# Evaluating Temporal Planning Domains

**William Cushing**
**Subbarao Kambhampati**
**Kartik Talamadupula**
Dept. of Comp. Sci. and Eng.
Arizona State University
Tempe, AZ 85281

**Daniel S. Weld**
**Mausam**
Dept. of Comp. Sci. and Eng.
University of Washington
Seattle, WA 98195

## Abstract

The last eight years have seen dramatic progress in temporal planning as highlighted by the temporal track in the last three International Planning Competitions (IPC). However, our recent work, (Cushing *et al.* 2007), showed that most of the competition winning planners are only complete for very restricted forms of temporal planning languages that are in a sense indistinguishable from STRIPS. In this paper we consider the impact of those results on the design of benchmark temporal planning domains, and by extension, the temporal planning competition. We start by setting out to verify our speculation that the competition domains are temporally simple. This turns out to be tricky, and we develop a set of increasingly powerful analytic methods for domain analysis. Our analysis establishes that the benchmark domains are indeed inherently sequential (i.e., do not require concurrency). We suggest some real-world domains with required concurrency, and use a compilation argument to show that these are harder in the sense that they correspond to longer sequential plans. We conclude with the argument that temporal planners should be evaluated on both inherently sequential domains as well as those requiring concurrency.

## 1 Introduction

Since its inception in 1998 the bi-annual International Planning Competition (IPC) has led to dramatic improvements in planner speed. Starting in 2002, the IPC has included a track dedicated to temporal planning, whose language (PDDL 2.1.3) was specifically designed to describe domains with concurrent actions (c.f. the "Match" domain of (Fox & Long 2003)). Much to general satisfaction, the performance of temporal competition winners tracked improvements in classical planner performance, suggesting that advances discovered in the classical context were transferring smoothly to more complex problem sets.

However our recent work, (Cushing *et al.* 2007), challenges this conclusion. In a nutshell, our works divides temporal domain description languages into two categories, "temporally simple" and "temporally expressive", and show that simple languages (e.g. TGP) can not even *express* problems whose solution plans *require concurrency* (though concurrency may lead to a shorter makespan). In contrast, expressive languages (e.g., PDDL 2.1.3) *can* describe prob-

lems where concurrency is required by all solutions, but *not all* domains written in an expressive language will have problems whose solutions require concurrency. Furthermore, if a language is temporally simple (i.e. every problem always has sequential solutions), then the much stronger property that the entire language is *inherently sequential* holds — in the sense that optimal solutions of problems in these languages always correspond to linear-time rescheduling of sequential solutions. Approximately, temporally simple languages are isomorphic to STRIPS in the sense that one could augment a classical planner with such a rescheduling step at every search node. Indeed, it is now clear that SG-PLAN (Chen, Hsu, & Wah 2006), which won the temporal track in 2004 and 2006, uses this strategy.[1]

In our prior work, we also point out that most of the top entrants of the planning competition, which are based on decision epoch planning, are not complete for domains and problems that require concurrency. Based on this, we speculated that most IPC temporal track domains/problems probably do not require concurrency. If true, this would suggest that the perceived progress in temporal planning may be illusory — we are simply seeing the result of faster classical planners on inherently sequential domains.

In this paper, we investigate the impact of our results on the temporal planning domains, and by extension, the competition itself. Specifically, we tackle the following questions:

- Are the IPC temporal domains inherently sequential?

- If so, then is this a sign that real world domains rarely require concurrency?

And most importantly:

- Can one automatically analyze a PDDL 2.1.3 domain description to determine if it is inherently sequential or has required concurrency?

In this paper we answer these questions, showing that the IPC domains are, in fact, sequential; that required concurrency is both important and common in real-world domains, and providing powerful analytic tools for determining the nature of PDDL 2.1.3 domains. In addition, our analysis of

---

[1]related in a personal communication by Yixin Chen, an author of SGPLAN.

the well-studied IPC domains reveals several modeling errors which lead us to suggest better ways of describing temporal domains.

The rest of the paper is organized as follows. In the next section we provide a brief background on our prior results. Section 3 focuses on developing tests to determine the inherent sequentiality of a domain. Section 4 explains how these tools are used to analyze and establish the inherent sequentiality of the competition domains. Section 5 discusses required concurrency in the real world. Section 6 argues that required concurrency can not be easily compiled away. Section 7 discusses the impact of all this on the temporal planning competition. Section 8 presents our conclusions.

## 2. Preliminaries

The framework of (Cushing *et al.* 2007) was developed to characterize the properties of planning algorithms in the context of various domain modeling *languages*. This section reformulates a set of those insights into a form where they can more easily be used to analyze *domain characteristics*.

**Definition 1 (Temporal Gap)** *An action* has temporal gap *if*

1. *There is a condition or effect on a fluent $x$* AT START

2. *There is a condition or effect on a, possibly different, fluent $y$* AT END

*A domain* has temporal gap *if any action in the domain has temporal gap. A domain* forbids temporal gap *if none of the domain's actions has temporal gap.*

**Definition 2 (Required Concurrency)** *A problem* has required concurrency *if there exists a plan solving the problem and every such solution has concurrently executing actions.*

*A domain* has required concurrency *if there exists any problem which has* required concurrency.

