Modelling Sequences of Processes in PDDL+ for Efficient Problem Solving

Elad Denenberg and **Amanda Coles**
King's College London
Bush House, 30 Aldwych,
London, United Kingdom WC2B 4BG

Abstract

For many years, the planning community has adopted the mantra 'physics not advice'; indicating that a planning model should describe the properties of the problem, but should not give advice on how to solve the problem. This is in contrast with other, more widely applied, search and optimization technologies, such as Mixed-Integer Programming and Constraint Programming as well as AI technologies, such as Evolutionary Algorithms and Machine learning. As models get more complex, solving planning problems becomes more challenging, and it becomes important for us to understand how to model problems so that they can be solved efficiently by planners; in order to be able to apply planning in real life applications. In this paper we focus on a particular common pattern: the need for several effects to be applied sequentially as the result of a single decision. This may occur, for example, when an action starts a cascade of effects. In this paper, we consider different ways of modelling this in PDDL, and compare the efficiency of solving the problem using each of these models in three state-of-the-art PDDL+ planners: SMT-Plan+, OPTIC and DiNo. Our results show that the more intuitive model is less scalable on the first two, and that a model ensuring fewer happenings is less scalable on the third. By presenting this work we hope to encourage more research in developing efficient planning models for expressive domains in order to allow planning to be applied in a wider range of applications.

Introduction

Many in the Domain Independent Planning community subscribe to the philosophy 'physics not advice'; that is, since we are dealing with the design of planners that should cope with any domain, when generating a testbed the domains must not contain hints or assist the planner in solving a given problem. In contrast, in many other fields of AI and optimization the realization that no tool can perform well on all domains (the No Free Lunch Theorem (NFL) (Wolpert and Macready 1997)) has inspired much work to be carried out in the area of selecting the proper search tool for a given model, or modelling in a way that would facilitate faster search with a given a tool. This area has received relatively little attention in planning, and in particular we do not yet have a good understanding of how modelling decisions affect the performance of the most expressive planning systems: PDDL+ planners.

In this work we will discuss a simple example of matching an expressive PDDL+ model to a search tool in planning problems. The example we discuss is modelling a process sequence: a number of processes which start one after the other. Processes in PDDL+, as defined by (Fox and Long 2002), can be described as effects that act upon a variable regardless of actions taken, as long as a set of predicates is true. An example of a process is gravity acting on a falling object, or a battery being charged by solar energy as the sun rises. A process sequence may appear naturally in a domain, however, one may also encounter such sequence when modelling a series of effects.

Often in real life engineering domains one may come across a series of continuous numeric effects acting one after another upon a variable. This may happen due to an action starting a cascade of such effects. For instance a rover transmitting data back to base. The transmission itself has several phases: linking, transmitting, awaiting confirmation, receiving confirmation data. All these phases have different continuous effects on battery usage, and therefore can be represented as a cascade of continuous numeric effects on the battery charge level. This may also apply to interval constraints (Tran et al. 2017) and (Tierney et al. 2012).

Another reason for effect series may be a result of piecewise linearization of a non-linear domain (Denenberg and Coles 2018; Cao et al. 2011). For instance, power landing using rockets: A robot is landing and is under the force of gravity, at any given point it may fire its engines to reduce the velocity of the fall. The engine firing changes the velocity exponentially while the gravity changes the velocity polynomially. Mixing these functions may prove to be hard, and therefore the user might chose to approximate each of the non-linear effects as a sequence of piecewise linear continuous effects.

This paper discusses three different methods for modelling sequences of continuous effects. One uses only featuers of PDDL 2.1, clips and durative actions. Next, two PDDL+ models are examined: The first of these two is an intuitive one which might be created by an end user. The second model is less intuitive, but ensures a smaller number of happenings. We compare the perofrmance of these models empirically using three different state-of-the-art PDDL+

planners: SMTPlan+ (Cashmore et al. 2016), OPTIC (Benton, Coles, and Coles 2012) and DiNo (Piotrowski et al. 2016). We present results indicating which is the most efficient model to use for each type of planner, and an analysis of why each model is better suited to that type of planner.

Problem Definition

A PDDL+ planning problem (Fox and Long 2002) is a tuple $\langle$F,v,A,P,I,G$\rangle$[1] where:

- F is a set of propositions (facts);
- v is a set of real numeric variables;
- A is a set of actions;
- P is a set of processes;
- I ($\subseteq F$) is the initial state;
- G (a conjunction of facts from F and numeric conditions over v) is the goal.

