# Complexity of Concurrent Temporal Planning

**Jussi Rintanen**
NICTA Ltd and the Australian National University
Canberra, Australia

## Abstract

We consider the problem of temporal planning in which a given goal is reached by taking a number of actions which may temporally overlap and interfere, and the interference may be essential for reaching the goals.

We formalize a general temporal planning problem, show that its plan existence problem is EXPSPACE-complete, and give conditions under which it is reducible to classical planning and is therefore only PSPACE-complete. Our results are the first to show that temporal planning can be computationally more complex than classical planning. They also show how and why a very large and important fragment of temporal PDDL is reducible to classical planning.

## Introduction

An important aspect of many real world application scenarios is time. Actions' and events' effects take place over a period of time, and the possibility of taking actions may depend on events and other actions taking place simultaneously. These are characteristic differences separating temporal planning from the classical planning problem.

- In temporal planning actions do not sequentially follow each other, but may temporally overlap and interfere. The possibility of taking an action may depend on whether some other actions are being taken.

  In classical planning actions are taken in a sequence, and the possibility of taking an action is independent of earlier (and later) actions, given the current state.

- In temporal planning the effects of an action may be a complex function of the state and other simultaneous actions (Pinto 1998), whereas in classical planning they are independent of other actions.

In this work we carry out one of the first analytic investigations on temporal planning. Temporal planning seems to be a radical departure from classical planning and the natural question is whether there is an essential difference between temporal and classical planning? Simple forms of temporal planning are reducible to classical planning (Cushing *et al.* 2007) but it is not more generally known where the border between classical and temporal planning is.

Our answer to this question is twofold. First, temporal planning can represent a wider class of problems than classical planning. This is because part of the world dynamics is represented by the actions taking place and the state variables do not completely characterize the system state unlike in classical planning. Second, the mechanisms for representing time in temporal planning have significantly more power than anything in classical planning.

At the technical level, our first result shows that temporal planning is more powerful than classical planning by being able to represent a wide class of transition systems not expressible as classical planning. Our second result shows that our formalization of temporal planning is EXPSPACE-complete, and therefore not in general reducible to classical planning (which is PSPACE-complete (Bylander 1994)). The proof of this result directly points to restrictions that decrease the complexity to PSPACE and make a reduction to classical planning possible.

The structure of the paper is as follows. In the next section we give two formalizations of temporal planning, one enumerative in which states are atomic objects and one succinct based on state variables (non-schematic, grounded). Then we show that the latter is sufficiently powerful to represent every problem instance in the former. The main section consists of an analysis of the complexity of temporal planning, showing it to be strictly more powerful than classical planning. As an application of our main results we show that many temporal planning languages, including ones used in the recent planning competitions, are polynomial-time reducible to classical planning. We conclude the paper by discussing related works and future research.

## Formal Definitions

Next we give two general formalizations of temporal planning with integer time. The first definition views states as atomic objects and makes explicit only the possible state sequences traversed when given sequences of actions are taken. The second, more practical definition expresses the same in terms of state variables so that states are associated with valuations of state variables and actions indicate how and when the values of the variables change.

The formalization with atomic states is very close to a formalization of classical planning. Apart from the possibility of concurrent actions, the formalization is almost identical.

A remarkable feature of the formalization is that action *duration*, which is an important concept in temporal planning, does not explicitly show up at this level.

## General Formulation without State Variables

**Definition 1** *A* problem instance *in temporal planning is* $\langle S, I, O, R, G \rangle$ *where*

- *$S$ is a finite set of states,*

- *$I \in S$ is* the initial state,

- *$O$ is a finite set of* actions,

- *$R : 2^O \rightarrow 2^{S \times S}$ assigns each set of actions a partial function (a binary relation) on the states and fulfills the property CUS explained below, and*

- *$G \subseteq S$ is the set of* goal states.

The function $R$ associates possible sets of joint actions a transition relation: a state $s$ has the successor state $s'$ when actions $O_1$ are taken if $(s, s') \in R(O_1)$. If there is no $(s, s') \in R(O_1)$ for a given state $s$, then it is not possible to take the actions $O_1$ in $s$.

We require that for every state the set of possible actions is closed under subsets (*CUS*): for every $s \in S$, if there is $(s, s_1) \in R(O_1)$ for some $s_1 \in S$, then for every $O_2 \subseteq O_1$ there is $(s, s_2) \in R(O_2)$ for some $s_2$. This means that taking an action is never obligatory, no matter which other actions are taken simultaneously or have been taken earlier.

An *execution* is a sequence $s_0, O_0, \ldots, s_{n-1}, O_{n-1}, s_n$ of interleaved states and sets of actions so that $s_{i+1} = R(O_i)(s_i)$ for all $i \in \{0, \ldots, n-1\}$.

## General Formulation with State Variables

We base our definitions on a finite set $A$ of state variables and a finite set $O$ of actions.

