The Perils of Discrete Resource Models

William Cushing
Dept. of Comp. Sci. and Eng.
Arizona State University
Tempe, AZ 85281

David E. Smith
Intelligent Systems Division
NASA Ames Research Center
Moffett Field, CA 94035-1000

Abstract

Finding and expressing a computationally tractable abstraction of the real world, for the purpose of plan synthesis, is extremely challenging — even when the scope of inquiry is severely limited. In the case of modeling complex behavior on resources, such as fuel or battery charge, Fox and Long propose an intuitive methodology: pessimistically discretize access. This methodology is satisfactory in many situations, however, naive attempts to enforce capacity constraints are prone to failure. The difficulty is that the technique tracks a lower bound and so is inappropriate for enforcing an upper bound.

We present two extensions of the modeling technique that allow enforcing upper bounds in a principled fashion. The first idea is to discretize the resource optimistically, yielding an upper bound on the actual resource profile. The second idea is to pessimistically track a dual variable.

The conditions necessary for the naive approach to succeed are quite strong, which motivates our extensions. Nonetheless, it is desirable to simplify domain models whenever possible. We suggest modeling in the most general, least error-prone, manner possible, and subsequently optimizing by compiling in knowledge (ideally, automatically). In pursuit of this we define the domain abstraction/approximation problem in quite general terms and present our analysis as motivation for general modeling techniques and automatic domain simplification tools.

Introduction

A resource is a useful thing to have, typically the greater the quantity the better. Effectively modeling the behavior of resources is surprisingly tricky though. It is first of all difficult to precisely predict the exact behavior of the world. Even when that is possible, current planning technology does not handle change over time. Older technology, for example Zeno (Penberthy & Weld 1994), does support change over continuous intervals of time, however, for the sake of computational efficiency (among other reasons), current planners insist on "instantaneous changes". So in order to cope with a lack of knowledge about the real world, and for the sake of computational efficiency, we are faced with the problem of effectively discretizing the behavior of resources.

Fox and Long suggest a *lower-bound* paradigm for modeling resources (see Figure 1), based on the intuition (for resources) that "more is better". That is, preconditions of

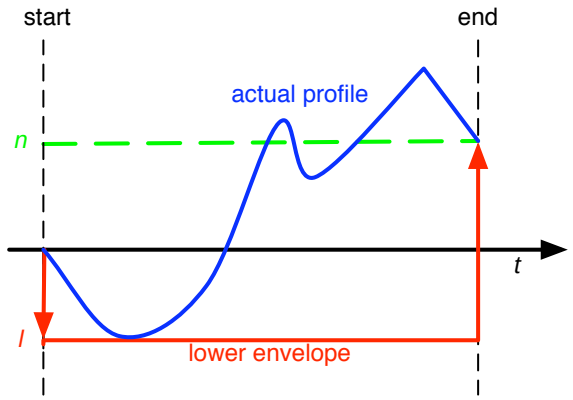


Figure 1: Modeling a lower bound (Fox & Long 2003)

actions are assumed to always be lower-bound checks on the fluent encoding the behavior of the resource. The technique is simple: "delay" all production to AT-END, and "hasten" all consumption to AT-START. This models a lower bound on the resource instead of the actual behavior: the actual resource profile will be everywhere at least as large as in the model. In other words, the model preserves correctness (since preconditions are "always" lower bounds).

However, there is a significant obvious weakness: encoding upper bounds. Typically, we are only interested in lower bounds on a resource, but it is also entirely normal for a resource to have a *capacity*. A capacity constraint is, of course, an upper bound on a resource. The naive idea is to directly enforce the capacity constraint against the model; however, the model only tracks a lower bound, so that the model may be less than the capacity while the actual behavior exceeds it. Nonetheless, the naive approach has the desired result if some special conditions apply to ones domain.

In general, the solution is to model the resource using two fluents: one for enforcing lower bounds and the other for enforcing upper bounds. We discuss two variations for supporting upper bounds, which may be understood as taking a dual in time or in state. The first idea is to explicitly track an upper bound on the resource by delaying consumption to the end and hastening production to the beginning (the op-

posite in time). The second idea is to track a lower bound on the dual of the resource: the amount of available storage (the opposite in state). The actual behavior of the dual is the negation of the primal behavior, which is discretized into a lower bound in the normal fashion.

Hastening consumption and delaying production does not alter the net usage of an action, nor, therefore, of a plan. So the major sacrifice in all of these approaches to discretization is that access to resources is slowed — a concurrent production and consumption could succeed in the real world, but the model may require delaying the consumption activity to after the production. If it happens to always be possible to delay consumptions then this apparent sacrifice is an enormous blessing: a loosely coupled planning and schedule approach could find such delayed plans very rapidly in a discretized model, and then recover the tighter plans using rescheduling techniques in a detailed model. Even when delaying consumptions is impossible, for example because concurrent consumption is necessary to avoid overfilling storage during a production, one can still recover completeness by employing a more precise discretization, or more generally, a more precise approximation. That is, one would only need an "infinitely precise discretization", i.e. a model of the true behavior, if the only solutions require perfectly synchronous activity.

Background

We stay within the general scope of PDDL and the planners which parse it, that is, a durative action based perspective on change and deterministic worlds (Fox & Long 2003). However, accurate models of the real world cannot be expected to obey any special restrictions; for example, models of the real world are allowed to have effects in the middle of actions (Smith 2003), effects as arbitrary functions of time and state (non-constant, non-linear, discontinuous, ...), and so forth. The figures define our meaning precisely enough; we limit ourselves to a short definition of *effect*:

Definition 1 (effect) *An* effect e *changes a fluent* f *over an interval of time* $I(e)$*. It may be one of two kinds of change:*

1. *An additive effect*
2. *An assignment effect*

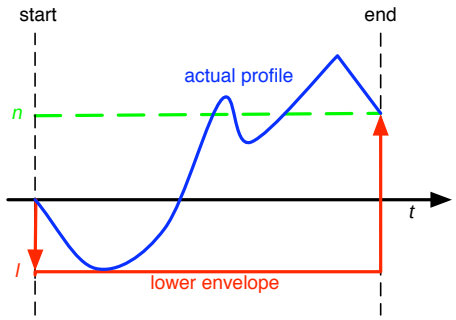
Two effects are mutually exclusive if they attempt to cause change to the same fluent at the same time and either one is an assignment effect. A set of concurrent additive effects yields the same result as applying each sequentially (the summation of all of them).