Each action has three sets of preconditions which must hold at the start of, at the end of, and throughout its execution respectively. Preconditions are conjunctions of propositions (or their negations) and numeric conditions (which for our purposes we will assume can be represented in the form **w.v**$\{>, \geq, <, \leq, =\}$c where (w is a vector of constants and c is a constant)). Actions can have instantaneous effects at their start or end, these can be propositions that are added or deleted, or updates to numeric variables of the form v$\{$+=, -=, =$\}$**w.v** + c where v $\in$ v. In addition to this, actions can have continuous numeric effects that happen throughout their duration, most generally of the form dv/dt $\{$+=, -=, =$\}$**w.v** + c; but in this work we focus on linear continuous change where continuous effects are of the form dv/dt $\{$+=, -=, =$\}$ c. Finally, actions have a duration constraint, defining a permissible range from which the planner can select the duration of the action.

Processes comprise a precondition and a set of continuous numeric effects. They differ from actions in the semantics of their execution: nominally actions model the activities the planner can choose to take; whereas processes model exogenous happenings in the environment. If the preconditions of an action are true in a given state, then the planner can *choose* to apply that action in that state (or not to). In contrast, if the preconditions of a process are true in a given state then that process will execute automatically, the planner has no choice over this.

A solution to this problem is a *plan*: a sequence of actions from A that transforms I into G.

Often in literature PDDL+ models are referred to as *hybrid*, meaning they mix continuous and instantaneous effects. This work does not explore the hybrid property of PDDL+, rather, it aims at making use of the added expressiveness of processes.

[1]We exclude events from our definition as we do not use them in this work

Running Example: Generator Domain

Throughout the remainder of the paper we use the well-known generator domain as a running example. The objects in this domain are a generator, which has a main tank with a given capacity. Generating electricity consumes fuel from the main tank. The generator can be refuelled using an auxiliary tank.

During the plan the Generator is required to work without "choking", that is, the level of fuel in the main tank must not reach zero. When refuelling, the fuel must not spill, i.e. the level of the fuel must not be greater than the capacity of the main tank.

The consumption of fuel by the generator is assumed to be linear. And thus modelled by a single action with a continuous linear effect. Refuelling from an auxilliary tank is approximated by three piecewise linear sections as described in Figure 1. These are the linear effects that become the effects of individual actions or processes that must be sequencesd to model the effect throughout the whole duration of refuelling.

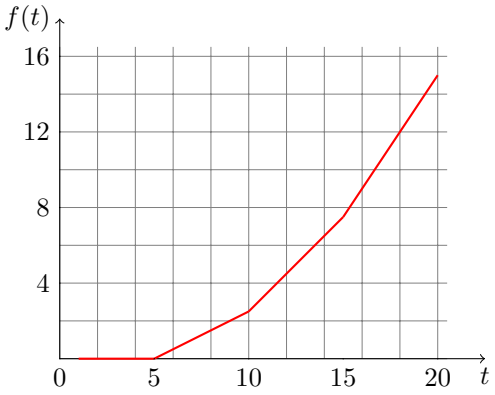


Figure 1: Linearized refuel

Modelling Sequences of Continuous Effects

We now consider the problem of modelling a sequence of conditional effects that occur as a result of a single decision. More formally, we have a physical effect E that acts continuously upon the variable v, and has an effect $f(u)$ that is defined such:

$$\frac{\mathrm{d}v}{\mathrm{dt}} = f(u) = \begin{cases} a_1 & 0 < u \leq u_1 \\ a_2 & u_1 < u \leq u_2 \\ \vdots & \\ a_n & u_{n-1} < u \leq u_n \end{cases} \quad (1)$$

where a_i are a set of contributions to the rate of v, and u_j are a set of values of u. These contributions could be constant, modelling a piecewise linear function (as those we consider in this paper) or could themselves be functions over problem variables. We present 3 different approaches to modelling this type of effect in PDDL.

Durative Action and Clip Model

It is possible to model sequences of continuous effects, in PDDL 2.1, without the need to invoke PDDL+ processes. In order to do this we make use of clips (Fox, Long, and Halsey 2004). Clips are additional actions that are used to bind together the execution of actions in order to ensure one starts as soon as another finishes.

Figure 2 illustrates this compilation: each action A_i has effect `(increase v (* #t) a_i)`. Additionally A_i ($i > 1$) has start and end preconditions that are added at the start, and deleted at the end of, C_i and C_{i+1} respectively. Finally, the start of A_{i+1} ($i < n$) adds an end precondition of ci. In this way, we can force A_{i+1} to happen immediately after Ai ends. In our generator example, A_1,A_2 and A_3 would be actions, with appropriate durations, that when sequenced model refuelling. Each has a continuous effect on *fuel* corresponding to the respective slope in Figure 1. Propositional preconditions and effects would enforce that the actions must be applied in sequence, in this order.

