original, converted* and *interpreted* plans.

Definition 4 *A family of constraints is* polynomially expressible *if it is expressible, the transformation is polynomial in the size of the input instance and the constraint set, and the interpretation is polynomial in the size of the plan.*

Since PDDL2.1 problem definitions may contain a plan metric, describing the way in which plans are to be evaluated and compared, it is important that our transformation should also preserve the ordering implied by the metric. The main implication of this is that we must be careful, in applying transformations, not to allow the durations of plans for the converted instances to have an unpredictable relationship with the durations of the corresponding interpreted plans. The following definitions introduce useful terms.

Definition 5 *A* happening *is the collection of action start and end points that occur at the same time point in a plan. The happenings of a plan therefore define the finite collection of time points at which things "happen". A* true happening *is a happening in the converted plan that corresponds to a happening in the interpreted plan. Two happenings are* ϵ-separated *if the first happens at time* t_1 *and the second at time* t_2 *and the difference between* t_1 *and* t_2 *is greater than or equal to* ϵ*.*

Definition 6 *A* generated proposition *is a new dummy proposition that is unique in the plan (not referred to by any domain action or problem element).*

We consider as our core language the PDDL2.1 extensions that support the modelling of durative actions with fixed or computed durations, but without duration inequalities. Duration inequalities provide additional expressive power, but they have been little explored by the community and we consider it of interest to show that the temporal constraint families discussed in this paper are – in principle – within the capabilities of existing planning systems.

In the semantics of PDDL2.1 plans all action end-points occurring at the same time point are considered to be simultaneous and not ordered. Thus it is important that simultaneously occurring start or end points should not interfere with one another in any way. We enforce a strong mutex relation that guarantees non-interference and this results in a need to separate conflicting start or end points by a non-zero quantity of time. In order to acknowledge the limitations of the underlying computer systems supporting the construction, validation and execution of plans, the need to fix the minimum value of separation, called ϵ, is imposed in the definition of a planning instance or family of instances. These issues are discussed in the published description of the PDDL2.1 language and semantics [7].

The transformations that we describe below involve enforcing relationships between the start or end points of actions in a plan solving a given planning instance. To enforce these relationships it can be necessary to insert activities within the ϵ-sized gaps between these actions. These activities are represented at a lower granularity than that visible to the validation of the original plan, therefore their insertion into the ϵ-sized gaps can be seen as *climbing inside time*. It might seem contradictory to split ϵ when its role is to ensure a minimal temporal separation. However, the subdivisions of ϵ occur only within plans for the converted planning instance and would be invisible to the validation process for the original plan. The process of interpretation removes these subdivisions so that the plan satisfying the expressed constraint set can be validated. Similarly it might seem, in the following transformations, that we are contradicting the observation that no executive can measure time more precisely than at some level of granularity ϵ. In these transformations it is necessary for actions to be placed very precisely with respect to ϵ, appearing to require an ability to measure more precisely than ϵ allows. In fact, this extra precision is only used to express constraints imposed by the world, so the behaviour of the executive will have to respect these constraints in order to interact correctly with the world. The choices that the planner appears to be making are actually forced upon it by the way the world is modelled. In this sense, a transformation is exactly like a component design in any proof of complexity: the elements of the component slot together in a precise way to achieve a desired theoretical behaviour.

We now present a collection of components that we have designed to enable the expression, in PDDL2.1, of a range of domain modelling features not explicit in the language. Afterwards we present several transformations to demonstrate their use. This paper does not present a complete collection of all of the features that are expressible in PDDL2.1 by means of transformation: this collection grows as new features are considered. We do not claim that the transformations constructed are convenient for use in domain modelling. Our view is that modelling convenience is an issue that should be kept separate from that of the expressive power of a language.

2.1 The Strut and the Clip

The strut enforces a minimum interval between two happenings. It is a durative action, where the duration is equal to the minimal separation required. The strut is associated with two generated propositions: s_1 is added as an effect of the preceding action end-point and an invariant of the strut, and is deleted by the succeeding action. The second proposition s_2 is added as the start effect of the strut and the start condition of the succeeding action. This configuration ensures that the strut appears between the two start or end points, so that they are minimally separated. Because the pairs of points are non-mutex the end of the preceding action can abut with the start of the strut and the end of the strut can abut with the start of the succeeding action.