Of course it is entirely unrealistic to expect accurate models of the real world to be infinitely precise or entirely deterministic, as is suggested in the figures. All that is meant is that one has sufficient predictive ability that any further noise due to uncontrolled/unmeasured variables is insignificant for the purposes of planning. It is often the case that uncertainty remains important, for example, the duration of a flight from one city to another often exceeds the nominal value quite significantly. It is only in very controlled situations that reliable accurate predictions are possible: egg timers are manufactured and sold because the variables affecting the process of boiling an egg can be tightly controlled. In the long run, developing effective abstraction techniques for uncertain models is important, but as we will show even the very restricted problem of "deterministic" behavior on resources is complex enough to warrant in-depth consideration.

So the problem we are considering is of transforming a "true" but infeasible model (of deterministic behavior on resources) into a feasible model. Feasible is meant in a practical sense: a model is infeasible if one cannot use it to automatically synthesize plans. That could be because no implemented planners can even parse the model, or because one would never have the patience to wait for the solution. In terms of current planners, the least common denominator is that effects be:

1. Constant except at endpoints
2. Independent of state, except at those endpoints

So, for practical reasons, one must discretize effects, further getting rid of any dependence on intermediate values of the state trajectory (getting rid of integrals or differential equations, for example).



	Lower-bound
Additive	(at start (+= f_l l))
	(at end (+= f_l $(-$ n $l)$)
Assignment	(at start (= f_l l))
	(at end (= f_l n))

Figure 2: Modeling a lower bound

Capacity in Lower-bound Modeling

Discretizing effects requires knowledge of what values or ranges of values are important in achieving conditions of actions; for example, reasonable discretizations of movement and position are possible, but often one needs to be at least somewhat adaptive (so that areas with more clutter are discretized finer). We focus on the special case of resources; what distinguishes a resource from an arbitrary fluent is that the only *useful* thing about a resource is that one have more of it. So, for example, *fuel* is a resource because it would always be better to have more fuel. The *position* of a vehicle, on the other hand, is not a resource because there isn't any ordering of possible positions so that "bigger" positions are always "better". There is some subtlety in this perspective:

there can be negative interactions between resources, so that acquiring more *mobility* (i.e., acquiring less *load*) requires dumping *fuel*. This does not mean that having low *fuel* is good, but rather that consuming *fuel* can be used to increase *mobility*, which is in turn useful (because it shortens the duration of a movement action, perhaps). The ideal situation would be to have an infinite amount of weight-less fuel.

This distinction between general purpose fluents (position) and resources is perhaps interesting in its own right, but it is also immediately useful in terms of discretizing. In particular, the preconditions for action success involving a resource f are always lower bounds: $f \geq v$. Enforcing such conditions only requires tracking a lower bound on the actual behavior: figure 2 depicts the basic approach (Fox & Long 2003). There is a fluent f being effected by either an additive or assignment effect $e(t)$, which is a function of time in the real world. The final value, $e(\text{end})$, is the net change in the fluent, and the one that persists. So for sequential planning, it would be enough to discretize all changes by simplifying every effect from $e(t)$ to just "(at end ($<op>$ f $e(\text{end})$))". To allow concurrency, one ensures a lower-bound by transitioning to the minimum AT-START, and returning to the net effect AT-END.

In short, the actual behavior is always larger than the discretized lower bound, so enforcing the (lower-bound) preconditions for action success in the discretization correctly enforces these preconditions with respect to the actual behavior as well. What is not correctly enforced in this framework, however, is a capacity constraint on the resource. While action preconditions must all be lower bounds, a resource may still have a capacity: a global upper bound. The naive method is to enforce the capacity constraint against the discretized lower bound. While in general this method fails, it is sometimes possible to get away with enforcing capacity restrictions only against a lower bound. Observe that the lower bound, f_l, equals the actual behavior whenever all change has ceased. To show that the naive method succeeds, one needs to prove that, for each effect of a plan in the discretization:

1. If the lower bound is less than the capacity at the end,
2. then the actual behavior is less than the capacity over the whole duration

In particular, consider applying some effect, by itself, in a situation that results in the lower bound meeting the capacity: $f_l(\text{end}) = c$. If, as in Figure 2, the actual behavior exceeds the net change, then a discretization has no hope of correctly enforcing a capacity constraint. So a critical property that a domain must satisfy for the lower-bound approach to succeed is: *the maxima of effects must be bounded by the starting and ending values*.

Definition 2 (endpoint-bounded) *A set of effects is* endpoint-bounded *if applying them always yields a trajectory with extrema only at its endpoints.*

Discretized effects, applied in isolation or in concurrent sets, are endpoint-bounded *if the extrema of actual trajectories are bounded by the extrema of the modeled trajectories (which occur at the endpoints in a discretization).*

Observation 1 *Every discretized effect, applied to a state by itself in the model, must be* endpoint-bounded *for there to be any hope of correctly enforcing capacity constraints against it. The minimum is guaranteed for free; the important extra property is that the maximum is bounded.*

This observation is only a necessary condition for the success of the naive method: sets of concurrent effects do not inherit the property of endpoint-boundedness. In the following we consider several examples of sufficient conditions which guarantee that such sets of concurrent effects only violate capacity if the lower bound violates capacity.

Alternating Consumption and Production

A *production* action monotonically increases a fluent, and a *consumption* action monotonically decreases a fluent. Both kinds of effects are endpoint-bounded in isolation, and sets of purely one kind inherit that property (follows immediately from *monotonicity*). If, in addition, every consumer is mutually exclusive with every producer, then any solution consists of periods of purely production, purely consumption, and persistence — in each of these periods the actual behavior is endpoint-bounded, so enforcing capacity against the lower bound is sufficient.

Theorem 1 *If:*

1. *Every effect is montonic, and*
2. *Every consumption is mutually exclusive with every production*

then enforcing a capacity constraint against a discretized lower bound correctly prevents actual behavior from violating the capacity constraint.