We exclude timed exogenous events and deadline goals from domain definitions, since their presence gives required concurrency to almost every domain.

**Definition 3 (Causally Equivalent)** *Two plans, which solve the same problem, are* causally equivalent *if they can be shown to be correct using the same causal-link proof.*[2]

*A causal-link proof consists of a set of causal-links (one for each goal and for all conditions of every action) and an ordering relation on effects and conditions; a* causal-link *matches each condition with a supporting effect. The* ordering relation *orders the supporting effect of a causal-link before its supported condition, orders action starts before their ends, and orders every* threatening *effect to a causal link either before the link's supporting effect or after the supported condition (Chapman 1987).*

*A plan can be* shown to be correct *using a causal-link proof if the dispatch times of the plan's actions induces the ordering relation of the proof, and respects the action's durations.*

---

[2]Space considerations demand of brief treatment of causal links in temporal plans (Penberthy & Weld 1994; Younes & Simmons 2003) and their use in plan-reordering (Bäckström 1998).



Figure 1: Venn diagram of domain concurrency properties.

**Definition 4 (Inherently Sequential)** *A plan, P, is* inherently sequential *if there exists a sequential plan $P'$ solving the same problem, such that $P$ and $P'$ are causally equivalent. A problem is* inherently sequential *if every solution is* inherently sequential. *A domain is* inherently sequential *if every possible problem is* inherently sequential. *A language is* inherently sequential *if every expressible domain is* inherently sequential.

As illustrated by Figure 1, required concurrency and inherent sequentiality are not inverses of one another. There also exist *mixed* domains in which every problem can be solved by *some* sequential plan, but there may exist concurrent solutions with no sequential equivalent. For example, suppose there are two ways to take a measurement: using a self-contained portable instrument and using a larger instrument which requires concurrent operation of a generator.

Interestingly, however, our survey suggests that mixed domains are quite rare in practice — most alternative action sets (e.g. the two ways of taking a measurement) have some distinguishing side effects. For example, if the large instrument produced a higher-quality reading, then there would exist a goal (quality measurement) without a serial solution and the domain would have required concurrency.

## 3. Determining Inherent Sequentiality

Finding sufficient conditions for which a problem, domain, or language is *inherently sequential* is important; such conditions allow one to determine whether or not planning and scheduling separately is sufficient for the task. Unfortunately, the definition of inherent sequentiality is not operational, since it quantifies over all plans for all problems in a domain. Fortunately, we have an operational sufficient condition:

**Theorem 1 ((Cushing *et al.* 2007))** *If a domain forbids temporal gap, the domain is* inherently sequential.

Proving this is relatively straightforward. Consider the special case where all the actions have AT START effects. Then delaying the dispatch of every action until all earlier actions have finished executing sequentializes the plan without changing the order in which effects occur — note that an action which lacks temporal gap and has AT START effects may not have AT END effects. Therefore, the plan retains an identical state sequence, and so remains executable and goal-achieving.

While the lack of temporal gap, indeed, suffices to show inherent sequentiality, it is rarely applicable. For example, every one of the IPC temporal domains has actions with temporal gap. Clearly we need better tools. In this section we develop two complementary extensions of the temporal gap criteria, one based on analysis of sets of actions and one based on analysis of the effects and conditions of actions.

## 3.1 Action Based Framework

In general, a necessary condition for required concurrency yields a sufficient condition for inherent sequentiality. Temporal Gap is the prime example: for languages, temporal gap is necessary for required concurrency and forbidding temporal gap is sufficient for inherent sequentiality. We extend this kind of analysis to domains by strengthening the notion of temporal gap, that is, giving a broader necessary condition for required concurrency.

Consider some (executable) plan $P$, with dispatch times $t$. Let $<_P$ be the causal orderings of $P$ on threats, supporters, and conditions, so that $x <_P y$ holds if $x$ must occur before $y$. Further let $<_P^*$ denote its transitive closure, which is acyclic since $P$ is executable. Let $<_P'$ be the ordering induced by $<_P^*$ on action instances of $P$, so that $A <_P' B$ holds if $x <_P^* y$ holds for effects or conditions $x$ and $y$ belonging to $A$ and $B$, respectively. $<_P'$ may not be acyclic — $A <_P' B <_P' A$ could hold for some pair $(A, B)$ — if and only if the plan $P$ requires concurrency. That is, if $<_P'$ is acyclic, we can sequentialize $P$ by simply strengthening $<_P'$ to a total order (and picking new dispatch times). Otherwise, we have a *conflict*:

**Definition 5 (Conflicting Action Pairs)** *Let* $(A, B)$ *be a pair of action instances in a plan* $P$. *Let* $\{x, z\}$ *be conditions or effects of* $A$, *and* $\{y\}$ *by a condition or effect of* $B$. $(A, B)$ conflict *with respect to* $P$'s *causal orderings if* $x <_P^* y$ *and* $y <_P^* z$ *both hold. I.e., if* $B$ *is required to start or end in the middle of* $A$, *then* $(A, B)$ *is a* conflict.