The transformation of the original domain to enable the use of the strut requires the addition of the strut actions and the modification of the conditions and effects of the original actions. This work is polynomial in the size of the original domain description and the collection of constraints that has to be expressed. There is then polynomial work required to interpret the interval constraints in the converted plan.

The clip forces non-mutex action end-points in the original grounded planning instance to coincide in the converted plan. We can distinguish between the tight clip, which plays this role, and the loose clip, which ties two or more happenings to lie within a certain interval of one another. Happenings can be forced to lie a fixed distance apart by using a strut and a clip attaching the two happenings to the two ends of the strut.

The clip is associated with a generated proposition h, which appears as a start effect and an invariant and is deleted as an end effect. The proposition h is made a precondition of the end-points that need to be clipped together. The duration of the "tight" clip is 2ϵ. This forces the clip to be placed equidistantly around the start or end points being clipped together. The clip can clip together as many end-points as are required to abut at a particular instant. It is necessary to ensure that everything that needs to be clipped together is forced into the same clip. To ensure this we make each of the end-points i in the collection achieve a new generated proposition, e_i, which we add as

a precondition of the end-point of the clip. We also ensure that the clip deletes the effect e_i in order to "clean up" after its application.

In general, if the length of the clip is greater than 2ϵ then the happenings that lie within it are not constrained to be simultaneous but just to lie within the length of the clip of one another (less 2ϵ to allow for the need to separate mutex actions at the two end points of the clip). We refer to this construct as the *loose clip*. This provides a means for expressing interval constraints, as we describe below.

Usually the clip will appear strictly inside the converted plan structure, meaning that it does not affect the duration of the original plan. In the special case where the clip extends beyond the end of the plan, so that the duration is affected, it is straightforward to recognize and remove the extra ϵ from the duration.

The exploitation of clips introduces only a constant amount of work in each abuttal case and the number of abuttal cases is polynomial in the length of the converted plan. In constructing the converted domain, a different clip needs to be made for each pair requiring clipping. The number of pairs is polynomial in the size of the original domain so therefore a polynomial number of clips is required.

2.2 The Epsilon Slice

In order to climb inside time we need a mechanism for breaking into an ϵ period. This means that we need to define a smaller-grained ϵ', and the size of this depends on how much activity needs to be fitted into the ϵ-sized gap. We consider the case in which two activities need to be inserted to play a specific role. In general it might be necessary to fit more activities into the period. This can be achieved but we do not consider the general case in this paper. For the transformations that we consider later the special case suffices.

The ϵ-slice allows happenings in the converted plan to be separated by $\epsilon/3$, instead of by ϵ. In the interpreted plan only ϵ-sized separations are visible, so actions in the original domain appearing in the converted plan must be ϵ-separated. To ensure this we introduce the *magnet* whose end-point "poles" repel one another. Instances of the magnet act as spacers to prevent happenings in the interpreted plan from getting closer than ϵ together. The difference between a magnet and a clip is that the magnet forces end-points apart, so each magnet encloses just one end point, while the clip forces them to abut and so encloses two or more start or end points in the original plan.

The magnet has duration $2\epsilon/3$ and is associated with a generated proposition m. It has a start effect m, an invariant m and an end effect $\neg m$. This prevents the magnets from being interlaced, and because their end-points are mutex they are automatically separated by $\epsilon/3$. Magnets must be placed equidistantly around happenings because no two happenings in the converted plan can be closer than $\epsilon/3$ apart. Thus, as shown by figure 1, is that actions from the original domain, appearing in the converted plan, must be ϵ-separated.