The general situation, then, is plans that have concurrent production and consumption activities. Consider the simple example of a resource with capacity c, a producer with a net effect of $+c$, and a consumer with a net effect of $-c$. Suppose one starts off at c. Then producing and consuming over the same interval, in the discretization, transitions from c, to 0, and back to c. However, suppose (in the real world) that production and consumption happen linearly, with production 5 times as fast as consumption. Then it is clear that one would exceed the capacity if one starts the production anywhere close to the beginning of the consumption. In particular, this example demonstrates that a domain must satisfy fairly strong conditions in order for us to allow concurrent production and consumption while still enforcing a capacity constraint directly against f_l.

Slow, Cautious, and Sequential Production

We can allow concurrent production and consumption, under a number of restrictions. First of all we need that production occurs sequentially: at any given time, at most one production occurs. Second we need that each such production is *cautious*, which means that it refuses to start if it would lead to a violation of capacity without future intervention. In particular, each production of f by v has a precondition of the form: "(at start ($\leq$ (+ f_l v) c))". Finally we need that each such production is *slow* — if a consumption is executing, regardless of whether a production is occurring, the instantaneous rate of change is everywhere non-positive. That is,

producing while consuming only serves to slow down the rate at which the resource is depleted.

Then the actual trajectory is not precisely endpoint-bounded, but, it is increasing only when all consumption has ceased (because production is *slow*). If only production is occurring, then the activity is unique (because production is *sequential*) and the actual trajectory can exceed the lower-bound by at most the net effect of the single production, and this is upper-bounded by the capacity since each production is *cautious*.

Theorem 2 *If:*

1. *Every effect is monotonic, and*
2. *Every production is slower than the slowest possible concurrent consumption*
3. *Production cannot start if future consumption is required to avoid exceeding capacity*
4. *Productions are not concurrently executable*

then enforcing a capacity constraint against a discretized lower bound correctly prevents actual behavior from violating the capacity constraint.

Productions must be cautious in addition to slow because otherwise one could just delay starting any concurrent consumptions until just before a capacity-exceeding production ends. In the discretization, all of the consumption happens immediately before all of the production, so that the capacity constraint remains, erroneously, satisfied — in the actual trajectory, all but an arbitrarily tiny amount of the production has already occurred, so the capacity is already violated before the consumption even begins. By forbidding productions that can be predicted to exceed capacity without future intervention, one ensures that concurrent production and consumption have predictable relative rates, allowing exploitation of slowness to prove correctness.

Productions must be slow in addition to cautious: consider starting a quick production of the entire capacity in the middle of a long consumption of the entire capacity. At the beginning of the production, the prediction of the final amount is based on the prediction of the current amount, which is based on hastening all of the consumption to the beginning of its interval (before the production starts). So the production can proceed, in the model, despite being cautious. In the actual trajectory not all of the consumption will have in fact happened when the production ends, so the capacity will be exceeded.

Productions must be sequential in addition to all the other properties because otherwise one could start two slow capacity-filling productions concurrently, neither of which is capable of predicting the presence of the other (to notice that the summation is exceeding the capacity). If one starts a quick consumption to burn off the excess production, near the end of these two productions, the actual behavior will exceed the capacity but the discretization will not.

So this kind of model allows exploiting concurrent production and consumption to get faster plans, but only if doing so (executing production and consumption concurrently) isn't actually necessary for respecting the capacity constraint. In particular, modeling a factory/refinery/machine-shop is still difficult: one may wish to model large productions and consumptions on the same, small, tank/storage-area. In this scenario, sequential plans fail, but concurrent plans can balance the rates of filling and emptying.

Decrease + Reset

Enforcing a capacity constraint is a trivial matter if it can never be violated under any circumstances, that is, if capacity is not so much a constraint but rather an emergent phenomenon. A good example is if every production is actually a *reset*, then there is no need to check the capacity constraint because it cannot be violated.

Theorem 3 *If every additive effect has a maximum at most 0, and the maximum of any executable assignment effect does not violate the capacity, then violating capacity is impossible.*

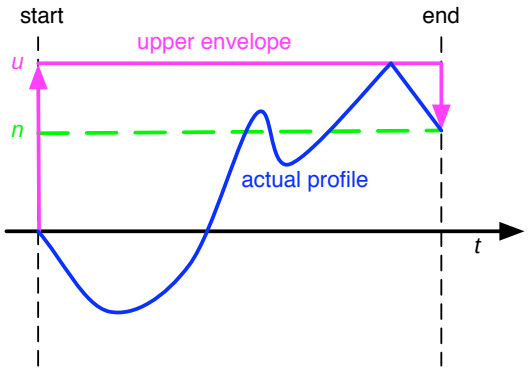
That is, if every additive effect is a consumption, then the only productions are assignment effects (i.e., *resets*), which are mutually exclusive with everything else. It is then a relatively straightforward matter to only model such non-capacity-violating productions. Capacity can only ever be violated by increasing, so clearly capacity cannot be violated in this kind of model.

In PDDL, one should take care to model such resets so that they are, in fact, mutually exclusive with other changes. This is a little more challenging than it sounds: effects are always instantaneous and effects at distinct times cannot be mutually exclusive. One solution is to introduce an abstract ternary lock for each resource, with modes *assigning, adding, constant*, with the appropriate preconditions and effects on all actions modifying the resource. A preferable solution is to explain the mutual exclusion in terms of the domain physics. For example, when production and consumption of a resource are mutually exclusive because they require the same conduit, for example a tank with one access pipe, then modeling access of the pipe introduces a mutual exclusion between production and consumption (of the material in the tank). This is preferable because the source of the mutual exclusion is correctly named; it will be much easier to adapt the model when the world changes (a separate pipe is added, another tank is added as output to the original pipe, ...).

Summary

The lower-bound methodology is not adequate for modeling resources in general, in particular resources with capacity present difficulties that straightforward extensions are unable to overcome in general, and require the domain in question to satisfy fairly strong properties when the techniques do work — properties that tend to vanish if the agent or agents gain even mild extensions to capability. Nonetheless, when such properties are known to hold, such simplifications to the model could produce some gains in performance for planning systems, if only by decreasing the time it takes to generate state descriptions. The principled approach is to model the domain using a general, less error-prone, technique and to produce a related optimized model when properties are known to hold. This of course requires having (and adhering to) a general framework: we present two such methods for modeling resources with capacity.