A *valuation*, representing a sequence of states, is $v : \mathbb{N} \times (A \cup O) \rightarrow \{0, 1\}$ which assigns to each state variable and each action at each $t \in \mathbb{N} = \{0, 1, \ldots\}$ the value 0 or 1. For actions, 1 means that it is taken and 0 means that it is not taken at that time point. All our results can be easily extended to multi-valued variables with any finite domain.

Formulae about the temporally extended behavior of actions make references to different time points. For this we use a temporal language $\mathcal{L}$ with operators $[i..j]\phi$ where $i, j \in \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ make a reference to time points relative to the current one: $\phi$ is true at every time in $\{i, \ldots, j\}$. We use the short-hand $[i]\phi$ for $[i..i]\phi$. Otherwise $\mathcal{L}$ is like the classical propositional logic. Literals are $a$ and $\neg a$ for variables $a \in A$. Complements $\bar{l}$ of literals $l$ are defined by $\bar{a} = \neg a$ and $\overline{\neg a} = a$. For sets $L$ of literals we define $\overline{L} = \{\bar{l} | l \in L\}$. The truth of formulae is defined as follows.

**Definition 2** *Let $v$ be a valuation, $a \in A$ any state variable, $o \in O$ any action, $\phi$, $\phi_1$ and $\phi_2$ any formulae in $\mathcal{L}$, and $i$, $j$ and $k$ integers for time points such that $j \leq k$.*

$$v \models_i a \quad \textit{iff} \begin{cases} i \geq 0 \textit{ and } v(i,a) = 1 \textit{ or} \\ i < 0 \textit{ and } v(0,a) = 1 \end{cases}$$
$$v \models_i o \quad \textit{iff } i \geq 0 \textit{ and } v(i,o) = 1$$
$$v \models_i \neg\phi \quad \textit{iff } v \not\models_i \phi$$
$$v \models_i \phi_1 \wedge \phi_2 \textit{ iff } v \models_i \phi_1 \textit{ and } v \models_i \phi_2$$
$$v \models_i \phi_1 \vee \phi_2 \textit{ iff } v \models_i \phi_1 \textit{ or } v \models_i \phi_2$$
$$v \models_i [j..k]\phi \textit{ iff } v \models_{i+h} \phi \textit{ for all } h \in \{j, \ldots, k\}$$

Before time point 0 no actions take place and the values of all state variables are the same as in time point 0.

An action $o$ is possible at a time point $i$ iff $v \models_i \phi$ where $\phi \in \mathcal{L}$ is the *precondition* of $o$.

Changes in the state of the world are described by rules $\langle p, e \rangle$ where $p$ is a formula in $\mathcal{L}$ and $e$ is a set of literals. If $v \models_i p$ and $l \in e$ then $l$ is made true at $i+1$. The formulae $p$ can refer to actions and values of state variables at arbitrary time points in the past. This makes it possible to describe any dependence of changes on a bounded length history.

Since possibility of taking an action can only depend on the past, action preconditions cannot refer to the future. Similarly, the future is a function of the past, so $p$ in a rule $\langle p, e \rangle$ refers to the past. Hence we require that $\text{times}(\phi) \subseteq \{0, -1, -2, \ldots\}$ for action preconditions $\phi$ and $\text{times}(p) \subseteq \{0, -1, -2, \ldots\}$ for rules $\langle p, e \rangle$, where

$$\begin{aligned} \text{times}(x) &= \{0\} \text{ for all } x \in A \cup O \\ \text{times}(\neg\phi) &= \text{times}(\phi) \\ \text{times}(\phi_1 \wedge \phi_2) &= \text{times}(\phi_1) \cup \text{times}(\phi_2) \\ \text{times}(\phi_1 \vee \phi_2) &= \text{times}(\phi_1) \cup \text{times}(\phi_2) \\ \text{times}([i..j]\phi) &= \{n + k | i \leq n \leq j, k \in \text{times}(\phi)\}. \end{aligned}$$

We express some rules more intuitively with time tags $[i]$ for effect literals in the rules, for example $\langle [-5.. - 4]\neg a, \{[1]a, [2]\neg b\} \rangle$. These rules can be easily reduced to the basic form of rules: if there are more than one time tag in the effect then split the rule to several, and then move the time-origin of each so that the effects refer to time point 1. The above rule hence reduces to the rules $\langle [-5..-4]\neg a, \{a\} \rangle$ and $\langle [-6.. - 5]\neg a, \{\neg b\} \rangle$. We will use this notation later in some of the proofs.

**Definition 3** *A* succinct problem instance *in temporal planning is $\langle A, I, O, R, D, G \rangle$ where*

- *$A$ is a finite set of state variables,*

- *$I$, the* initial state, *is a valuation of $A$,*

- *$O$ is a finite set of* actions,

- *$R : O \rightarrow \mathcal{L}$ assigns each action a precondition formula,*

- *$D$ is a finite set of rules, and*

- *$G \in \mathcal{L}$ is the* goal *with $\text{times}(G) \subseteq \{0, -1, -2, \ldots\}$.*

This representation is succinct in the sense that an instance of size $n$ can represent a state space of size $\Theta(2^n)$. This is not possible with the representation in Definition 1.