**Lemma 1** *If a given plan* $P$ *has no* conflicts, *then that plan is* inherently sequential. *If every executable plan in a domain has no* conflicts, *then that domain is* inherently sequential.

Note that if two actions, $A$ and $B$, both lack temporal gap then their effects and conditions can only induce one order for the pair: either $A$ will completely proceed $B$ or vice versa. However, in analyzing domains, we find that most actions do have temporal gap. Nonetheless, the rest of such domains lack actions which can exploit these temporal gaps to produce plans with non-sequential structure. We strengthen the temporal gap condition for these situations, by generalizing from a local condition on actions to pairs of actions. The idea is to prevent cycles in the order induced on actions by ensuring that each action with temporal gap cannot, in fact, have a predecessor or a successor along such a cycle.

**Definition 6 (AT START-causally independent)** *An action* $A$ *is* AT START-causally independent *if there cannot be a direct causal ordering between the beginning of* $A$ *and some concurrent action:*

1. *For all* AT START *conditions* $x$, *there is no possibly concurrent action* $B$ *with an effect* $y$ *threatening* $x$
2. *For all* AT START *effects* $x$, *there is no possibly concurrent action* $B$ *with either a condition* $y$ *supported by* $x$ *nor an effect* $y$ *conflicting with* $x$.

If an action is AT START-causally independent then it may still participate in causal orderings: the restriction is that in any plan, *if the beginning of such an action must precede some condition or effect, then so too must the end of that action*. Vice versa, if an action is AT END-causally independent, then the restriction is that *if the end of that action must succeed some condition or effect, then so too must the beginning of that action*:

**Definition 7 (AT END-causally independent)** *An action* $A$ *is* AT END-causally independent *if there cannot be a direct causal ordering between the ending of* $A$ *and some concurrent action:*

1. *For all* AT END *conditions* $x$, *there is no possibly concurrent action* $B$ *with an effect* $y$ *supporting* $x$
2. *For all* AT END *effects* $x$, *there is no possibly concurrent action* $B$ *with an effect* $y$ *conflicting with* $x$.

Note that an action lacking temporal gap must either be AT START- or AT END- causally independent in the trivial sense: one of the two endpoints will lack effects and conditions to quantify over. As long as we can show that every action has at least one endpoint which is causally independent of all concurrent activity, then it is a relatively simple matter to sequentialize any plan — simply insert appropriate delays so that the order of the *other* endpoints are preserved.

**Theorem 2** *If every action in the domain is* AT START-causally independent *or* AT END-causally independent *then the domain is* inherently sequential.

**Proof sketch:** For any plan, the ordering induced by a causal-link proof on a pair of action instances is cycle-free. Suppose not: consider some pair $(A, B)$. Without loss of generality, suppose $x$ and $z$ are AT START and AT END elements of $A$, respectively, and that $y$ is an element of $B$, and that the plan causally orders these elements as: $x <_P^* y <_P^* z$. For such an ordering to hold there must be a transitive chain of direct causal orderings from $x$ to $y$ and from $y$ to $z$. Let the element closest to $x$ and $z$ in these chains be $w_s$ and $w_e$. $A$ is either AT START-causally independent, contradicting $x <_P w_s$, or AT END-causally independent, contradicting $w_e <_P z$.

These definitions are more powerful than the notion of temporal gap: applying these concepts to the *Satellite* domain allows one to show that every action is AT START-causally independent. The only problematic cases are concurrently modifying the state of power (*switch_on* and *switch_off*), or the direction the satellite points in (*turn_to* with different parameters), for which one needs to employ the global knowledge that the satellite only points at a single target or only has a single state of power at a time. Employing such global knowledge about fluents is awkward in this model, as it must be repeatedly stated within the analysis of every possibly concurrent pair of actions. A key insight is that deep properties about the causal structure of domains are naturally stated with respect to fluents, not actions. This insight motivates the finer-grained framework developed in the next section, which ultimately allows a straightforward analysis of the benchmarks.

## 3.2 Element Based Framework

In this section we aim to split up the property of inherent sequentiality into a conjunction of smaller properties over *fluents* and the effects and conditions upon them. Ideally

the result would be a necessary and sufficient condition, realistically, for the purposes of domain analysis, a reasonably tight sufficient condition is a success. In pursuit of this goal, we up-front fix the strategy for converting between concurrent and sequential versions of a plan. Specifically, the rescheduling maintains the ordering of action start times. In terms of the action-based approach, this is equivalent to showing that every action is AT END-causally independent: in this section we split up the property of AT END-causal independence over the effects and conditions on individual fluents.

Let an *element* of an action be an effect or condition of that action. Consider some plan $P$, with dispatch times $t$. Extend the domain of $t$ to map every element of each action instance to the time at which it begins to occur. Let $s$ map such elements to the start times of their containing action instances.