Upper-bound Modeling



	Upper-bound
Additive	(at start (+= f_u u))
	(at end (+= f_u (− n u)))
Assignment	(at start (= f_u u))
	(at end (= f_u n))

Figure 3: Modeling an upper bound

The key difficulty in enforcing a capacity constraint is a violation of our basic assumption about a resource: "more is always better". When producing, we must take care not to violate a capacity constraint: in this particular situation, less is better. One way to support such upper-bound conditions is to simply repeat the manner in which lower-bound conditions are supported: track an upper-bound trajectory (in addition to a lower-bound trajectory). So for any given resource f we can track the normal lower bound, f_l, as well as an upper bound, f_u. Figure 3 shows the details. Basically, every increase should happen AT-START, and every decrease should happen AT-END. Then any arbitrary upper-bound constraint, including a global capacity constraint, can be checked against f_u.

The reason endpoint-boundedness for isolated effects was important in the preceding is that we were inferring an upper bound from f_l, which is normally just a lower bound. However, at the endpoints of changes, f_l regains equality with f, thereby becoming an upper bound (as well as lower bound). It was this property that was exploited to enforce capacity, but one must have some additional means of bounding intermediate values in terms of values at endpoints if such an approach is to succeed.

If one is more careful, as we are in the figure, one need not even use the assumption that isolated actual trajectories are endpoint-bounded: f_u is always an upper bound, so it is sufficient to enforce that f_u is less than the capacity over the duration of the plan. The approach is clearer if we specialize the presentation in the following way — consider the actual trajectory induced by an additive effect and a starting value (or just the trajectory of an assignment effect). Find the starting, final, minimum, and maximum values over that interval and discretize using 4 effects:

1. (at start (− = f_l (starting − minimum)))
2. (at end (+ = f_l (final − minimum)))
3. (at start (+ = f_u (maximum − starting)))
4. (at end (− = f_u (maximum − final)))

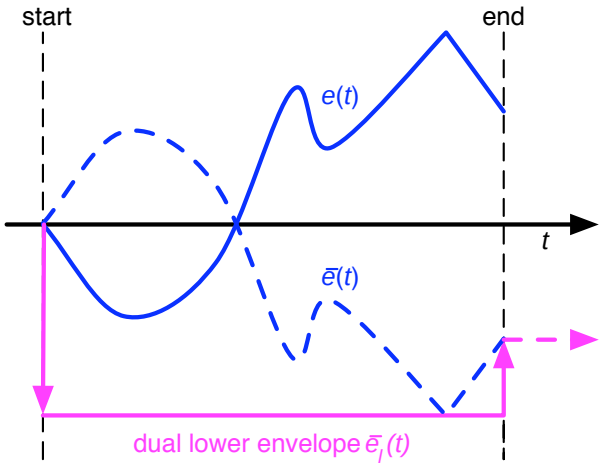
These are more or less equivalent to the effects given in the figure, but closer to the following intuition: "for a lower bound: consumption at start, production at end; for an upper bound: production at start, consumption at end". The above description adds the insight that general resource effects are simultaneously productions and consumptions. For pure production/consumption effects, half of the discrete effects simplify to addition or subtraction by 0 (no-ops).

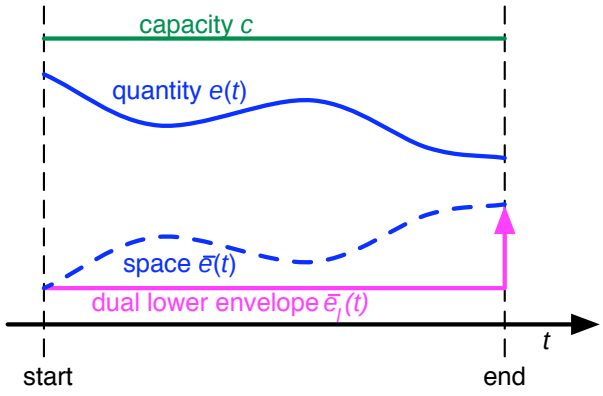
This method easily supports capacity constraints, in fact, even dynamically changing upper bounds can be verified, so that in fact this methodology is appropriate even for numeric fluents that are not really resources (like, say, the position of a robot). Viewed in this way, it is clear that what is being modeled is the uncertainty in the intermediate state of a deterministically known transition, when that uncertainty is restricted to intervals. If, for example, one wanted to state the precondition that a robot be within a certain distance of a target position (in one dimension, say), it would be enough to check that the lower and upper bounds on the uncertainty in its position were within that distance. Viewed from this perspective, one could further extend this method to relax the restriction that $f_l(t) = f_u(t) = f(t)$ whenever all change has ceased — to support the modeling of domains where one does not in fact know the actual behavior any more accurately than interval-valued uncertainty around the actual value.

Dual Resource Modeling

An alternative approach is to stick to the philosophy that it is *always* better to have more of a resource, in particular, one insists that only lower bounds are permissible. How, then, to handle capacity? The idea is that capacity is always an emergent phenomenon of the world: a summary of one's physical limits (Fox & Long 2003; Hoffmann 2002). Such limits are resources as well. For example, it is always good to have more fuel. The only reason I cannot have as much fuel as I like (on a given vehicle) is that there is insufficient *space* to store the fuel in. As another example, consider a bathtub. One way of describing the water in a bathtub is as a resource, with a capacity (say currently there are 10 gallons, and the capacity of the tub is 50 gallons). An alternative perspective is that there are two resources: the water in the tub (10 gallons), and the free space available for storing water (40 gallons). Naturally, the actual amount of resource and the actual amount of free space for storing that resource at any moment sum to a constant: the capacity of that resource. Instead of modeling such a constant directly, one can instead separately model the effects on the two resources. Then every effect is both a production and a consumption: a conversion of units of one resource into another.

In particular one can model every resource f with a lower-bound trajectory f_l (of its primal value) and a lower-bound trajectory f_d of its dual value. The dual value of a resource is just the available free space, i.e., the capacity minus the current value. Unbounded resources can just have f_d pegged





	Dual lower-bound
Additive	(at start $(- = f_d\ u)$)
	(at end $(- = f_d\ (-\ n\ u))$)
Assignment	(at start $(= f_d\ (-\ c\ u))$)
	(at end $(= f_d\ (-\ c\ n))$)

Figure 4: Modeling a dual lower bound

at infinity, or dropped from the model altogether. For example, wealth can be modeled without a dual fluent. Figure 4 presents the details of the technique. Then, enforcing a "capacity constraint" is just a matter of ensuring one never consumes more than the available space: "$(\geq f_d\ 0)$".