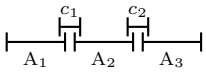


Figure 2: Clip and Durative Action Model

This representation has the advantage that it does not require a PDDL+ capable planner; although it still exhibits a level of expressiveness captuerd by few planners. It is worth noting that this representation requires a large number of actions to be added to the domain, over which the planner must search.

Modelling as Process Sequences

An alternative is to use a set of PDDL+ processes to describe the effects, as demonstrated in (Denenberg and Coles 2018).

The most intuitive way to model such a series of effects using a sequence of processes would be to define a set of processes P_i, each defined on the interval $u_{i-1} < u \leq u_i$, as shown in Figure 3 and visualized in Figure 4a. In our generator example, the refuel action would be a single action that sets a counter u to zero and has effect `(increase u (* #t) 1)`, with a start add/end delete effect of some fact $refuelling$ denoting that refuelling is happening. The processes, with precondition f and their respective u value ranges would then be responsible for updating the fuel level only during their respective ranges.[2]

It was discovered that many planners struggle with processes defined in this fashion. Therefore, we suggest the "stacking" of the processes. That is, instead of defining the process on an interval, we defining a single condition and incorporating the effect of process P_i to the effect of P_{i+1} in the following manner:

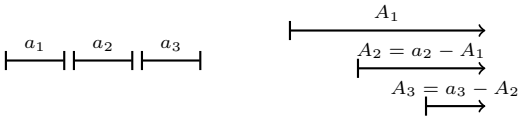
[2]Note that this model is correct for the case where refuelling with the same tank cannot self-overlap.

```
(:process Pi
 :parameters ()
 :precondition (and
        (< u u_{i-1})
        (>= u u_i) )
 :effect ( increase v (* #t a_i) )
)
```

Figure 3: Intuitive Process

$$\frac{\mathrm{d}v}{\mathrm{d}t} = F(u) = \begin{cases} A_1 & 0 < u \leq u_1 \\ A_2 & u_1 < u \\ \vdots & \\ A_n & u_{n-1} < u \\ A_{n+1} & u_n < u \end{cases} \quad (2)$$

Where $A_1 = a_1$ and $A_i = a_i - A_{i-1}$ for all $i > 1$. Only a starting condition is defined for each process PF_i. The effect of the process PF_i incorporates the new change a_i as well as the removal of the previous effect A_{i-1}. This is visualized in Figure 4b and demonstrated in Figure 5. Again, in our generator example we create a refuel action assigning u to zero at the start, and with effect `(increase u (* #t) 1)`, the processes now start in sequence, but overlap, all continuing until the end of the refuel action.



(a) Intuitive Model (b) "Stacking" Model

Figure 4: Visualization of Processes Models

```
(:process PFi
 :parameters ()
 :precondition (< u u_{i-1})
 :effect ( increase v (* #t A_i) )
)

(:process PFiplusone
 :parameters (?b - bar)
 :precondition (< u u_{i})
 :effect ( increase v (* #t A_{i+1}) )
)
```

Figure 5: Intuitive Process

Evaluation

In this section we examine two linearised versions of the well known generator domain. The first, a symmetric domain, where there is no importance as to which tank is to be used when. The second an asymmetric domain, in which the

use of tank number one must precede that of number two and so on. Using the PDDL 2.1 and the two PDDL+ models to describe these versions resulted in six domains: Symmetric and asymmetric Clips, symmetric and asymmetric intuitive model domains and symmetric and asymmetric "stacking" model domains.

On each domain four problems were tested: a generator with one,two, three and four auxiliary tanks. The problem files for all domains and problems are given in the Appendix.

The problems were tested on three state of the art planners: a PDDL+ implementation of OPTIC, SMTPlan+ and DiNo.

The Clips models proved to be inefficient: SMTPlan+ and DiNo timed out on all problems, both symmetric and asymmetric. Only Optic was able to find a valid solution, and the results are presented in Table 1 and Table 2.

The average results of the tests of the PDDL+ symmetrical domain are summarized in Table 1, and the asymmetrical in Table 2.

The results of the symmetric PDDL+ domains are as follows: In both OPTIC, SMTPlan+ the process "stacking" model performed better than the intuitive model. Using the process "stacking" model in the one tank problem proved to be 85% faster than the intuitive model. Using SMTPlan+ the fast model was 98% faster in the two tank problem, while OPTIC was unable to solve the intuitive model within 1000 seconds and timed out. STMPlan+ was able to solve the 3 tank problem within 769 secons.