The technique of sequentializing by start times produces new dispatch times, $t'$, so that $t'(x) < t'(y)$ if, and only if, $s(x) < s(y)$. Two elements *conflict*, in terms of sequentializing, if the order in the concurrent plan ($t(x) < t(y)$) is causally important ($x <_P y$) and different from the new order ($t'(x) > t'(y) \Leftrightarrow s(x) > s(y)$).

**Definition 8 (Conflict)** *Let $x$ and $y$ be elements of actions instances in a plan which are causally ordered: $x <_P y$. I.e., $x$ is either an effect supporting $y$, or $y$ is an effect supporting another condition and $x$ was a threat which was demoted, or $x$ is a condition and $y$ was a threat which was promoted (Chapman 1987).*

*We say that $(x, y)$ is a* conflict *if $s(x) > s(y)$.*

If there are no conflicts, then sequentializing $P$ by start times succeeds. However, reasoning about potential conflicts in terms of pairs is awkward; thus, we rewrite the definition to assign blame to the later element.

**Definition 9 (Conflict-free)** *An element $y$ is* conflict-free, *with respect to a plan, if for every $x$, $(x, y)$ is not a* conflict.

**Definition 10 (Safe)** *An element is* safe, *if it is conflict-free in every executable plan.*

*An action is* safe *if every element of that action is* safe.

**Observation 1** *If every action in a domain is* safe, *then the domain is* inherently sequential.

*The reverse direction is not true because we have fixed the strategy for sequentializing plans.*

Note that if an element is AT START or OVER ALL it is trivially *safe*; an element is *safe* if it cannot possibly switch places with something it is causally ordered after. If $y$ is causally ordered after $x$, then the actual and starting times of $y$ (which are the same) succeed both the actual and starting times of $x$ (since $s(x) \leq t(x)$). Preserving start times then preserves that causal ordering.

So we have succeeded in reducing the problem of proving a domain to be inherently sequential to the problem of certifying that AT END elements of actions are *safe*. This is still clearly not operational. However, with a little extra knowledge about the domain one can avoid explicitly enumerating executable plans. It is helpful to split up the notion of *conflict-free* over kinds of causal orderings:

**Definition 11 (Threat-free)** *An effect $y$ is* threat-free, *with respect to plan $P$, when, for every conflicting effect $x$, if $s(y) \leq t(x) < t(y)$ then $s(x) < s(y)$*

**Definition 12 (Interaction-free)** *An effect $y$ is* interaction-free, *with respect to plan $P$, when, for every threatened condition $x$, if $s(y) \leq t(x) < t(y)$ then $s(x) < s(y)$*

**Definition 13 (Link-free)** *A condition $y$ is* link-free, *with respect to plan $P$, when, for every supporting effect $x$, if $s(y) \leq t(x) < t(y)$ then $s(x) < s(y)$*

We can now write a powerful tool for benchmark analysis:

**Lemma 2** *An effect $y$ is* safe *if, and only if, it is* threat-free *and* interaction-free *in every plan.*

*A condition $y$ is* conflict-free *if, and only if, it is* link-free *in every plan.*

**Proof Sketch:** Note that if shrinking all the durations of actions to 0 preserves the orderings of a causal-link proof, then so does sequentializing by start times (and vice versa). The orderings that must be preserved can then be split up by quantifying over demoted threats (*threat-free*), promoted threats (*interaction-free*), and causal links (*link-free*). In all cases, if $t(x) < s(y)$, then $s(x) < s(y)$ holds, and if $t(x) > t(y)$ then it would be $x$ that receives blame, not $y$. So the search for conflicts with $y$ can be restricted to the interval $[s(y), t(y))$. □

This lemma can be used to directly verify that many elements of actions in the benchmarks are *safe*. As a trivial case, consider:

**Observation 2** *In any of the benchmarks, the effect (at end (increase (total-cost) <value>)) is* safe.

**Proof Sketch:** Such an effect (in any plan) is interaction-free, and threat-free, because the benchmarks never assert conditions of any kind on "(total-cost)", nor do the benchmarks contain any effects which conflict with such *increase* effects (i.e., *decrease* or *assignment* effects). By Lemma 2, the result follows.□

Of course, Lemma 2 can also be applied in more complex situations:

**Observation 3** *The effect "(at end (calibrated ?c ?r))" of "(calibrate ?r ?c ?o ?w)" in* Rovers *is* safe.

**Proof Sketch:** This effect is *interaction-free* in any plan because there are no negative preconditions in *Rovers*. There is only one conflicting effect in the domain, "(at end (not (calibrated ?c ?r)))", and it belongs to the action *take-image*. Suppose a *take-image* terminates in the middle of a *calibrate*. Then *take-image* starts earlier, since *calibrate* has a duration of 5 and *take-image* has a duration of 7. So the effect "(at end (calibrated ?c ?r))" is *threat-free*. By Lemma 2 the result follows.□

## 4. Analyzing the Competition Domains

**Theorem 3** *Every temporal domain in IPC 2002, 2004, 2006 is* inherently sequential.

A caveat: Theorem 3 only holds once several modeling errors are repaired in the IPC domains (as explained below). Space precludes the presentation of the

proof, which consists of documenting that each element of each action of 13 domains is *safe* (full details at http://rakaposhi.eas.asu.edu/is-benchmarks.html). Instead we discuss the sufficient conditions which allow us to rapidly verify that each element is *safe*.