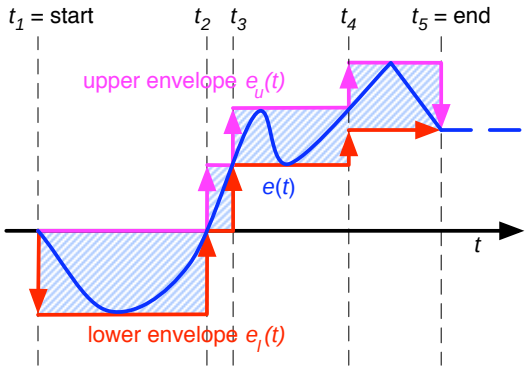
It is helpful to rewrite the effects in terms of the starting, final, maximum, and minimum values of an actual trajectory:

1. (at start $(- = f_l$ (starting $-$ minimum)))
2. (at end $(+ = f_l$ (final $-$ minimum)))
3. (at start $(- = f_d$ (maximum $-$ starting)))
4. (at end $(+ = f_d$ (maximum $-$ final)))

These are just two instances of an application of lower-bound modeling: all consumption at start, and all production at end. The subtlety is in identifying the distinction between the consumption and production of material/resource and the consumption and production of space for storing that resource.

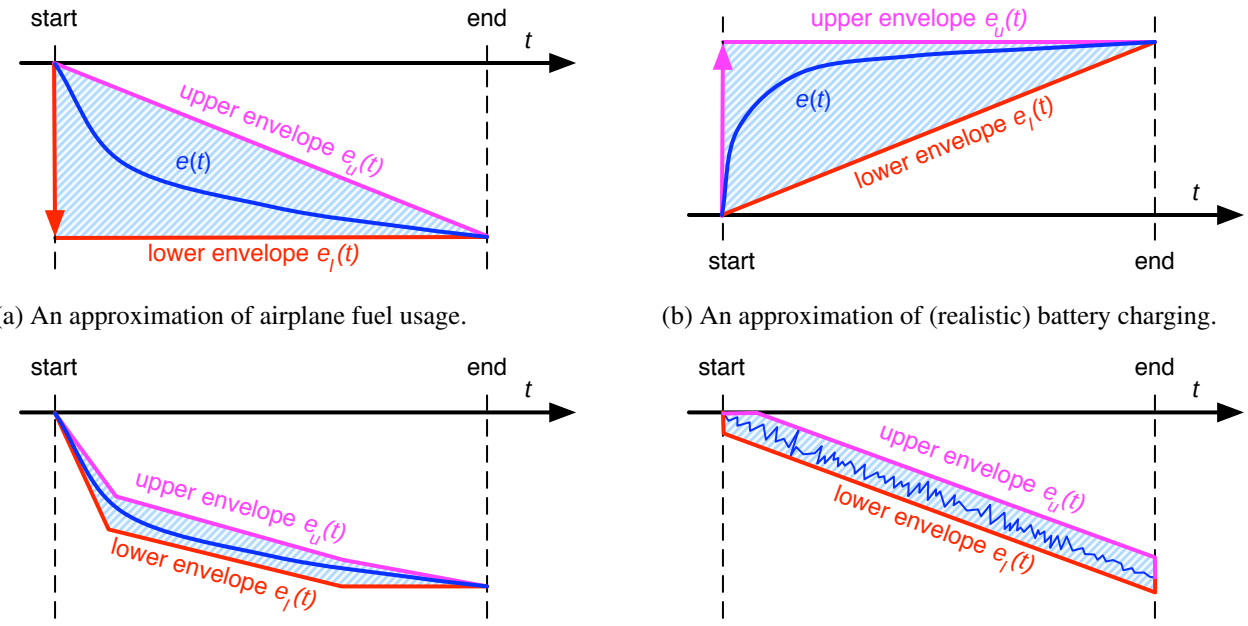
The two techniques barely differ at the syntactic level. The distinction is that the effects on f_d are the negation of the effects on f_u, and that f_d is initialized to $c - f_l$ whereas f_u is initialized to f_l. If there is in fact a global upper bound (the capacity c) then there is effectively no difference in the techniques; both methods end up tracking an interval of possible values for the actual behavior in this case. When there is no global upper bound then the method of tracking an upper bound is more powerful than tracking a dual variable. The reason is simple: the dual variable must be pegged at infinity if the primal variable can grow without bound. In particular, in modeling a one-dimensional continuosly changing position, the methodology of upper and lower bounds can enforce interval constraints on position (even equality constraints) without further restrictions (up to the precision of the discretization), whereas the dual variable approach is equivalent to just tracking a lower-bound (since the dual will be stuck at infinity).

The major advantage of the dual variable approach is that it is entirely sufficient for resources, which are a highly important class of fluents: resources usually participate directly in the solution metric (total fuel consumed, cash left on hand, so forth). Further there can be computational advantages to the restricted form: permitting only lower bounds can be helpful in computing heuristics. For example, consider the special case of boolean propositions. It is entirely normal to assume that all preconditions are purely positive, or to force the issue by adding a dual proposition to encode the negation of the primal proposition (Lifschitz 1986; Blum & Furst 1995; Hoffmann & Nebel 2001).



	Additive	Assignment
Lower	(at t_i $(+= f_l\ (-\ l_i\ l_{i-1}))$)	(at t_i $(= f_l\ l_i)$)
Upper	(at t_i $(+= f_u\ (-\ u_i\ u_{i-1}))$)	(at t_i $(= f_u\ u_i)$)
Dual	(at t_i $(-= f_d\ (-\ u_i\ u_{i-1}))$)	(at t_i $(= f_d\ (-\ c\ u_i))$)

Figure 5: Precise discretization



(a) An approximation of airplane fuel usage.

(b) An approximation of (realistic) battery charging.

(c) A tighter approximation of airplane fuel usage.

(d) A tight approximation of noisy behavior.