DiNo was able to solve both PDDL+ models and scale well. The difference between the intuitive and "stacking model" was relatively small. Even though the process "stacking" model ensures fewer happenings, DiNo scaled slightly better while using the intuitive model.

number of tanks			1	2	3	4
OPTIC+	Clips and Struts		0.07	14.16	TO	TO
	Processes	Intuitive	22.86	TO	TO	TO
		Stacking	2.95	922.32	TO	TO
		%diff	12.90	-	-	-
SMTPlan+		Intuitive	0.22	132.39	TO	TO
		Stacking	0.03	0.72	769.33	TO
		%diff	14.09	0.55	-	-
DiNo		Intuitive	3.02	4.04	5.52	7.43
		Stacking	2.78	4.07	5.84	7.64
		%diff	92.05	100.74	105.80	102.83

Table 1: Symmetric Generator Problem Test Results

number of tanks			1	2	3	4
OPTIC+	Clips and Struts		0.07	14.06	TO	TO
	Processes	Intuitive	22.70	TO	TO	TO
		Stacking	2.89	29.51	38.78	49.86
		%diff	12.72	-	-	-
SMTPlan+		Intuitive	TO	TO	TO	TO
		Stacking	0.59	TO	TO	TO
DiNo		Intuitive	2.57	3.58	Failed	Failed
		Stacking	2.58	Failed	Failed	Failed

Table 2: Asymmetric Generator Problem Test Results

The results of the asymmetric PDDL+ domains are: OPTIC showed great improvement when using the "stacking" model, SMTPlan+ operated better on the "stacking" model, and was able to solve the porblem with one tank, though it timed out on the rest of the problems. DiNo performed better on the intuitive model. In addition, DiNo failed to run on any of the "stacking" domains.

Discussion

The problems presented in the previous section implies that the non-intuitive model is more efficient on two planners while less efficient on the third regardless of whether the domain is symmetric. In this section we will propose an explanation for the observed performance.

OPTIC solves a planning problem by transforming durative actions (and processes), to snap actions. The search in OPTIC is over happenings, which are either the application of snap actions, or the decision to start or stop a process. If a process has a precondition that is trivially false (for example, a proposition (e.g. *refuelling*) is known to be false in the state) then there is no need to make a search decision about whether that process executes or not. However, when a precondition is over a continuously changing variable (e.g. $(< \ u \ u_i)$ where u is currently subject to continuous numeric change), then the planner must make a search decision, whether to start or stop a process conditioning on it. The plan for the intuitive model requires more happenings (start refuel, start p_1, end p_1,start p_2, end p_2,start p_3, end p_3, end refuel) all at different times. For the stacked model, however, the plan: start refuel, start P_1, start P_2, start P_3, end refuel, with only 5 happenings, suffices as the planner can deduce that as soon as it ends refuel all the processes must end P_1,P_2 and P_3 immediately, as one of their preconditions *refuelling* has been deleted. This means that the solution plan appears at a lower depth in the search tree.

A similar thing happens with the SMTPlan+. This planner uses a defined number of happenings, for each happening a set of Satisfiability Problem (SAT) equations is formulated and solved. Because fewer happenings are required to solve the stacked model (the processes can end at the same happening as the refuel action), SMTPlan+ can solve the problem with fewer happenings. Since SMTPlan+ by default starts with a small number of happenings and increases this until it finds a solution; finding a solution with fewer happenings means fewer SAT problems have to be proven unsolvable to find a solution.

The DiNo planner generates states by discretization. The states generated end up being comprised of a large amount of happenings ϵ time units from one another. at each such happening the planner may chose to perform an instantaneous action, start or end a durative action or process, or update the variables using all the currently active effects. Therefore the performance of this planner depends on the number of time units the plan requires, and the user defined size of the ϵ, rather than by the number of happenings required by the domain. Furthermore, since the planner needs to update the variables every ϵ time step, the more effects acting on a variable the more calculations would be required. This is why the "stacking" model scales slightly worse in this planner.

Conclusions

This paper presented three models for each of two variants for the same problem, and the performance of three planners on these models. The PDDL2.1 model proved to be inefficient on all three planners. Two of the planners showed high sensitivity to the type of PDDL+ model, and the third, showed slightly better scalability with one domain over the other. Had this been a real life problem, the engineer solving it would have been required to either chose the planning tools to fit his problem, or to model the problem in a manner that would allow the problem to be solved within reasonable time. This is a clear demonstration of the need to research modelling in planning. In addition to emphasizing the need to study the matching of models to planners, this work also introduces a means of modelling a cascade of effects.

Modelling a cascade of effects in PDDL+ as a sequence of processes was explored: Two models proposed were tested on three planners in symmetrical and asymmetrical representation. Future work must include additional PDDL+ capable planners. In addition, the domains used here were linear, in the sense the effects were of linear change. Since the contribution of the model was in reducing the number of happenings, we do not expect a non-linear effect to show significantly different behaviour scalability wise. However, future tests will include planners performance on such non-linear domains.