Lemma 2 easily supports simple useful knowledge about the fluents in a domain, for example *static* and *monotonic* (Observation 2) fluents are *safe*. In the following we design two language constructs to simplify the application of more powerful knowledge concerning fluents: *multi-valued fluents* and *resources*. For the benchmarks, the decompilation process produces only OVER ALL elements (which are trivially *safe*). In fact, if one applies a strong interpretation of the intended physical systems behind each of the benchmarks one can transform every element into an OVER ALL element. Instead we leave the few oddities as-is and employ Lemma 2 to demonstrate that they are anyways *safe*. E.g., Observation 3 demonstrates that the strange AT END effect of *calibrate* is *safe*, but it would also be quite reasonable to alter this effect into an OVER ALL effect. Of course, PDDL 2.1.3 does not allow OVER ALL effects; below, we define the meaning of OVER ALL effects on *multi-valued fluents* and *resources*.

## 4.1 Multi-Valued Fluents

A *fluent* is a time-varying variable which takes values from a *domain*. Boolean and numeric fluents are special cases, and the only kind allowed within the syntax of PDDL 2.1.3. Our analysis of the IPC domains shows that the concept of a *multi-valued fluent* is abundant, in compiled form. To support *multi-valued fluents* directly we write "==" to check equality, ":=" to give assignments, and "–>" to give changes, i.e., one row of a partial transition function. For example:

$$(== (at\ ?rover)\ ?location)$$

is a condition on a fluent encoding the position of a rover, verifying that the rover is at a particular location. The effect:

$$(:= (at\ ?rover)\ ?destination)$$

unilaterally assigns the location of the rover to a particular destination. Typically one cannot just teleport objects around; instead, it is natural to write:

$$(-> (at\ ?rover)\ ?source\ ?destination)$$

This encodes one row of a partial transition function, that is, when the effect is applied, the rover's location transitions from the source location to the destination location. If, however, the rover is not in fact at the source, then the whole action is not executable. We will have no need of the obvious generalization to partial transition functions with more than one tuple in their domain, but for the sake of completeness, one could model a toggle effect using some syntax such as:

$$(-> (status\ ?light-switch)\ (on\ off)\ (off\ on))$$

Whenever effects on multi-valued fluents are given OVER ALL, then the meaning is that the effect is *mutually exclusive* with any concurrent effect or condition upon the same fluent. So we have:

**Lemma 3** (OVER ALL **effects on multi-valued fluents**)
*"(over all (–> f x y))" and "(over all (:= f y))" are* safe.

**Proof:** "(over all (–> f x y))" is simultaneously a condition and an effect; as a condition, it is trivially *link-free* since any

supporting effect precedes the beginning of the action. As an effect, it is mutually exclusive with any effect or condition occurring in the interval of the effect, which is the same as the interval of the entire action, and so it is trivially *threat-free* and *interaction-free*. Likewise, "(over all (:= f y)" is trivially *threat-free* and *interaction-free*. By Lemma 2 the result follows.□

**(De-)Compiling Multi-valued Fluents** Of course, the entire reason PDDL 2.1.3 does not support multi-valued fluents directly is that the method for compiling them out is more or less obvious and certainly straightforward, especially in the non-temporal case:

1. Represent a multi-valued $f$ with domain $D$ using a set of boolean propositions $\{f_v \mid v \in D\}$.

2. Represent a condition "(== $f$ $v$)" as "$f_v$".

3. Represent an effect "(:= $f$ $v$)" as "(and $f_v$ (forall x - $D$ (not $f_x$)))".

4. Represent a change "(–> $f$ $x$ $y$)" as the condition "$f_x$" and the effects "(and $f_y$ (not $f_x$))".

The temporal case requires more care. Of course, instantaneous effects and conditions can be handled as in the non-temporal case by simply transferring the appropriate temporal annotation. PDDL 2.1.3 also supports OVER ALL conditions, so the only difficulty lies in OVER ALL effects. The typical solution to simulating the mutual exclusion is to slightly modify the invariant of the compilation; instead of exactly one boolean proposition from the set encoding $f$ holding at a time, the relaxation is that at most one holds. This is achieved by labeling all deletes AT START and all adds AT END:

1. "(over all (:= $f$ $v$))" becomes "(and (at start (forall $x$ - $D$ (not $f_x$)) (at end $f_v$)))".

2. "(over all (–> $f$ $x$ $y$))" becomes a condition "(at start $f_x$)" and two effects "(and (at start (not $f_x$)) (at end $f_y$))".