Figure 6: Better approximations [than pure discretization]

Quality in Modeling

Lines, Triangles, and Tubes oh my!

So far we have remained within the scope of current planners, that is, discretizing effects so that they are constant over the entire duration of actions. In fact it is not that difficult to support piecewise constant behavior either computationally or syntactically — e.g., Prottle and Zeno both support effects in the middle of actions (Little, Aberdeen, & Thiebaux 2005; Penberthy & Weld 1994). Using piecewise constant approximations of behavior allows significant improvements in modeling fidelity: compare Figures 2, 3, and 5. One can further generalize from a step function basis to any basis of functions which are syntactically and/or computationally simpler to reason with than the actual physical changes being modeled. For example, airplanes consume more fuel while climbing than while cruising: this is a concave consumption. Conversely, battery charging is convex: it is easy to add charge in the beginning, but it becomes increasingly more difficult. Note that the limits of concavity and convexity are step-functions: see Figure 6.

If we in fact knew that a change was convex (or concave), instead of bounding the change using upper and lower step functions, one could instead bound the actual trajectory inside of a triangle. In fact, supporting piecewise constant functions seems easier than supporting linear functions — so that if one can reason with lines one may as well assume that reasoning with piecewise constant functions is also possible. Taking this one step further it seems no great additional burden to add support for piecewise linear functions, as in Zeno (Penberthy & Weld 1994). This allows very precise and flexible approximation, i.e., very tight "tubes" around the actual behavior (see Figure 6). While the limit of increasingly precise piecewise linear and piecewise constant approximations are identical (both can represent arbitrary functions if allowed infinite pieces), it is clear that piecewise linear bounds can achieve much greater precision using some quota of pieces (even if one penalizes slightly for the additional cost of reasoning with lines over discrete effects).

Lines and step functions are not the only computationally tractable basis set of functions; in some applications, one could know an approximate Fourier decomposition of the behavior, where the sum of all uncovered amplitude was bounded by a small constant. The possibilities are quite diverse, but the basic point remains the same: at the level of planning, some sacrifice in modeling fidelity must be made for the sake of computational efficiency. The techniques discussed may be used to mitigate such losses up to a point, as reasoning with the suggested functions is not that much more computationally difficult than discrete effects (just more difficult to implement). Further, doing something is better than nothing — so any computationally feasible model is better than no model. Still, as compared to an oracle, such sacrifices in modeling fidelity ultimately lead to a loss in quality. Fortunately, one can often recover such losses using a hybrid planning and scheduling approach — employing scheduling against a detailed model to optimize otherwise inefficient abstract plans.

Rescheduling

Consider Figure 7 — a very simplified version of the problem faced by an aging planetary rover every day: avoid overcharging the (degrading) battery. In this example, recharging will exceed capacity twofold, so that regardless of any

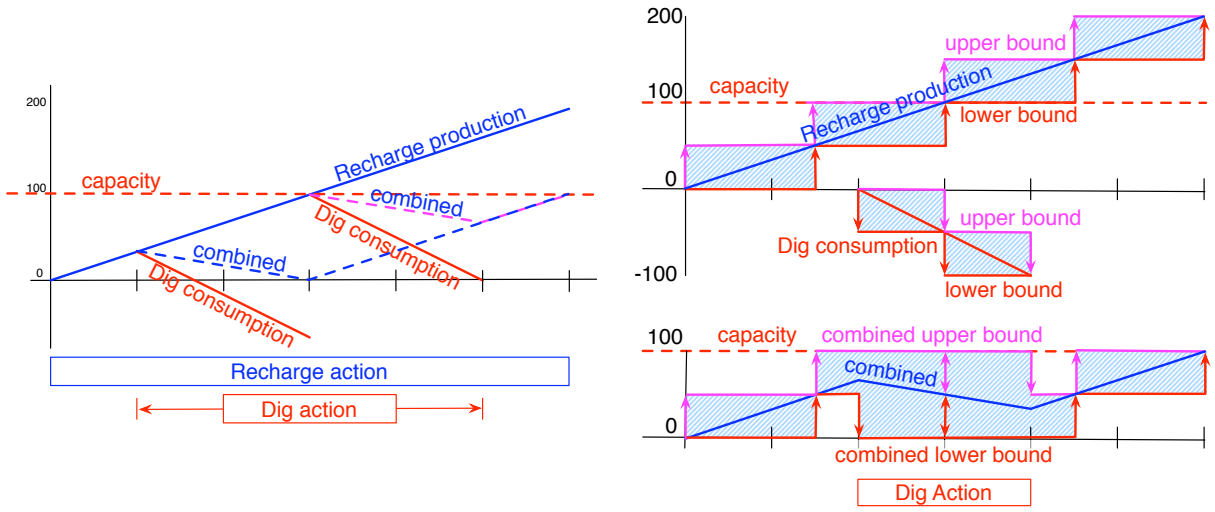


Figure 7: Discretizing at half-capacity

science objectives it is important to burn off the excess energy. The dig action does precisely that, and has a reasonably large window of opportunity within which to start in order to achieve this goal. Any discretization approach, if it can encode any solutions, will certainly miss some of the potential start times of the dig, in turn likely leading to a loss in quality (of course, this depends on exactly what the quality metric is).

Let us suppose we discretize so that every change consumes or produces half of the capacity (as in the figure). Then there is a solution — exactly one. Yet even though this solution is quite unlikely to be optimal, we can still be quite pleased with the discretization, because one could take this plan and *reschedule* it (Bäckström 1998; Do & Kambhampati 2003). That is, given the plan shown on the right of Figure 7, one could use the detailed model on the left to infer all the possible start times of the dig action. Note that performing inference in the detailed model can be feasible for the purpose of rescheduling even if the detailed model is inappropriate for planning: rescheduling is a local search (a global search would be replanning). Bäckström defines two kinds of rescheduling: deordering and reordering. The generalization of deordering to temporal planning is to consider only alternative plans with identical causal-link proofs, and reordering generalizes by allowing alternative proofs (but keeping the set of actions the same). The former kind of rescheduling is polynomial, even linear, whereas the latter is NP-complete; though even the latter could be considered feasible if the planning problem is, for example, PSPACE-complete (consider Blackbox (Kautz & Selman 1998)).