Since the use of processes on intervals are slow in planners that depend on the amount of happenings, it might be useful to create a preprocessing tool for these planners, such that would parse a domain and identify the intuitive, yet less efficient model, and replace it with the "stacking" model suggested in this paper.

References

Benton, J.; Coles, A. J.; and Coles, A. 2012. Temporal planning with preferences and time-dependent continuous costs. In *ICAPS*, volume 77, 78.

Cao, J.; Bell, K. R. W.; Coles, A. J.; and Coles, A. I. 2011. Voltage control of distribution network using an artificial intelligence planning method. In *Proceedings of the Twenty First International Conference and Exhibition on Electricity Distribution (CIRED)*.

Cashmore, M.; Fox, M.; Long, D.; and Magazzeni, D. 2016. A compilation of the full pddl+ language into smt. In *AAAI Workshop: Planning for Hybrid Systems*.

Denenberg, E., and Coles, A. 2018. Automated planning in non-linear domains for aerospace applications. In *58th Israel Annual Conference on Aerospace Sciences*.

Fox, M., and Long, D. 2002. Pddl+: Modeling continuous time dependent effects. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, volume 4, 34.

Fox, M.; Long, D.; and Halsey, K. 2004. An investigation into the expressive power of pddl2. 1. In *ECAI*, volume 16, 338.

Piotrowski, W.; Fox, M.; Long, D.; Magazzeni, D.; and Mercorio, F. 2016. *Heuristic planning for PDDL+ domains*, volume 2016-January. International Joint Conferences on Artificial Intelligence. 3213–3219.

Tierney, K.; Coles, A. J.; Coles, A.; Kroer, C.; Britt, A. M.; and Jensen, R. M. 2012. Automated planning for liner shipping fleet repositioning. In *ICAPS*, 279–287.

Tran, T. T.; Vaquero, T.; Nejat, G.; and Beck, J. C. 2017. Robots in retirement homes: Applying off-the-shelf planning and scheduling to a team of assistive robots. *Journal of Artificial Intelligence Research* 58:523–590.

Wolpert, D. H., and Macready, W. G. 1997. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* 1(1):67–82.

Acknowledgments

This work was supported by the UK Engineering and Physical Sciences Research Council (EPSRC) grant EP/P008410/1 (AI Planning with Continuous Non-Linear Change).Elad Denenberg was supported in part by the Joan and Reginald Coleman-Cohen Fund.

Appendix

```
(define (problem run-generator2)
(:domain generator2)
(:objects gen - generator tank1 tank2 tank3 -
    tank)
(:init
;(= (fuelLevel gen)  990) ; 1 tank
;(= (fuelLevel gen)  970) ; 2 tanks
(= (fuelLevel gen)  955) ; 3 tanks
;(= (fuelLevel gen) 940) ; 4 tanks
(= (capacity gen)  1000)
(available tank1)
(= (reftime tank1) 0)
(available tank2)
(= (reftime tank2) 0)
(available tank3)
(= (reftime tank3) 0) )
(:goal (generator-ran))
)
```

Figure 6: Symmetric Problems

```
(define (domain generator2)
(:requirements  :fluents  :durative-actions
    :duration-inequalities  :adl  :typing)
(:types generator tank)
(:predicates (refueling ?g - generator) (
    generator-ran) (available ?t - tank))
(:functions (fuelLevel ?g - generator) (capacity
    ?g - generator) (reftime ?t - tank) )

(:durative-action generate
:parameters (?g - generator)
:duration (= ?duration  1000)
:condition (over all (>= (fuelLevel ?g) 0))
:effect (and ( decrease (fuelLevel ?g) (* #t 1))
        (at  end (generator-ran)) ) )

(:durative-action refuel
:parameters (?g - generator ?t - tank)
:duration  (= ?duration  20)
:condition (and ;(at start (not (refueling ?g)) )
        (at  start (available ?t))
(over all (< (fuelLevel ?g) (capacity ?g))))
:effect (and (at  start (refueling ?g))
        (in     rease (reftime ?t) (* #t 1))
        (at  start (not (available ?t)))
        (at  end (not (refueling ?g))) ) )

(:process refueling1
```

```
:parameters (?g - generator ?t - tank)
:precondition (and (refueling ?g)
        (>= (reftime ?t) 5)
        (< (reftime ?t)  10) )
:effect ( increase (fuelLevel ?g) (* #t 0.5)) )

(:process refueling2
:parameters (?g - generator ?t - tank)
:precondition (and (refueling ?g)
        (>= (reftime ?t) 10)
        (< (reftime ?t)  15) )
:effect ( increase (fuelLevel ?g) (* #t 1.0)) )

(:process refueling3
:parameters (?g - generator ?t - tank)
:precondition (and (refueling ?g)
        (>= (reftime ?t) 15) )
:effect ( increase (fuelLevel ?g) (* #t 1.5)) ) )
```