**Definition 4** *A plan $P : T \rightarrow 2^O$ over a set of actions $O$ is a mapping from a finite set $T \subseteq \mathbb{N}$ to sets of actions.*

281

**Definition 5** *The* execution *of a plan* $P : T \rightarrow 2^O$ *for a problem instance* $\langle A, I, O, R, D, G \rangle$ *is a sequence of states represented as a valuation which is obtained as follows, for all* $o \in O$, $a \in A$ *and* $i \geq 0$:

$v(0, a) = I(a)$,
$v(i, o) = 1$ *iff* $o \in P(i)$,
$v(i, a) = 1$ *if there is* $\langle p, e \rangle \in D$ *with* $a \in e$ *and* $v \models_{i-1} p$,
$v(i, a) = 0$ *if there is* $\langle p, e \rangle \in D$ *with* $\neg a \in e$ *and* $v \models_{i-1} p$,
$v(i, a) = v(i - 1, a)$ *if there is no* $\langle p, e \rangle \in D$ *with*
$\quad \{a, \neg a\} \cap e \neq \emptyset$ *such that* $v \models_{i-1} p$.

*This is only defined if* $v \models_i R(o)$ *for all* $i \geq 0$ *and* $o \in P(i)$ *and there are no* $\langle p, e \rangle$ *and* $\langle p', e' \rangle \in D$ *such that* $a \in e$ *and* $\neg a \in e'$ *and* $v \models_i p \wedge p'$ *for some* $i \geq 0$.

Our language can expressing durative actions, delayed effects, joint actions and actions with interfering effects.

**Example 6** In traditional temporal planning languages a central notion is that of action duration. Some actions cannot temporally overlap. We can express that two actions $o_1$ and $o_2$, respectively with durations 5 and 10, are not allowed to overlap. This is handled by the preconditions $R(o_1) = [-10..0]\neg o_2$ and $R(o_2) = [-5..0]\neg o_1$.

An action $o_1$ with duration 5 can have an immediate effect and an effect in the end of its duration. This is handled by rule $\langle o_1, \{[1]a_1, [5]a_2\} \rangle$.

An action $o_1$ can only be taken simultaneously with $o_2$ or $o_3$, not alone, as required by the precondition $R(o_1) = o_2 \vee o_3$.[1]

Two actions $o_1$ and $o_2$ have an effect if taken close to each other at most 5 time points apart. This is expressed by the rule $\langle (o_1 \wedge [-5..0]o_2) \vee (o_2 \wedge [-5..1]o_1), \{a\} \rangle$. ∎

More generally, the change in the state of the world can be an arbitrary function of the current and past values of state variables and current and past actions, over a finite past horizon. Any such function can be defined in terms of the temporal formulae and the rules for expressing change.

## Relation between Formulations

It is important that a language for a planning problem is sufficiently expressive for expressing every instance of the problem. It is easy to see that an action that changes the value of a variable to 1 if it was 0 and to 0 if it was 1 cannot be expressed in the classical STRIPS language. In this sense STRIPS is not sufficient for classical planning. Some more general classical languages can be reduced to STRIPS if we allow one action to be represented by several STRIPS actions, but this reduction is only applicable for planning problems with full observability.

In this section we will show that the formalization in Definition 3 is expressive enough for temporal planning. Most temporal planning languages do not have this property.

Our benchmark for temporal planning with discrete time is Definition 1: we will show that the language in Definition 3 can represent every problem instance representable as in Definition 1, *for an arbitrary assignment of valuations of*

---

[1]This is not possible in Definition 1 because of CUS.

---

*state variables to states* and assuming that for some $n$ any two states can be distinguished by their length $n$ histories.

In Definition 1 states can be viewed as encoding information about actions which were started earlier and are still continuing. Hence we do not want to assign each state different values of the state variables: two states might differ only with respect to the actions that are being taken, but not with respect to the state variables.

One consequence of Theorem 7 is that temporal planning problems can be formalized without state variables: the bounded horizon system dynamics can be described exhaustively by using only actions. Of course, state variables often make more compact description possible.

**Theorem 7** *Let* $\langle S, I, O, R, G \rangle$ *be a problem instance in temporal planning. Let* $A$ *be a set of state variables and* $S'$ *the set of all valuations* $v : A \rightarrow \{0, 1\}$ *of* $A$. *Let* $x : S \rightarrow S'$ *be an assignment of valuations to the states.*

*Let* $n$ *be the length of the history needed for uniquely identifying a state in* $S$. *Then there is a succinct problem instance* $\langle A, x(I), O, R', D, G' \rangle$ *that has exactly the same set of plans and associated executions.*

*Proof:* Brief sketch: The proof is based on the possibility of expressing any set of possible histories $n$ steps back as a temporal formula. This way the preconditions of actions in $\langle A, x(I), O, R', D, G' \rangle$ can be made to correspond exactly the situations in which the actions are possible in $\langle S, I, O, R, G \rangle$ (also observing which other actions are simultaneously taken), the actions can be made to have exactly the desired effects, and the goal states can be exactly expressed as temporal formulae. □

## Computational Complexity

The main results of this work are a simulation of deterministic Turing machines (DTM) with an exponential space bound by a temporal planning problem, showing the EXPSPACE-hardness of the problem, and a solution of the temporal planning problem by a nondeterministic Turing machine (NDTM) with an exponential space bound, showing its membership in EXPSPACE.