Formally, we say the domain modeling problem is the task of finding a computationally feasible abstraction of a model of the real world, in particular procedures for mapping between the real and abstract models. The inverse procedure — mapping abstract plans to concrete plans — can be quite complex, involving, among other things, *rescheduling*.

Definition 3 *An* abstraction $(\hat{D}, T, T^{-1})$ *of a domain* D *consists of a domain* $\hat{D}$ *and procedures* T *and* T^{-1} *for translating between the domains;* T *maps problems from* D *to* $\hat{D}$ *and* T^{-1} *maps solutions from* $\hat{D}$ *to* D. T^{-1} *may be nondeterministic.*
An abstraction is correct *if:*

$$\forall \mathcal{P} \in \mathit{problems}(D), \forall \hat{\pi} \in \mathit{plans}(T(\mathcal{P})), \\ \forall \pi \in T^{-1}(\hat{\pi}), \pi \in \mathit{plans}(\mathcal{P})$$

An abstraction is complete *if:*

$$\forall \mathcal{P} \in \mathit{problems}(D), \forall \pi \in \mathit{plans}(\mathcal{P}), \\ \exists \hat{\pi} \in \mathit{plans}(T(\mathcal{P})), \pi \in T^{-1}(\hat{\pi})$$

An abstraction is optimal *if any optimal abstract solution contains an optimal concrete solution in its neighborhood:*

$$\forall \mathcal{P} \in \mathit{problems}(D), \forall \hat{\pi} \in \mathit{optimal}(T(\mathcal{P})), \\ \exists \pi \in \mathit{optimal}(\mathcal{P}), \pi \in T^{-1}(\hat{\pi})$$

The discretized bounds on a resource continually return to the actual values — so the only sacrifice in the discretization is to slow down access to the resource (one must wait for the bounds to return to the true value). If such delays can be tolerated, then a hybrid approach to planning will succeed — a planner in a very coarse abstraction of the domain can solve the action selection and ordering problems, and a scheduler in a refined model of the domain can solve the dispatch problem by rescheduling the abstract plan.

Definition 4 *Let* R *denote a* rescheduling procedure*: a local search in the space of alternative schedules of its input. Let* I *be the trivial mapping:* $I(\pi) = \pi$ *for all* π. *An abstract model,* $\hat{\mathcal{D}}$ *is* complete *under rescheduling when* $(\hat{\mathcal{D}}, T, R)$ *is a* complete *abstraction, similarly,* $\hat{\mathcal{D}}$ *is* complete *(without rescheduling) when* $(\hat{\mathcal{D}}, T, I)$ *is a* complete *abstraction.*

Observation 2 *If* production *and* consumption *are mutually exclusive in* $\mathcal{D}$*, then naive enforcement of capacity in a lower-bound discretization* $\hat{\mathcal{D}}$ *is* correct *and* complete *without rescheduling.*

Theorem 4 *If* production *and* consumption *are never required to be concurrent in (solutions to problems of) $\mathcal{D}$, then forcing a mutual exclusion and naively enforcing capacity in a lower-bound discretization $\hat{\mathcal{D}}$ is* correct *and* complete *under rescheduling.*

Caveat: There exist domains which do not require concurrency and yet possess concurrent plans which cannot be sequentialized (Cushing *et al.* 2007). However, such domains are odd in that these unrecoverable concurrent plans have no distinguishing side-effects.

In many real world domains, agents producing and consuming shared resources would make reservations ahead of time to ensure there were no conflicts. That is, in such domains there is some form of global management where each producer and consumer *reserves* the appropriate amount of *space* or *material* before making changes. The technique of upper-bound and lower-bound modeling is equivalent to such conservative management of the resource, as is dual-resource and lower-bound modeling.

Observation 3 *If* production *and* consumption *are* conservatively managed *in $\mathcal{D}$, then a discretization $\hat{D}$ by upper-bound and lower-bound modeling, or dual-resource and lower-bound modeling, is* correct *and* complete *without rescheduling.*

Theorem 5 *If* production *and* consumption *are never required to be concurrent in $\mathcal{D}$, then a discretization $\hat{D}$ by upper-bound and lower-bound modeling, or dual-resource and lower-bound modeling, is* correct *and* complete *under rescheduling.*

Caveat: Again there exist odd domains which have inherently concurrent plans with no distinguishing side effects; so that while plans are lost, optimality is not.

One can relax conservativeness while preserving correctness and completeness: consider Figure 5. In this approach, each update is modeled conservatively, but the action as a whole is given multiple internal updates. In particular, the action can begin even if the action as a whole will require future intervention, just so long as it does not require future intervention before the next update. Within this kind of framework one can model individual productions and consumptions that exceed capacity (in absolute quantity), perhaps many times over. Planners that can handle only discrete changes can still find the plans requiring concurrency of such large productions and consumptions, by interleaving many smaller discrete changes (see Figure 7). Formally:

Observation 4 *There exists a sufficiently precise discretization of any domain that is* complete *under rescheduling.*

This holds with a caveat: there are domains that require infinite precision. Normally one assumes that agents do not have such perfect control, and must execute plans that would succeed equally well if start times of actions were perturbed just slightly. In discrete models of the world it is fine to allow planners to synthesize plans requiring exact simultaneity, as in Figure 7, because the very fact that one has discretized implies that there is a non-empty interval of alternative schedules of any plan one finds in the discrete model. The caveat is in the other direction: when the real model, not the discretization, requires simultaneity, one can justify refusing to deem such plans executable.

Conjecture 1 *Sampling every effect twice as frequently as the minimum window of opportunity in concrete solutions is* complete *under rescheduling. Similarly, if a discretization fails to achieve a given level of quality at a given frequency, it requires an agent with access to the real model at least half as much precision in dispatch control to do any better.*

Our first conjecture was that discretizing actions at every change of half the capacity of a resource would be complete barring any external constraints (see Figure 7). The idea is that if global bounds on the resource are the only cause for forced concurrency, one could just delay consumption until the resource is above the half-capacity mark and delay production until the resource is below the half-capacity mark. The counterexample is when neither can be delayed because both production and consumption significantly exceed the capacity, with very slightly different rates. It does appear that only highly constrained domains require greater precision than this; and in these cases it might be preferable to instead relax correctness at the planning level. Specifically, it could be quite helpful to allow the global bounds to be violated during periods of concurrent modification, as long as the bounds hold whenever change ceases. Rescheduling against the detailed model then faces the problem of picking dispatch times to put the concurrent modification within bounds, given that the net effect remains within bounds (so it is at least plausible that it is possible). This will cause greater backtracking between planner and scheduler, but allows much coarser abstraction while still allowing complicated concurrent access to the resource.