In order to force a magnet to be placed around every true happening in the converted plan we modify the actions in the original domain to have m as a precondition of both start and end. The transformation of the original domain and problem into the converted domain and problem is polynomial in the number of actions in the original domain. Each action has two new conditions added, and we have added one new action - the magnet. The interpretation process is polynomial in the length of the plan, requiring only that the magnets be removed. The length of the converted plan is polynomial in the length of the original plan: one magnet has been placed around every happening in the original plan.

In order to preserve the meaning of the metric of the original problem it is necessary to replace total-time in the original metric with total-time $-\ \epsilon/3$ in the converted problem. This is needed because the duration of the converted plan will be extended by the final $\epsilon/3$-sized interval overlapping the end of the original plan. It is also necessary to ensure that the first true happening is ϵ-separated from the initial state. This can be enforced by adding an instantaneous action *magnets-power-up* to the converted plan, preceding every other action and occurring at $\epsilon/3$ from the initial state. Its effect, *m-on*, is added as a precondition of the magnet.

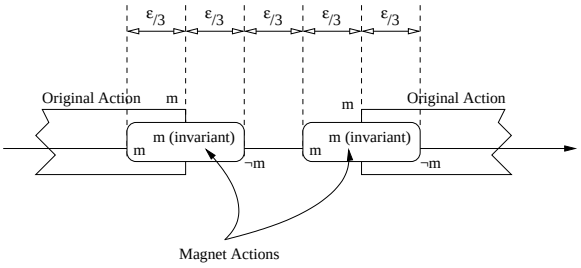

Figure 1. Magnets enforce temporal separation in the converted plan and therefore the separation of start or end points of the original actions in the interpreted plans. Note that we are following a convention of listing effects at the base of an end point on the right, and conditions at the top of the end point on the left.

3 Transformations

We now show how, using the components described in the previous section, it is possible to model in PDDL2.1 some of the language features most often proposed as desirable.

3.1 Maintenance Goals and Deadline Goals

A maintenance goal is of the form: $\Box c$, where c is a condition and the constraint expresses that it must hold at all times in a plan. One can also express $U\,p\,c$ and $F\,p\,c$, where U stands for *Until* and F stands for *From*, and p is the condition until which, or from which, the condition c must hold. These are simple LTL formulas [6] considered also by Bacchus and Kabanza [2], Baral *et al* [3], Pistore *et al* [15] and other authors. We begin by considering the constraint $\Box c$ and return to the other forms afterwards.

Maintenance constraints require a condition to remain true at all times during an interval. In fact, the only opportunities for violating a condition are at the points of change in a plan: that is, the time points associated with happenings. In order to model the maintenance condition it is therefore sufficient and necessary to enforce that the condition holds over a small interval encompassing each happening. In order to achieve this we use $\epsilon/3$ slicing and attach the condition c as an invariant to the magnet action. Since every end point of an original action must be embedded within a magnet in the converted plan this will enforce c to hold invariant across every true happening.

In fact, if the propositional part of the maintenance goal is atomic there is no need to use the $\epsilon - slice$. One can ensure preservation of the goal by a simple pre-processing of the domain to remove the deleting actions. However, in the general case, where the propositional part is a formula, this solution will not work because the actions to be removed cannot be uniquely identified.

To deal with goals of the form $U\,p\,c$ we need to introduce a condition-checking action. Using $\epsilon/3$ slicing the condition-checking action has duration $2\epsilon/3$. It has start effect cc, invariants c and cc and end effect $\neg cc$. We attach the precondition $wp \vee cc$ to every

original action start and end point, where wp is a generated proposition meaning that p was achieved. This triggers the moment at which c can cease to hold. To complete the transformation we need an additional instantaneous action with precondition p and effect wp. This means that until p is achieved a plan must include a constraint-checking action enforcing c around each true happening. After p has been achieved the constraint-checkers are no longer needed, but the magnets continue to be required because of the $\epsilon/3$ slicing. Figure 2 (part a) shows the details.