During the times when no value holds, $f$ is said to be *undefined*; careful inspection shows that no action which accesses or alters $f$ can start or end in any period where $f$ is *undefined*. That is, the approach correctly simulates the mutual exclusion. So, to analyze the benchmarks, one finds and reverses the results of such compilations. For example:

Compiled
```
(:durative-action navigate
:parameters (?r - rover ?s ?d - waypoint)
:duration (= ?duration 5)
:conditions (and (at start (at ?r ?s)) ... )
:effects (and (at start (not (at ?r ?s))) (at end (at ?r ?d)) ... ))
```
Decompiled
```
(:durative-action navigate
:parameters (?r - rover ?s ?d - waypoint)
:duration (= ?duration 5)
:conditions (and ... )
:effects (and (over all (–> (at ?r) ?s ?d)) ... ))
```

In the decompiled form, it is easy to show that *navigate* is *safe* (by Lemma 3). One could try to show that the AT END effects of the compiled form are *threat-free* and *interaction-free* — they are — but showing this requires no less than

|                         Correct                          |                         Benchmark                          |
|----------------------------------------------------------|------------------------------------------------------------|
| (:durative-action drop                                   | (:durative-action drop                                     |
| :parameters (?r - rover ?s - store)                      | :parameters (?r - rover ?s - store)                        |
| :duration (= ?duration 1)                                | :duration (= ?duration 1)                                  |
| :conditions (at start (store_of ?s ?r))                  | :conditions (and (at start (store_of ?s ?r)) (at start (full ?s))) |
| :effects (over all (−> (amount ?s) full empty)))         | :effects (and (at end (not (full ?s))) (at end (empty ?s)))) |

Figure 2: Models of emptying the storage of a planetary rover

inferring the invariant that at most one proposition in the set encoding the multi-valued fluent can ever hold at a time, which amounts to the same thing as decompiling the representation.

The task of decompilation is greatly aided by having a separate understanding of the physical domains being modeled; however, inferring multi-valued fluents from propositional domains can be automatically performed as well (Helmert 2006; Backstrom & Nebel 1995). An interesting situation arises when there is a discrepancy between a mechanical decompilation and the result of remodeling the domain based on a separate understanding. For example, consider Figure 2.

The benchmark version is not a result of compiling the correct version of *drop* we give. The important distinction is that the benchmark version allows a different rover with access to the same store to concurrently empty it, because over the duration of *drop*, the state of the storage *remains defined*.[3] In turn, this allows a *sample* action to start just before the end of the second such *drop* to start, with the final result being that the storage is simultaneously *full* and *empty*. The mistake is that "(at end (not (full ?s)))" should have been given AT START.

## 4.2 Resources

We follow prior work in extending PDDL 2.1.3 to represent resources, borrowing the constructs of *produce, consume,* and *use* (Bedrax-Weiss, McGann, & Ramakrishnan 2003). A *use* effect is a temporary consumption of a resource: the specified quantity is removed during the duration of the effect, but returned by the end. The opposite effect is to temporarily produce a resource: the specified quantity is temporarily available, but is removed by the end. We use *lend* to denote this kind of resource effect. For example, the effect of lighting a match can be modeled as:

(over all (lend (light)))

alternatively, "(light)" could count light sources:

(over all (lend (light) 1))

Within PDDL 2.1.3, resource effects must be compiled out into *increase* and *decrease* effects at the appropriate times. When the resource model is *pessimistic*, so that all consumption happens as soon as possible, all production happens as late as possible, all resource effects are *overall*, and there are no *lend* effects, then the key observation is that resources only serve to bound the amount of concurrency possible: there is never an advantage to starting an action in

---

[3]For that matter, the very same rover could initiate the very same drop action concurrently with *itself*. This is a "feature" of PDDL 2.1.3.

the middle of another action. This can be formally stated in the compiled form:

**Observation 4** (AT START **resource elements**) *An* AT START *lower-bound on a resource, "(at start ($\geq$ r v))", is trivially* safe. *So too is "(at start (decrease r v))" (i.e., "(over all (consume r v))"), or any other* AT START *resource element.*

**Lemma 4** (AT END **increase**) *The effect "(over all (produce r v))", that is, the effect "(at end (increase r v))" is* safe *if:*

1. *Every condition on the resource r is a lower-bound*
2. *Every possibly concurrent effect is commutative*

**Corollary 1** (OVER ALL **use**) *"(over all (use r v))" is* safe *— it is equivalent to "(and (over all (produce r v)) (over all (consume r v)))", i.e., "(and (at start (decrease r v)) (at end (increase r v)))".*

**Proof Sketch:** The concept of a causal-link proof is more complicated in the presence of numeric fluents and commutative effects. In particular, even though *decrease* and *increase* conflict, if they are commutative and the *increase* is helping to *support* a later *lower-bound* condition, then switching the order of the *increase* and the *decrease* does not alter the satisfaction of the condition. So the effect is *threat-free* by commutativity. The effect is *interaction-free* by assumption: there are no upper-bound conditions. By Lemma 2, the result follows. □