Modeling Within Proper PDDL2.1

Writing down a model that correctly enforces a capacity constraint is quite difficult in proper PDDL2.1. Richer languages support global constraints more or less directly; for example, PDDL2.2 supports derived predicates, which can be used to encode global constraints in a straightforward manner. PDDL2.1 is, however, a highly relevant language in that it represents the least common denominator of current planning technology; rather more specifically, PDDL2.1 level 3 with the additional restriction of fixed durations is the current core of supported features. In this language, one can only express constraints on action executability, so that global constraints must be compiled into local constraints. There are 3 approaches for performing this compilation:

1. Assert the constraint in every action
2. Prevent actions that cause violation
3. Prevent actions that undo violation

The first method is cumbersome, especially so in PDDL where the constraint must be given not only on each action, but in fact 3 times per action (AT-START, AT-END, OVERALL). As in the third method, this also requires that one check the constraint as part of the goal, in order to catch the case that the last action causes a violation AT-END.

For capacity constraints, the best compilation is to check the capacity right after any increase in the resource, that is, the second method. However, this is impossible in PDDL — AT-END effects occur after AT-END conditions. If one is willing to exploit the full technical details of the PDDL specification, then one can rewrite such a constraint so that it is the regression of the global constraint through the enclosing action's AT-END effects. This will be correct as far as (our understanding of) the specification is concerned, but there are fine technical details involved (concerning simultaneity) that existing planners disagree on. Moreover, many planners handle AT-END conditions poorly, e.g., by lacking effective heuristics to cope with such conditions, or by treating them as OVER-ALL conditions instead of AT-END conditions.

The third method is completely counter-intuitive: one allows a constraint to be violated. Correctness is preserved by enforcing the constraint at the goal (of every problem) and preventing actions which undo violation. In the case of capacity constraints, the problem with this approach is the strong temptation to move all the capacity constraints from the consumption actions to the production actions: doing so breaks the model. Consider, for example, the recharge action given in Figure 8 of the PDDL2.1 specification (Fox & Long 2003). This model allows executing recharge twice — even sequentially — leading to an uncaught violation of the capacity constraint (one can follow it up with one or more navigates to burn off energy if the constraint is included in the goal). On the other hand, this model can be fixed by moving the constraint from the recharge action to the navigate action (and to every other energy-consumer) and including the constraint in the goal.

Needing to repeat the constraint as part of the goal *of every problem* is of course very error-prone: problems and domains are separate files in PDDL. By and large the least evil for modeling within proper PDDL2.1 seems to be an optimized version of the first approach:

1. Add "(not done)" as a condition on every action.
2. Add "(done)" as a part of every goal (always starts false).
3. Add "(plan-end)" as an instantaneous action to the domain, which checks every global constraint and gives "(done)".
4. Check *every* global constraint AT-START and AT-END on *every* action.

This mitigates the maintenance problem between domain and problem files, and is closest to the unattainable goal of checking global constraints immediately after violations may have occurred — instead, the constraint is checked at the very next transition in state. One *could* drop the constraint from actions that cannot change the status of the constraint; however, this allows long non-goal-achieving, but executable, plans, which must then be eliminated through inference or search. Also, it makes maintenance of the domain trickier; if the constraints change, but they are all copied uniformly across actions, they can be easily replaced wholesale with the new constraints. If the treatment is non-uniform, then manual inspection is required.

Conclusion

Domain modeling is a difficult problem, even when restricted to the case of resources. We discuss the hidden pitfalls of the current approach to discretizing resource behavior. In particular, modeling capacity requires much more than a lower bound on the resource. We showed a number of conditions that can hold in the real world that allow a direct enforcement of capacity against a lower bound, but argue that the exploitation of such domain knowledge should be reserved for automatic methods. We give in-depth details on two extended forms of resource discretization that guarantee correctness without special restrictions on the domain: modeling an upper bound, or modeling a dual resource. The ideas themselves are not new, however, various existing benchmarks either fail to model a second fluent or get the details wrong (Cushing *et al.* 2007), including examples presented within the PDDL specification itself (Fox & Long 2003, Figure 8). This motivates our in-depth treatment of the matter, which we take further in considering a hopefully near-term future where alternative approaches to discretization/abstraction of complex effects can be empirically compared with one another (on planning+scheduling systems).

Acknowledgements

We thank Jeremy Frank and the anonymous reviewers for their helpful insights. Also, this research was performed with the support of the NASA SEVH Program.

References

Bäckström, C. 1998. Computational aspects of reordering plans. *JAIR* 9:99–137.

Bedrax-Weiss, T.; McGann, C.; and Ramakrishnan, S. 2003. Formalizing resources in planning. *ICAPS Workshop on PDDL*.

Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis. In *IJCAI*.

Cushing, W.; Weld, D.; Kambhampati, S.; Mausam; and Talamadupula, K. 2007. Evaluating temporal planning domains. In *ICAPS*.

Do, M. B., and Kambhampati, S. 2003. SAPA: A multi-objective metric temporal planner. *JAIR* 20:155–194.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *JAIR* 20:61–124.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Hoffmann, J. 2002. Extending FF to numerical state variables. In *ECAI*.

Kautz, H., and Selman, B. 1998. BLACKBOX: A new approach to the application of theorem proving to problem solving. In *AIPS*, 58–60.

Lifschitz, E. 1986. On the semantics of STRIPS. In *Proceedings of 1986 Workshop: Reasoning about Actions and Plans*.

Little, I.; Aberdeen, D.; and Thiebaux, S. 2005. Prottle: A probabilistic temporal planner. In *AAAI'05*.

Penberthy, S., and Weld, D. 1994. Temporal planning with continuous change. In *AAAI*.

Smith, D. E. 2003. The case for durative actions: A commentary on PDDL2.1. *JAIR* 20:149–154.