Figure 7: Symmetric Intuitive Model Domain

```
(define (domain generator2)
(:requirements  :fluents  :durative-actions
    :duration-inequalities  :adl  :typing)
(:types generator tank)
(:predicates (refueling ?g - generator) (
    generator-ran) (available ?t - tank))
(:functions (fuelLevel ?g - generator) (capacity
    ?g - generator) (reftime ?t - tank) )

(:durative-action generate
:parameters (?g - generator)
:duration (= ?duration  1000)
:condition (over all (>= (fuelLevel ?g) 0))
:effect (and ( decrease (fuelLevel ?g) (* #t 1))
        (at  end (generator-ran)) ) )

(:durative-action refuel
:parameters (?g - generator ?t - tank)
:duration  (= ?duration  20)
:condition (and ;(at start (not (refueling ?g)) )
        (at  start (available ?t))
        (over all (< (fuelLevel ?g) (capacity ?g)
            )))
:effect (and (at  start (refueling ?g))
        ( increase (reftime ?t) (* #t 1))
        (at  start (not (available ?t)))
        (at  end (not (refueling ?g))) ) )

(:process refueling1
:parameters (?g - generator ?t - tank)
:precondition (and (refueling ?g)
        (>= (reftime ?t) 5)
        ;(< (reftime ?t)  10)
        )
:effect ( increase (fuelLevel ?g) (* #t 0.5)) )

(:process refueling2
:parameters (?g - generator ?t - tank)
:precondition (and (refueling ?g)
        (>= (reftime ?t) 10)
        ;(< (reftime ?t)  15)
        )
:effect (
    increase (fuelLevel ?g) (* #t 0.5));1.0))
)

(:process refueling3
:parameters (?g - generator ?t - tank)
:precondition (and (refueling ?g)
        (>= (reftime ?t) 15)
:effect (
    increase (fuelLevel ?g) (* #t 0.5));1.5))
) )
```

Figure 8: Symmetric Process “Stacking” Model Domain

```
(define (domain generator2)
(:requirements  :fluents  :durative-actions
    :duration-inequalities  :adl  :typing)
(:types generator tank)
(:predicates (refueling ?g - generator) (
    generator-ran) (available ?t - tank) (s0 ?t -
     tank) (s1 ?t - tank) (s2 ?t - tank) (s3 ?t -
     tank) (c01 ?t - tank) (c12 ?t - tank) (c23 ?
    t - tank))
(:functions (fuelLevel ?g - generator) (capacity
    ?g - generator) (reftime ?t - tank) )

(:durative-action generate
:parameters (?g - generator)
:duration (= ?duration  1000)
:condition (over all (>= (fuelLevel ?g) 0))
:effect (and ( decrease (fuelLevel ?g) (* #t 1))
(at  end (generator-ran)) ) )

(:durative-action refuel
:parameters (?g - generator ?t - tank)
:duration  (= ?duration  20)
:condition (and (at  start (available ?t))
(over all (< (fuelLevel ?g) (capacity ?g)))
(at  end (s3 ?t)) )
:effect (and (at  start (refueling ?g))
(at  start (not (available ?t)))
(at  start (s0 ?t))
(at  end (not (refueling ?g))) ) )

(:durative-action strt-refueling1
:parameters (?g - generator ?t - tank)
:duration (= ?duration 5)
:condition (and (at  start (c01 ?t))
(at  end (c12 ?t)) )
:effect (and (
    increase (fuelLevel ?g) (* #t 0.5))
(at  start (s1 ?t))
(at  end (not(s1 ?t))) ) )

(:durative-action strt-refueling2
:parameters (?g - generator ?t - tank)
:duration (= ?duration 5)
:condition (and  (at  start (c12 ?t))
(at  end (c23 ?t)) )
:effect (and ( increase (fuelLevel ?g) (* #t 1))
(at  start (s2 ?t))
(at  end (not (s2 ?t))) ) )

(:durative-action strt-refueling3
:parameters (?g - generator ?t - tank)
:duration (= ?duration 5)
:condition (and (at  start (c23 ?t)) )
:effect (and (
    increase (fuelLevel ?g) (* #t 1.5))
(at  start (s3 ?t)) ) )

(:durative-action clp01
:parameters (?g - generator ?t - tank)
:duration (= ?duration 0.15)
:condition (and (at  start (s0 ?t))
(at  end (s1 ?t)) )
:effect (and (at  start (c01 ?t))
(at  end (not (c01 ?t))) ) )

(:durative-action clp12
:parameters (?g - generator ?t - tank)
:duration (= ?duration 0.15)
:condition (and (at  start (s1 ?t))
(at  end (s2 ?t)) )
:effect (and (at  start (c12 ?t))
(at  end (not (c12 ?t))) ) )

(:durative-action clp23
:parameters (?g - generator ?t - tank)
:duration (= ?duration 0.15)
:condition (and (at  start (s2 ?t))
(at  end (s3 ?t)) )
:effect (and  (at  start (c23 ?t))
(at  end (not (c23 ?t))) ) ) )
```