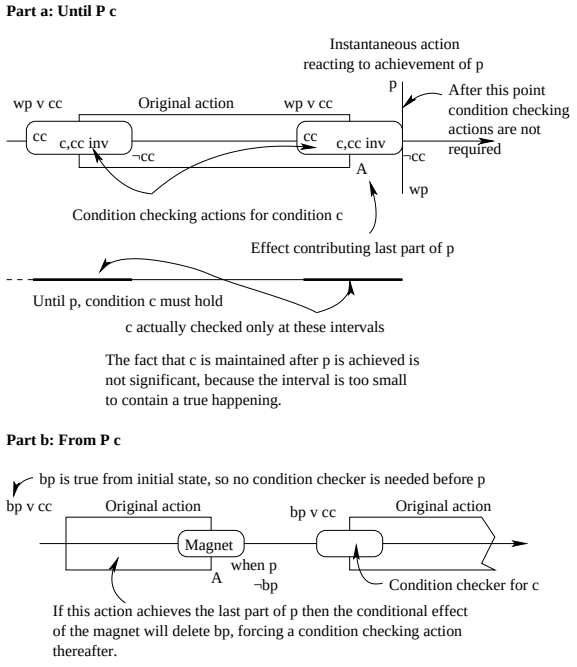

Figure 2. The enforcement of $U\,p\,c$ and $F\,p\,c$ maintenance constraints.

Goals of the form $F\,p\,c$ require slightly more subtlety. It is again necessary to perform $\epsilon/3$ slicing. A condition-checking action is required, as before. The new precondition added to every action start and end point is $bp \vee cc$, and we add bp to the initial state conditions. Since bp is true from the outset all actions can be applied without constraint, but now it has to be ensured that bp is deleted when p becomes true. To achieve this we add a conditional effect to the end point of the magnet, of the form `(when` $p\,(\neg bp)$`)`. Because magnets must be applied whenever the original actions are applied this ensures that, as soon as p is satisfied bp will be deleted and the planner must then apply condition-checking actions around each true happening thereafter. Figure 2 (part b) illustrates this transformation.

A *deadline goal* is a goal with a deadline by which it must be achieved. It can be expressed $(g$ `by` $t)$, where g is a goal proposition and t is an absolute time point.

Our transformation in this case relies on the maintenance goal form $F\,d\,g$ described above, where d (deadline-reached) will be a dummy proposition that we ensure is available at time t. We assume that, once achieved, a goal has to remain true. If this is not the case we can alter the transformation slightly to express the requirement that the fact that g was once achieved is available from time t.

The mechanism involves creating a durative action, D, of duration t. It is forced to be applied before any other action by the fact that it achieves at its start a generated proposition, tg, which is added as a precondition to all of the actions in the original domain description. D has end effect d and we use the technique for ensuring $F\,d\,g$ to complete the transformation. Because we are using $\epsilon/3$ slicing (by virtue of using the $F\,d\,g$ mechanism) the durative action is applied at $\epsilon/3$ after the initial state. This time period is too small to be seen after interpretation so we do not need to be concerned with modifying the duration of D to account for its separation from the initial state.

If the goal is not required to continue holding forever after its achievement we can use the dummy proposition wg to express the desired constraint. Now we use $F\,d\,wg$ instead of $F\,d\,g$, to express the fact that g was achieved by d. To enforce the achievement of wg we add an instantaneous action with precondition g and effect wg. All other details of the transformation remain the same. It might seem that the need to add an action to check the achievement of the goal could result in the accumulation of periods of length $\epsilon/3$ until duration constraints in the original problem have been violated. In fact, this does not arise here as figure 3 shows. D can be applied at time $\epsilon/3$. The transformation of constraint $F\,d\,wg$ includes a magnet around the end point of D that will respond to the achievement of d, forcing all subsequent activity to depend on a condition checker for wg. This implies that by $t + 2\epsilon/3$ all subsequent activity will require wg. Thus, the action that achieves wg, with precondition g, must be applied by $t + \epsilon/3$, requiring that g itself is achieved no later than t, which is the intended original deadline.