**Definition 8** *A* nondeterministic Turing machine *is a tuple* $\langle \Sigma, Q, \delta, q_0, g \rangle$ *where*

- $Q$ *is a finite set of states (the internal states),*
- $\Sigma$ *is a finite alphabet (the contents of tape cells),*
- $\delta$ *is a transition function* $\delta : Q \times \Sigma \cup \{|, \square\} \rightarrow 2^{\Sigma \cup \{|\} \times Q \times \{L, N, R\}}$,
- $q_0$ *is the initial state, and*
- $g : Q \rightarrow \{\exists, accept, reject\}$ *is a labeling of the states.*

Configurations of an NDTM consist of a working tape (a sequence of cells each containing a symbol in $\Sigma$), the location of the R/W head on the working tape, and an internal state in $Q$. At each computation step an NDTM makes one of the possible transitions as represented by $\delta$: depending on the current state and the symbol in the current cell, write a

symbol to the current cell, go to a new state, and move the R/W head to the left, right or don't move it.

The end-of-tape symbol | and the blank symbol $\square$ in the definition of $\delta$ respectively refer to the beginning of the tape and to the unwritten part of the tape. It is required that $s = |$ and $m = R$ for all $\langle s, q', m \rangle \in \delta(q, |)$ for any $q \in Q$, that is, at the left end of the tape the movement is always to the right and the end-of-tape symbol | may not be changed. For $s \in \Sigma$ we restrict $s'$ in $\langle s', q', m \rangle \in \delta(q, s)$ to $s' \in \Sigma$ which means that | can be written only when the R/W head is on the end-of-tape symbol. The NDTM computation terminates upon reaching a state $q \in Q$ such that $g(q) \in \{\text{accept}, \text{reject}\}$.

We use the notation $\sigma^i$ for the $i$th symbol of the string $\sigma$. The index of the first (leftmost) element is 1.

A deterministic Turing machine (DTM) is an NDTM with $|\delta(q, s)| = 1$ for all $q \in Q$ and $s \in \Sigma \cup \{|, \square\}$.

PSPACE is the class of decision problems that are solvable by deterministic Turing machines that use a number of tape cells bounded by a polynomial on the input length $n$ for all but a finite number of input strings. Hence no configuration in the computation of the Turing machine has more than a polynomial number of non-blank tape cells. Analogously to PSPACE, EXPSPACE is the class of decision problems with an exponential space bound.

A problem $L$ is *C-hard* if for all problems $L' \in C$ there is a function $f_{L'}$ that can be computed in polynomial time on the size of its input and $f_{L'}(x) \in L$ if and only if $x \in L'$. A problem is *C-complete* if it belongs to C and is C-hard.

## Plan Length

Unlike in the classical planning problem where only the current state and not the past is relevant, in temporal planning the consequences of actions and the possibility of taking them depends on the past. References to past time points $[i]$ in the rules and action preconditions makes it possible to refer to exponentially far in the past: exponentially in the size of the problem description assuming that the constants $i$ in $[i]$ are represented in binary (or any other base allowing a logarithmic representation of integers.)

Hence the successor of a state is determined by the current state and an exponential number of past states, which yields a doubly exponential upper bound $2^{2^n}$ on the number of different situations, which is in stark contrast with the $2^n$ upper bound for classical planning. We show that a plan may indeed need to have this many actions.

**Theorem 9** *For succinct problem instances of size $\mathcal{O}(n)$, the length of a shortest plan may be of the order $2^{2^n}$.*

*Proof:* The proof is based on forcing the plan to go through a sequence of $2^{2^n} \cdot 2^n$ states which consists of $2^{2^n}$ blocks of $2^n$ states, representing an increasing sequence of $2^n$ bit binary numbers. Least significant bits come first.

The construction uses the temporal operator $[2^n]B$ for referring to past values of $B$, where $n$ is proportional to the size of the problem instance. Let $m = 2^n$ be the number of bits in the binary number which is being incremented.

The problem instance is $\langle A, I, O, R, D, G \rangle$ where:

- $A = \{S, F, B, C\}$ where $S$ is true in the initial state only, $F$ is true in states corresponding to the least significant bit, $B$ is the current bit, and $C$ is the carry.