Figure 9: Symmetric Clips Domain

```
(define (problem run-generator2)
(:domain generator2)
(:objects gen - generator tank1 tank2 tank3 -
    tank) ;
;(:objects gen - generator tank1 tank2 - tank)
;(:objects gen - generator tank1 - tank)
(:init
;(= (fuelLevel gen)  990) ; 1 tank
;(= (fuelLevel gen)  970) ; 2 tanks
(= (fuelLevel gen)  955) ; 3 tanks
;(= (fuelLevel gen) 940) ; 4 tanks
(= (capacity gen)  1000)
(= (last-used gen) 0)
(available tank1)
(= (reftime tank1) 0)
(= (tanknum tank1) 1)
(available tank2)
(= (reftime tank2) 0)
(= (tanknum tank2) 2)
(available tank3)
(= (reftime tank3) 0)
(= (tanknum tank3) 3)
)
(:goal (generator-ran))
)
```

Figure 10: Asymmetric Problems

```
(define (domain generator2)
(:requirements  :fluents  :durative-actions
    :duration-inequalities  :adl  :typing)
(:types generator tank)
(:predicates (refueling ?g - generator) (
    generator-ran) (available ?t - tank))
(:functions (fuelLevel ?g - generator) (capacity
    ?g - generator) (reftime ?t - tank) (
    last-used ?g - generator) (tanknum ?t - tank)
     )

(:durative-action generate
:parameters (?g - generator)
:duration (= ?duration  1000)
:condition (over all (>= (fuelLevel ?g) 0))
:effect (and ( decrease (fuelLevel ?g) (* #t 1))
        (at  end (generator-ran)) ) )

(:durative-action refuel
:parameters (?g - generator ?t - tank)
:duration  (= ?duration  20)
:condition (and (at
    start (= (last-used ?g) (- (tanknum ?t) 1) ))
        (at  start (available ?t))
        (over all (< (fuelLevel ?g) (capacity ?g)
            )))
:effect (and (at  start (refueling ?g))
        ( increase (reftime ?t) (* #t 1))
        (at  start (not (available ?t)))
        (at      end (not (refueling ?g)))
        (at  end (assign (last-used ?g) (tanknum
            ?t))) ) )

(:process refueling1
:parameters (?g - generator ?t - tank)
:precondition (and (refueling ?g)
        (>= (reftime ?t) 5)
        (< (reftime ?t)  10) )
:effect ( increase (fuelLevel ?g) (* #t 0.5)) )

(:process refueling2
:parameters (?g - generator ?t - tank)
:precondition (and (refueling ?g)
        (>= (reftime ?t) 10)
        (< (reftime ?t)  15) )
:effect ( increase (fuelLevel ?g) (* #t 1.0)) )

(:process refueling3
:parameters (?g - generator ?t - tank)
:precondition (and (refueling ?g)
        (>= (reftime ?t) 15)
:effect ( increase (fuelLevel ?g) (* #t 1.5)) ) )
```