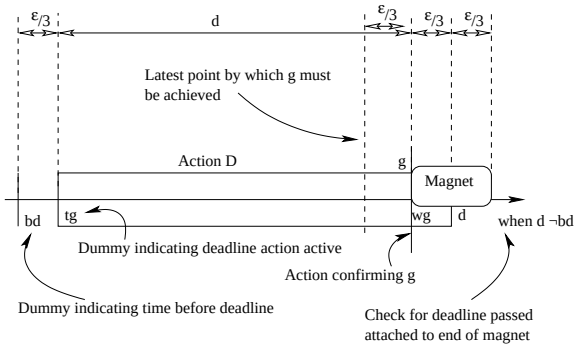

Figure 3. The structure of activity around the deadline action, when the goal need not persist after the deadline.

Note that a plan can postpone application of D, but doing so has the effect of simply translating the origin since no actions can be used before D starts.

3.2 Predetermined Exogenous Events

Predetermined exogenous events are effects that will occur at times specified in the initial state, independent of any planned activities. They can be easily modelled using the simple "tight" clip. Given predefined times $t_1, ..., t_m$ at which events $e_1, ..., e_n$ will occur, We create a collection of durative actions, $d_1, ..., d_m$, of durations $t_1, t_2 - t_1, .., t_m - t_{m-1}$ respectively. These durative actions have as their effects the event propositions associated with the time points corresponding to their durations. We ensure the correct ordering of these actions, and their occurrence at the correct times, by clipping them together using clips. To force the chain to begin immediately we add a generated proposition $echain$ as a start effect of the first action. This is then added as a precondition of every action in the domain. This mechanism has the additional effect of forcing the first true happening to occur ϵ after the start of the first action in the event chain sequence, which is, in turn, at least ϵ after the initial state. The

first true happening will then be 2ϵ after the initial state. Therefore, we adjust the plan during the interpretation stage by shunting every step back by t, where t is the time at which the first action in the event chain is applied. The plan metric must be modified to use `total-time` $-\epsilon$ in place of `total-time`, ensuring that the transformed instance is optimised with the same ordering on plans as the original planning instance imposes. An alternative approach to this adjustment during interpretation is to use ϵ-slicing, as for deadlines.

3.3 Interval Temporal Constraints

It is sometimes necessary to separate activities in a plan by fixed intervals. The use of interval constraints has been considered in several planning systems, such as HSTS [13] and IxTeT [10].

Sometimes the interval between two actions needs to be quite precisely specified, in which case a strut can be used, together with two tight clips to force the strut to intervene between two actions. When the interval constraint is less precise, requiring only that two actions happen within a certain interval of one another, a loose clip can be used to surround the end point of the first actions and the start point of the second.

To enforce that two actions *start or end at the same time* we use a tight clip around the start points or end points respectively. To ensure that they *abut* we clip the end of the first action to the start of the second. Again, this is a tight clip. To ensure that one action *overlaps* another we place a loose clip around the start points of the actions, where the duration of the clip is equal to the duration of the action that starts first. To make one action *contain* another we place loose clips around the two start points and the two end points, where the duration of the clip is equal to the difference in duration between the encloser and the enclosed.

All of these transformations are simple and do not necessitate ϵ-slicing. Using these transformations we can express any of the constraints of Allen's interval calculus [1] within the point-based language of PDDL2.1.

3.4 Durative Actions with Intermediate Effects

IxTeT [10] uses a language supporting the expression of multiple intermediate effects, and the benefits of these for modelling have been observed. It is true that forcing all effects to take place at the start and end points of actions, as in PDDL2.1, imposes constraints on how activity can be modelled. However, it does not represent any constraint on expressiveness. It is possible to define durative actions with multiple intermediate effects using the durative actions of PDDL2.1 and a combination of loose and tight clips. No ϵ-slicing is required to achieve the transformation, so it seems that these actions do not in fact represent a significant advance on the PDDL2.1 durative action.

The strategy for modelling an action, A, with intermediate effects is as follows. Suppose that the intermediate effects are a collection of propositions $p_1..p_n$ occurring at times $t_1..t_m$, each time t_i lying between t and $t + d$ (the start and end times of A. A collection of PDDL2.1 actions is defined, in a way similar to the modelling of exogenous events defined above. The actions have durations $t_1, t_2 - t_1, .., t_m - t_{m-1}$ and end effects corresponding to the propositions associated with their durations. The start effects of A will be start effects of the first action in the sequence. Tight clips are then used to connect the actions into a sequence.