- $I(S) = 1, I(F) = 1, I(B) = 0, I(C) = 0$

- $O = \emptyset$

- $R = \emptyset$

- The rules describing the system dynamics are as follows.

$$D = \{ \langle S, \{\neg S\} \rangle, \ \langle F, \{\neg F\} \rangle, \\ \langle [-m](F \wedge [1]\neg S), \{[0]F\} \rangle, \\ \langle [-m+1](F \wedge [1]\neg S), \{[0]C\} \rangle, \\ \langle [-1]C \wedge [-m]B, \{[0]\neg B\} \rangle, \\ \langle [-1]C \wedge [-m]\neg B, \{[0]B, [0]\neg C\} \rangle, \\ \langle [-1]\neg C \wedge [-m]B, \{[0]B\} \rangle, \\ \langle [-1]\neg C \wedge [-m]\neg B, \{[0]\neg B\} \rangle \}$$

The first rule makes $S$ false at the time point 1. This variable is used by the third rule which, together with the second rule, makes $F$ true exactly in those states which correspond to the least significant bit. The fourth rule detects that the current binary number has been processed and the next follows: $C$ is set to 1. The fifth and the sixth rule increment the binary number by one: this is by replacing some of the least significant bits 011..11 by 100..00 (there is a 0 and zero or more 1s). The carry $C$ indicates whether we are changing the current 1s to 0s, and after encountering the first 0 it is turned to 1 and $C$ is made 0. The last two rules handle bits that don't change.

- $G = [-m+1..0]B$

The values of $B$ at time points $km$ to $km+m-1$ for any $k \in \{0, \ldots, 2^{2^n} - 1\}$ is the binary representation of $k$.

We can modify the problem instance so that the only action in $O = \{o\}$ has to be taken at every time point so that the plan consists of $2^{2^n} 2^n - 1$ actions. $\qquad\square$

The doubly exponential plan length is based on the possibility of making references to state variables' values at individual time points exponentially far in the past. Essentially, the current situation consists of the current values of the state variables and also exponentially many of their past values.

If the past time horizon is restricted to be polynomial in the problem size (for example if a *unary* encoding of $i$ in $[i]$ is used), or if no references to an exponential number of time points is made individually, only collectively, then the plan lengths are only exponential.

One such restriction is that all past references have the form $[i..0]\phi$ with $i \leq 0$. The truth of $[i..0]\phi$ is determined by the number of time points since $\phi$ was false the last time. This reduces the information in the current situation to be only polynomial in the size of the problem instance. This restriction still allows expressing constraints that for example forbid or require overlapping of two actions.

Actions having delayed effects arbitrarily far in the future can also be allowed by using a slightly more complex combination of constraints: an action with delayed effects may not overlap with another instance of itself and the effects are conditional on the past only in terms of formulae $[i]o$

and $[i..0]\phi$. In this case the delay and keeping track of the truth of $[i]o$ and $[i..0]\phi$ can again be implemented with only a polynomial number of counters. An exponential number of counters is needed if an exponential number of actions can be active simultaneously. This condition is equivalent to an action having at different time points an exponential number of instances which overlap.

The restrictions which lead to $\mathcal{O}(2^n)$ plan length also lead to PSPACE-membership of the plan existence problem. In these cases temporal planning is not more complex than classical planning. And indeed, under these restrictions the temporal planning problem is reducible to classical planning. We give a reduction from temporal to classical planning later in the connection with temporal PDDL which is closely related to a fragment of our planning language.

## Difficulty of Planning

In the general case, with the possibility of doubly exponentially long plans, the plan existence problem for temporal planning is harder than for classical planning. The proof is based on an idea similar to that of the proof of Theorem 9.

**Theorem 10** *The problem of testing whether a succinct problem instance has a plan is EXPSPACE-hard.*

*Proof:* We give a reduction of the halting problem of deterministic Turing machines (DTM) with an exponential space-bound to the plan existence question. The reduction can be done in polynomial time.

The key idea in the proof is the representation of Turing machine configurations as exponentially long state sequences, with each tape cell represented by one state. Let $n$ be the length of the input and $m = e(n)$ the exponential space bound. States at time points 0 to $e(n)-1$ represent the initial configuration, states at time points $e(n)$ to $2e(n) - 1$ represent the next configuration, and so on. Previous contents of a cell can be accessed by $[-e(n)]$.

The state variable $H$ indicates the location of the R/W head. The state variable $S$ is true at time 0 but not later. The state variable $F$ indicates a violation of the space bound.

State variables $q \in Q$ refer to the current TM state, and they are only accessed at time points where $H$ is true, corresponding to the R/W head location. State variables $a \in \Sigma \cup \{|, \square\}$ indicate tape cell contents.