Figure 11: Asymmetric Intuitive Model Domain

```
(define (domain generator2)
(:requirements  :fluents  :durative-actions
    :duration-inequalities  :adl  :typing)
(:types generator tank)
(:predicates (refueling ?g - generator) (
    generator-ran) (available ?t - tank))
(:functions (fuelLevel ?g - generator) (capacity
    ?g - generator) (reftime ?t - tank) (
    last-used ?g - generator) (tanknum ?t - tank)
     )

(:durative-action generate
:parameters (?g - generator)
:duration (= ?duration  1000)
:condition (over all (>= (fuelLevel ?g) 0))
:effect (and ( decrease (fuelLevel ?g) (* #t 1))
        (at  end (generator-ran)) ) )

(:durative-action refuel
:parameters (?g - generator ?t - tank)
:duration  (= ?duration  20)
:condition (and ;(at start (not (refueling ?g)) )
        (at  start (= (last-used ?g) (- (tanknum
            ?t) 1) ))
        (at  start (available ?t))
        (over all (< (fuelLevel ?g) (capacity ?g)
            )))
:effect (and (at  start (refueling ?g))
( increase (reftime ?t) (* #t 1))
        (at  start (not (available ?t)))
        (at  end (not (refueling ?g)))
        (at  end (assign (last-used ?g) (tanknum
            ?t))) ) )

(:process refueling1
:parameters (?g - generator ?t - tank)
:precondition (and (refueling ?g)
        (>= (reftime ?t) 5)
        ;(< (reftime ?t)  10)
        )
:effect ( increase (fuelLevel ?g) (* #t 0.5)) )

(:process refueling2
:parameters (?g - generator ?t - tank)
:precondition (and (refueling ?g)
        (>= (reftime ?t) 10)
        ;(< (reftime ?t)  15)
        )
:effect (
    increase (fuelLevel ?g) (* #t 0.5));1.0))
)

(:process refueling3
:parameters (?g - generator ?t - tank)
:precondition (and (refueling ?g)
(>= (reftime ?t) 15) )
:effect (
    increase (fuelLevel ?g) (* #t 0.5));1.5))
) )
```

Figure 12: Asymmetric Process “Stacking” Model Domain

```
(define (domain generator2)
(:requirements  :fluents  :durative-actions
    :duration-inequalities  :adl  :typing)
(:types generator tank)
(:predicates (refueling ?g - generator) (
    generator-ran) (available ?t - tank)
(s0 ?t - tank) (s1 ?t - tank) (s2 ?t - tank) (s3
    ?t - tank) (c01 ?t - tank)
(c12 ?t - tank) (c23 ?t - tank) )
(:functions (fuelLevel ?g - generator) (capacity
    ?g - generator)
(last-used ?g - generator) (tanknum ?t - tank))
```

```
(:durative-action generate
:parameters (?g - generator)
:duration (= ?duration  1000)
:condition (over all (>= (fuelLevel ?g) 0))
:effect (and ( decrease (fuelLevel ?g) (* #t 1))
        (at  end (generator-ran)) ) )

(:durative-action refuel
:parameters (?g - generator ?t - tank)
:duration  (= ?duration  20)
:condition (and
(at  start (available ?t))
        (at  start (= (last-used ?g) (- (tanknum
            ?t) 1) ))
        (over all (< (fuelLevel ?g) (capacity ?g)
            ))
        (at  end (s3 ?t)) )
:effect (and (at  start (refueling ?g))
        (at  start (not (available ?t)))
        (at  start (s0 ?t))
        (at  end (not (refueling ?g)))
        (at  end (assign (last-used ?g) (tanknum
            ?t))) ) )

(:durative-action strt-refueling1
:parameters (?g - generator ?t - tank)
:duration (= ?duration 5)
:condition (and (at  start (c01 ?t))
        (at  end (c12 ?t)) )
:effect (and (
    increase (fuelLevel ?g) (* #t 0.5))
        (at  start (s1 ?t))
        (at  end (not(s1 ?t))) ) )

(:durative-action strt-refueling2
:parameters (?g - generator ?t - tank)
:duration (= ?duration 5)
:condition (and (at  start (c12 ?t))
        (at  end (c23 ?t)) )
:effect (and ( increase (fuelLevel ?g) (* #t 1))
        (at  start (s2 ?t))
        (at  end (not (s2 ?t))) ) )

(:durative-action strt-refueling3
:parameters (?g - generator ?t - tank)
:duration (= ?duration 5)
:condition (and (at  start (c23 ?t)) )
:effect (and (
    increase (fuelLevel ?g) (* #t 1.5))
        (at  start (s3 ?t)) ) )

(:durative-action clp01
:parameters (?g - generator ?t - tank)
:duration (= ?duration 0.15)
:condition (and (at  start (s0 ?t))
        (at  end (s1 ?t)) )
:effect (and (at  start (c01 ?t))
        (at  end (not (c01 ?t))) ) )

(:durative-action clp12
:parameters (?g - generator ?t - tank)
:duration (= ?duration 0.15)
:condition (and (at  start (s1 ?t))
        (at  end (s2 ?t)) )
:effect (and (at  start (c12 ?t))
        (at  end (not (c12 ?t))) ) )

(:durative-action clp23
:parameters (?g - generator ?t - tank)
:duration (= ?duration 0.15)
:condition (and (at  start (s2 ?t))
        (at  end (s3 ?t)) )
:effect (and (at  start (c23 ?t))
        (at  end (not (c23 ?t))) ) ) )
```

Figure 13: Asymmetric Clips Domain