Upper-bounds on resources, i.e. capacity constraints, introduce some difficulties, but can be converted into lower-bounds by modeling the dual resource: the amount of available space (Fox & Long 2003, page 77). If both the resource and its dual are modeled pessimistically (decreases and lower-bounds AT START, increases AT END), then all of the elements on both will be *safe*, by the above arguments. Boolean propositions may be resources as well when the STRIPS bias holds: purely positive preconditions. However, while *adds* and *deletes* can be thought of *increases* and *decreases* (in a *clamping* arithmetic), they are not commutative. When a dual resource is employed, c.f. "(full ?s)" and "(empty ?s)" in Figure 2, then there are no possibly concurrent effects, so commutativity is moot. In these cases the fluents can be shown to be safe using either an argument based on *multi-valued fluents* or an argument based on *resources*. For example, a (correct) resource encoding of *drop* is:

```
(:durative-action drop
 :parameters (?r - rover ?s - store)
 :duration (= ?duration 1)
 :conditions (at start (store_of ?s ?r))
 :effects (and (over all (consume (amount ?s) 1))
          (over all (produce (space ?s) 1))))
```

which becomes unit-capacity, and can be encoded with booleans "(full ?s)" and "(empty ?s)" instead of "(amount ?s)" and "(space ?s)" (respectively), if the initial state always satisfies "(== (+ (amount ?s) (space ?s)) 1)". From this perspective the mistake in the benchmark is in putting the wrong temporal annotation on the *consume* effect.

In fact, *Rovers* contains a separate resource modeling bug in the *recharge* action. In particular, there is supposed to be a capacity of 80 units on battery energy, however, one can in fact achieve any arbitrary amount of energy. The mistake is that the modeler forgot to model the dual resource of battery energy: the available space for storing charge. This is very easy to do, consider Figure 8 and the text of page 77 in the JAIR article on PDDL 2.1.3 (Fox & Long 2003).

# 5. Required Concurrency in Real-World Domains

Having shown that all *domains* (as opposed to problems) in the previous competitions were temporally simple, we now make an attempt to understand why that is so: is it the case that required concurrency is merely a theoretical construct to illustrate the power of the domain description language and that the real world planning problems rarely exhibit required concurrency? Or is it the case that while required concurrency merits to be part of competitive domains, its absence was unintentional and was observed predominantly due to historical/legacy reasons?

We argue that the latter is a more accurate description; there exist many real world scenarios that require concurrency to solve. In our prior work we exhaustively identify the mechanisms by which required concurrency may appear: various patterns of temporal gap. These patterns include required concurrency introduced by compiling away intermediate effects, deadline goals, and exogenous events, as well as all naturally occurring required concurrency. In the following we sketch some examples of real world scenarios requiring concurrency — confirming that such domains exist, i.e., the theory of requiring concurrency does more than just draw a line in the sand.

## 5.1 Temporal Machine Shop : A New Benchmark

Recall that a Machine Shop domain involves several machines and different pieces that need to be worked upon. Different machines are capable of performing different jobs (*e.g.*, painting, rolling, shaping, *etc.*). We add the following actions to the domain.

**Baking a ceramic** Let us suppose we can make pottery: the machine shop contains a kiln and machines for working clay. To keep things simple, suppose the kiln can fire for 20 minutes at a time (and then it must be made ready again), and that baking ceramic takes, in general, less time (and can be put into and retrieved from the kiln while firing). In such a scenario we need to be performing *bake-ceramic* concurrently with *fire-kiln*. If we have multiple pieces of pottery to produce, then we can save costs by baking them altogether — a subtlety which is lost by only modeling a combined *fire-kiln-and-bake-ceramic* action. Using the notation developed in this paper for OVER ALL effects and resources, we can model these actions as:

```
(:durative-action fire-kiln
:parameters (?k - kiln)
:duration (= ?duration 20)
:effect (and (over all (lend (firing ?k)))
    (over all (-> (ready ?k) true false))
```

```
(:durative-action bake-ceramic
:parameters (?p - piece ?k - kiln)
:duration (= ?duration (bake-time ?p))
:condition (and (over all (firing ?k)) (over all (shaped ?p)))
:effect (over all (-> (baked ?p) false true)))
```

**Ventilation** In a similar vein if the task is to join two pieces using epoxy, it can be important to concurrently ventilate the area; both for the sake of the drying process as well to mitigate the danger of inhaling toxic fumes. It is important to ventilate an area while painting, as well, for exactly the same reason.

**Etc.** There are many similar scenarios in the Temporal Machine Shop domain requiring concurrency, and in similar real world domains. For instance, heating a beaker while adding chemical titrant (so the chemical reaction happens successfully), spraying cutting oil while milling, firing a strobe in conjunction with a high speed camera, and so forth.[4]

# 6. Compiling away Required Concurrency

We have seen several real-world problems demonstrating required concurrency. However, we haven't yet addressed a fundamental question: "Does required concurrency make planning for a domain any harder?" Whether the domain is inherently sequential or has required concurrency, it is still likely in the PSPACE complexity class, but that is a coarse notion of difficulty.