The idea of the reduction is by the following Turing machine execution in which the transition between the first two configurations ($m = 5$) writes $b$, moves the R/W head to the right, and changes the state from $q_0$ to $q_1$.

| | \multicolumn{5}{c}{1st configuration} | | | | | \multicolumn{5}{c}{2nd configuration} | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... |
| $S$ | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| $F$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| $H$ | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| $a$ | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| $b$ | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | |
| $\square$ | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | |
| $|$ | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | |
| $q_0$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | |
| $q_1$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | |

Next we describe the DTM simulation in detail.

$$
\begin{aligned}
I(S) &= 1 & I(F) &= 0 \\
I(H) &= 0 & I(|) &= 1 \\
I(a) &= 0 \text{ for all } a \in \Sigma \cup \{\square\} & I(q_0) &= 1 \\
I(q) &= 0 \text{ for all } q \in Q \backslash \{q_0\}
\end{aligned}
$$

The goal is $G = \bigvee_{q \in Q, g(q) = accept} q \wedge \neg F$ which says that the current configuration is an accepting configuration and the space bound has not been violated. The initial configuration is generated by the following rules.

$$
\begin{aligned}
&\langle S, \{\neg S, \sigma^1, \neg|, H\} \rangle \\
&\langle S \wedge [1] \neg S, \{[i]\sigma^i, [i]\neg H\} \rangle \text{ for all } i \in \{2, \ldots, n\} \\
&\langle S \wedge [1] \neg S, \{[i]\neg \sigma^{i-1}\} \rangle \text{ for all } i \in \{2, \ldots, n\} \text{ st. } \sigma^i \neq \sigma^{i-1} \\
&\langle S \wedge [1] \neg S, \{[n+1]\square, [n+1]\neg \sigma^n\} \rangle
\end{aligned}
$$

The first rule says what the first input symbol on the tape is and indicates that the R/W head is on the corresponding cell. The second and the third rules represent the rest of the input string. The fourth rule represents the rest of the working tape with blank symbols $\square$ in it.

For every $q \in Q$ and $a \in \Sigma \cup \{|, \square\}$ with $\delta(q, a) = \langle a', q', d \rangle$ the following rules simulate the movement of the R/W head and the state transition of the TM.

$$
\begin{aligned}
&\langle [-m+2](H \wedge a \wedge q), \{H, q'\} \cup \overline{Q \backslash \{q'\}} \rangle \text{ if } d = L \\
&\langle [-m+1](H \wedge a \wedge q), \{\neg H\} \rangle \text{ if } d = L \\
&\langle [-m](H \wedge a \wedge q), \{H, q'\} \cup \overline{Q \backslash \{q'\}} \rangle \text{ if } d = R \\
&\langle [-m-1](H \wedge a \wedge q), \{\neg H\} \rangle \text{ if } d = R \\
&\langle [-m+1](H \wedge a \wedge q), \{H, q'\} \cup \overline{Q \backslash \{q'\}} \rangle \text{ if } d = N \\
&\langle [-m](H \wedge a \wedge q), \{\neg H\} \rangle \text{ if } d = N
\end{aligned}
$$

The next rules respectively simulate writing a symbol to the current cell and detect violations of the space-bound.

$$
\begin{aligned}
&\langle [-m+1](H \wedge a \wedge q), \{a'\} \cup \overline{\Sigma \cup \{\square, |\} \backslash \{a'\}} \rangle \\
&\langle [-m+1](H \wedge a \wedge q) \wedge [-m+2]|, \{F\} \rangle \text{ if } d = R
\end{aligned}
$$

Additionally, two rules say that the contents of tape cells which are not under the R/W head do not change: for all $a \in \Sigma \cup \{\square, |\}$ we have

$$
\begin{aligned}
&\langle [-m+1](a \wedge \neg H), \{a\} \rangle \\
&\langle [-m+1](\neg a \wedge \neg H), \{\neg a\} \rangle.
\end{aligned}
$$

It is easy to verify that the change between two consecutive blocks of $m$ states corresponds to a transition of the Turing machine, and therefore the simulation is faithful. $\square$

That shortest plans cannot be longer than $2^{2^n}$ can be shown by an argument similar to that for the $2^n$ upper bound for classical planning. Assume there are more than $2^{2^n}$ states in the execution of a shortest plan which reaches the goals. Then there must be time points $i$ and $j$ so that their histories $2^n$ steps back are the same. Now a shorter execution and a shorter plan can be constructed by skipping everything from $i+1$ until $j$, which contradicts the assumption. Hence shortest plans have length $\mathcal{O}(2^{2^n})$.

**Theorem 11** *The problem of testing whether a succinct problem instance has a plan is in EXPSPACE.*

*Proof:* Sketch: We show that the problem can be solved by a NDTM with an exponential space bound, and then use the fact that EXPSPACE=NEXPSPACE.

The NDTM starts from the initial state and guesses a sequence of $2^{2^n}$ states. For counting the number of states encountered so far only $\mathcal{O}(2^n)$ space is needed. For guaranteeing that the state sequence corresponds to a plan execution only an exponential size $n$ window of the state sequence has to be maintained. Here $n$ is the maximum $n$ such that $[-n-1..m]$ occurs in the problem description.

The NDTM accepts when reaching a time point $i$ such that $v \models_i G$ and rejects when the counter exceeds $2^{2^n}$.  $\square$

## Implications to Temporal PDDL

The factors decreasing the complexity to PSPACE and reducing plan lengths to $\mathcal{O}(2^n)$ directly imply the reducibility of large fragments of existing temporal planning languages to classical planning.