An alternative transformation approach uses a loose clip, with duration equal to the duration of A, to surround the action sequence, and then uses generated propositions to ensure that the actions inside the clip do not overlap (by having each action achieve a condition that is invariant in its successor). The interior actions are forced to pack tightly into the enclosing clip. The loose clip can be seen as a kind of packing case that exactly holds the sequence of actions needed to achieve the intermediate effects. This approach is reported by David Smith in his commentary on the PDDL2.1 language [16].

4 Conclusion

In this paper we have considered the expressive power of PDDL2.1, examining a range of modelling features suggested by the community as being amongst the most useful for expressing models of planning domains. We have introduced a small collection of powerful components that support these transformations, and we have used them to show that features including temporally extended goals, exogenous events and many interesting interval constraints can be expressed within PDDL2.1 with only polynomial transformation work.

It is important to emphasise that we are not claiming that these transformations make PDDL2.1 a convenient language for actually modelling problems that involve these features. We make a firm distinction between the expressive power of the language, and its consequent role as a canonical representation language, and the question of modelling convenience.

We do not claim that the transformations presented here are unique, or even the most efficient possible. However, they all require only polynomial work so they achieve their intended function and their aesthetic value is a question of taste.

REFERENCES

[1] J. F. Allen, 'Maintaining knowledge about temporal intervals', *Comm. ACM*, **26**, 832–843, (1983).

[2] F. Bacchus and F. Kabanza, 'Planning for temporally extended goals', *Annals of Mathematics and Artificial Intelligence*, **22**, 5–27, (1998).

[3] C. Baral, V. Kreinovich, and R. Trejo, 'Computational complexity of planning with temporal goals', in *IJCAI*, pp. 509–514, (2001).

[4] T. Bylander, 'Complexity results for serial decomposability', in *Proc. of 10th National Conference on AI*, (1992).

[5] S. Cresswell and A. Coddington, 'Adapting LPGP to plan with exogenous events and goals with deadlines', in *Proc. of ECAI'04*, (2004).

[6] E.A. Emerson, 'Temporal and modal logic', in *Handbook of Theoretical Computer Science*, volume B, 995–1072, MIT Press, (1990).

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

[8] M. Helmert, 'Decidability and undecidability results for planning with numerical state variables', in *Proc. of AIPS'02*, (2002).

[9] R. Howey and D. Long, 'VAL's progress: The automatic validation tool for PDDL2.1 used in the international planning competition', in *Proceedings of the ICAPS 2003 workshop on the IPC series*, (2003).

[10] P. Laborie and M. Ghallab, 'Planning with sharable resource constraints', in *Proceedings of IJCAI'95*. Morgan Kaufmann, (1995).

[11] D. McDermott, 'PDDL–the planning domain definition language', Technical report, Available at: `www.cs.yale.edu/homes/dvm`, (1998).

[12] D. McDermott, 'Reasoning about autonomous processes in an estimated-regression planner', in *Proceedings of ICAPS'03*, (2003).

[13] N. Muscettola, 'HSTS: Integrating planning and scheduling', in *Intelligent Scheduling*, eds., M. Zweben and M.S. Fox, 169–212, Morgan Kaufmann, San Mateo, CA, (1994).

[14] B. Nebel, 'On the compilability and expressive power of propositional planning formalisms', *Journal of Artificial Intelligence Research*, **12**, 271–315, (2000).

[15] M. Pistore and P. Traverso, 'Planning as model checking for extended goals in non-deterministic domains', in *IJCAI*, pp. 479–486, (2001).

[16] D. E. Smith, 'The case for durative actions: A commentary on PDDL2.1', *Journal of AI Research*, **20**, 149–154, (2003).

[17] S. Thiebaux, J. Hoffmann, and B. Nebel, 'In defense of PDDL axioms', in *Proceedings of 18th IJCAI*, (2003).