Inherently sequential domains can be solved by: 1) searching the space of sequential plans, 2) performing a linear-time rescheduling step at each search node (to accurately assess quality) (Cushing *et al.* 2007). One can highly optimize this approach (Chen, Hsu, & Wah 2006; Edelkamp 2003). One of the key optimizations is to perform this rescheduling incrementally. In fact, one can more or less force a STRIPS planner to perform such incremental rescheduling itself, by tracking the time at which propositions are added and deleted as part of the state information. In Nebel's notation this is a linear-time compilation scheme that preserves plan-size exactly (Nebel 2000).

On the other hand, a 1-1 compilation from PDDL 2.1.3 into STRIPS is out of the question. At the very least one needs a 1-2 compilation, that is, to split each durative action into two pieces. Note that the planner suggested in our prior work TEMPO performs a search in such a 1-2 space, but that it must also employ additional complicated scheduling inferences. In fact, the only accurate compilations we can construct require actions to advance by "unit time", i.e., an exponential blowup in plan-size.

---

[4]We thank the anonymous reviewers for their input on required concurrency in the real-world.

## 7. Lessons for the Planning Competition

In our prior work we showed that the planners winning temporal planning competition, which are based on decision epoch planning, are incomplete for any subset of PDDL 2.1.3 that is able to model domains and problems whose solutions require concurrency. Our results show that the domains used in the temporal planning, while nominally expressed in a syntactically rich language, are nevertheless incapable of admitting problems that require concurrency. Taken in combination, these paint a disappointing picture of the current temporal planning competition. There are some clear challenges being avoided by the current competition and its contenders.

The divide in the temporal planning community between the designers of PDDL 2.1.3 and the benchmark designers and competition winners is rather stark. The designers of PDDL 2.1.3 cite *required concurrency* as a motivation for the features of the language, and indeed, the only domain appearing in the specification (*light-match*) does require concurrency. As we showed, there is not a single benchmark that requires concurrency. This has lead to the ironic situation that the winner of temporal planning competition, SGPlan, outputs "no solution" when given the light-match problem used in the PDDL 2.1.3 specification!

Why is it that competition benchmarks are temporally simple? The best explanations we can come up with are: (a) most of the domains were existing classical domains simply annotated with temporal information, and thus retained the limitations of classical planning, i.e., inherent sequentiality (b) the domains were developed by researchers who were themselves working on specific temporal planners and thus had a bias towards the ones that were solvable by their own systems, and (c) all domains were written directly in PDDL 2.1.3 (as opposed to written in complex languages and compiled down) and PDDL 2.1.3 has its own limitations in directly modeling many naturally occurring phenomena (*e.g.*, intermediate effects, actions whose durations can be controlled).

There is not a dearth of planners capable of solving problems requiring concurrency: these planners search a much larger space than their competitors, and so are easily defeated in the current competition. If the temporal planning competition actually were to use benchmarks that model problems requiring concurrency, the community would be encouraged to recognize and improve upon the efficiency of the expressive planners. In Section 5, we suggested a simple benchmark requiring concurrency; we hope the community and the competition organizers will develop additional benchmarks.

## 8. Conclusions

We set out to establish our speculation that the domains used in the temporal tracks of the last three International Planning Competitions did not require concurrency and hence were equivalent to STRIPS domains. When we attempted to verify these claims, we discovered that the domain models were too complex for easy analysis. This led us to the following contributions:

- Using an "action-based" framework, we derive a stronger

sufficient condition than temporal gap (Theorem 2) which may be used to prove a domain inherently sequential.

- We refine this framework into an "element-based" methodology which proves more useful.

- Using the element-based approach, we prove that all competition domains are inherently sequential.

- We observe several design patterns in the existing domains that are error-prone. We suggest extensions to the PDDL 2.1.3 language to reduce the occurrence of such bugs and ease the verification of domains.

- We argue that required concurrency is a necessary domain construct that needs serious investigation. As a starting-point we extend the Machine Shop benchmark with new actions that yield required concurrency. We also list several other real world scenarios where required concurrency plays an important role in domain modeling.

We conclude with recommendations for the next competition, suggesting that it should incorporate a specific required-concurrency subtrack to motivate researchers to build systems which are powerful enough to handle this complex class of planning problems.

## References

Backstrom, C., and Nebel, B. 1995. Complexity results for $SAS^+$ planning. *Computational Intelligence* 11(4):625–655.

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

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

Chapman, D. 1987. Planning for conjunctive goals. *Artificial Intelligence* 32(3):333–377.

Chen, Y.; Hsu, C.; and Wah, B. 2006. Temporal planning using subgoal partitioning and resolution in SGPlan. *JAIR* to appear.

Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. 2007. When is temporal planning *really* temporal? In *IJCAI*.

Edelkamp, S. 2003. Taming numbers and duration in the model checking integrated planning system. *JAIR* 20:195–238.

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

Helmert, M. 2006. The fast downward planning system. *JAIR* 26:191–246.

Nebel, B. 2000. On the compilability and expressive power of propositional planning formalisms. *JAIR* 12:271–315.

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

Younes, H., and Simmons, R. G. 2003. VHPOP: Versatile heuristic partial order planner. *JAIR* 20:405–430.