A main difference between our language and temporal PDDL (Fox & Long 2003) is that in the latter each action has a duration, change can only occur in the starting and ending points of actions, and the possibility of taking an action is determined by facts at its starting and ending points as well as during the action excluding the end points.

The main result of this section is a reduction from a large fragment of temporal PDDL to classical planning. The fragment, which we call discrete temporal PDDL, is characterized by the following restrictions.

- We ignore continuous change and continuous state variables because they easily lead to unsolvability (Helmert 2002) and are orthogonal to the temporal questions.

- No action can overlap with another instance of itself. This is necessary so that only one counter for each action is needed. Unrestricted overlapping would require an exponential number of counters. The importance of this property has never been understood before.

- Action durations are constant, i.e. independent of the state in which the action is taken. This restriction could be relaxed and we could allow a finite number of different durations for every action, but for simplicity of representation we restrict to constant durations.

Further, we use a discrete time model. PDDL under the above restrictions can always be discretized: there is a plan iff there is a plan over integer time points only (under a suitable choice of unit time.)

### Classical Target Language

Our target language is a classical planning language with some syntactic sugar which is all reducible to standard STRIPS planning in polynomial time and with only a relatively small polynomial size increase, even when the variables represent exponentially high integer values.

We consider sets $A$ of state variables that take their values from a finite domain of integer values $D = \{-1, 0, 1, \ldots, m\}$. Let $s : A \to D$ be a state. We use the following atomic expressions about the values of state variables. They can be combined with the logical connectives.

- $a$ means $s(a) \neq 0$ (some of the variables are used like Boolean state variables: $a$ means the value is non-zero and $\neg a$ means that it is zero.)

- $a < k$ for an integer constant $k$ means $s(a) < k$.

Based on the comparisons with $<$ we can define syntactic shorthands: $a \leq k$ means $a < k+1$, $a > k$ means $\neg(a \leq k)$, and $a \geq k$ means $\neg(a < k)$.

Effects are defined as follows.

- $a := b+k$ means that $a$ is assigned the value $b+k$ where $b$ is a state variable and $k$ is an integer constant. Overflows and underflows result in the maximum and minimum values of the variable's domain, respectively. We use similarly assignments of constants $a := k$.

- $a$ means $a := 1$, and $\neg a$ means $a := 0$.

A classical action $\langle p, e \rangle$ consists of a precondition $p$ (a formula) and an effect $e$ which is a set of pairs $c \rhd d$ where $c$ is a formula and $d$ is an atomic effect or a set of atomic effects. Each member of $e$ expresses a conditional effect: if $c$ is true then execute $d$.

### The Reduction

The idea of the reduction is simple: all the temporal aspects of PDDL are reduced to counters for time points. The reduction uses three classes of classical actions.

- Actions for indicating which temporal actions start at the current time point.

- One action for executing the *at start* effects of all actions starting at the current time point and the *at end* effects of all actions ending at the current time point. This action is executable only if the indicated actions are indeed executable: they do not conflict with each other or active actions that were started at earlier time points.

- Actions for progressing time: changing the values of all relevant counters corresponding to the time passing until a new temporal action is taken or until the end point of an active temporal action is reached.

Let $O = \{o_1, \ldots, o_m\}$ be the temporal actions. For a temporal action $o \in O$, let $AS(o)$ be the starting precondition (at start), $AE(o)$ be the end precondition (at end), and $OA(o)$ be the invariant condition (over all). These are all formulae. Let $SE(o)$ be the starting effect (at start) and $EE(o)$ the end effect (at end). These are sets of atomic effects. Let $DU(o)$ be the action's duration (an integer).

We use state variables $c_1, \ldots, c_m$ for counting the time remaining before an action ends. Their domain is from $-1$ to the maximum duration of any of the actions. When an action is started, its counter is set to its duration. The counter is decremented as time progresses. We use (Boolean) state variables $a_1, \ldots, a_n$ for representing the state variables of the original temporal problem. Additionally, we have the variable $F$ for indicating that the actions that progress time are free to execute (there are no action effects waiting to be executed at the current time point).

In the initial state all the counters $c_i$ have value $-1$, $F$ has value *false*, and the remaining state variables have the same initial values as in the original temporal problem.

The goal formula is the original goal formula.

Actions for indicating the start of temporal actions simply set the corresponding counter to the action's duration. The action's precondition guarantees that another instance of the action is not already active.

$$\langle c_i = -1, \{\top \rhd c_i := DU(o_i), \top \rhd \neg F\}\rangle \text{ for action } i$$

After zero or more actions have been activated in the current time point, their executability is tested, their *at start* effects are executed, *at end* conditions of actions ending at the current time point are tested, and the *at end* effects of earlier actions ending in the current time point are executed. All this is taken care of by one action.

- The precondition tests that the executability conditions of the current and earlier actions are not violated.

  The precondition consists of the following tests for each of the original temporal actions $o_i$.

  If the action's counter equals 0 then its *at end* condition must be true and no variable in it is changed by another action's *at start* or *at end* effect at the current time point: $(c_i = 0) \rightarrow (AE(o_i) \wedge \phi)$. Here $\phi$ is conjunction of formulae $\neg(c_j = 0)$ for actions $o_j$ with *at end* effect changing a variable in the *at end* effect of $o_i$ (excluding $o_i$ itself) and formulae $\neg(c_j = DU(o_j))$ for actions $o_j$ with *at start* effect changing a variable in the *at end* effect of $o_i$.

  If the action's counter equals its duration then its *at start* condition must be true and no variable in it is changed by another action's *at start* or *at end* effect at the current time point: this is analogous to the previous case for *at end*.

  If the action's counter is $\geq 1$ then its *over all* condition must be true[2]: $c_i \geq 1 \rightarrow OA(o_i)$.

- The action always makes $F$ true with $\top \rhd F$.

  For every action $o_i$ in the original temporal problem instance the effect does the following.

  If the action's counter equals its duration then execute its *at start* effects: $c_i = DU(o_i) \rhd SE(o_i)$.

  If the action's counter equals 0 then execute its *at end* effects: $c_i = 0 \rhd EE(o_i)$.

Actions for progressing time decrease the values of all active counters with some amount. The preconditions of the actions require that no end point of an active action is passed.

It is sufficient to have one action for progressing time by 1 but a more practical alternative is use several with increments 1, 2, 4, 8, ... so that only a logarithmic number of actions is needed for progressing a long period of time.

We define an action for progressing time by $t$.

- The precondition tests that the effects for the current time point have been executed and that every counter $c$ satisfies $c \leq 0 \vee c \geq t$ so that no action end points are skipped.

---

[2]PDDL does not handle *over all* conditions and *at start/end* condition uniformly: for the former it suffices for them to be true but for the latter it is additionally required that no variables occurring in them change.

- The action effect adds $-t$ to every counter (but values are not decreased below $-1$) and further progression is prevented if an end point of an action is encountered.

- The action is

  $\langle F \wedge \bigwedge_{i=1}^{m}(c_i \leq 0 \vee c_i \geq t),$
  $\{c_i := c_i - t | 1 \leq i \leq m\} \cup \{c_i = t \rhd \neg F | 1 \leq i \leq m\}\rangle.$

**Theorem 12** *Discrete temporal PDDL is polynomial-time reducible to classical planning.*

It is straightforward to extend the above translation to handle the conditional effects in temporal PDDL.

An inspection of instances of temporal planning problems described in research papers in the area and used in planning competitions did not turn up any problem instance which requires an action overlapping itself. Theorem 12 shows how to reduce all of these problems to classical planning.

## PDDL with Self-Overlapping Actions

The definition of temporal PDDL (Fox & Long 2003) does not make it clear whether an action can overlap with another instance of itself. Earlier we showed that temporal PDDL without such overlapping is polynomial-time reducible to classical planning. For temporal PDDL with self-overlapping this is not possible because the plan existence problem is EXPSPACE-hard.

The simulation of EXPSPACE Turing machines in the proof of Theorem 10 can be modified to the temporal PDDL setting. For the simulation to be correct actions mapping the current configuration to the next have to be forced to take place at the right time intervals, corresponding to the relevant tape cells.

Like in the proof of Theorem 10 in the beginning of the interval of every tape cell actions for the changes to obtain the *next configuration* have to be taken. This involves copying the current cell content if the cell is not under the R/W head, writing new content to the current R/W location, and indicating the new position of the R/W head. Similarly to the proof of Theorem 10, one action needs to be executed for the new contents of the cell and two actions for setting the variable $H$ for the R/W head position and the new internal state $q \in Q$. Additionally, other two actions respectively preceding and succeeding the said three actions are taken to enable the same steps for the next cell.

A small amount of book-keeping is needed to guarantee that if some of the actions corresponding to one cell are not taken, plan execution cannot proceed with actions corresponding to the next tape cell, to invalidate plans that do not correspond to faithful Turing machine simulations.

This simulation shows that the plan existence for temporal PDDL with self-overlapping actions is EXPSPACE-hard.

## Related Work

Cushing et al. (2007) show that their *temporally simple* languages can be reduced to plain classical planning in the trivial sense that no concurrent (overlapping) actions are required. They also show that there is no same kind of simple

reduction of their *temporally expressive* languages to classical planning. However, their notion of reduction is very restrictive, and does not include unrestricted polynomial time reductions. Our Theorem 12 shows for a very wide class of temporal languages that they are easily reducible to classical planning. Reducibility in general is determined by the possibility of having an exponential number of actions simultaneously active.

There are planning systems which implement different forms of temporal planning in connection with numeric state variables (Muscettola 1993; Penberthy & Weld 1994; Laborie & Ghallab 1995). No analysis of the power and complexity of these planning systems exists, apart from the observation that a sufficiently strong language with unbounded integer or real variables is undecidable, even without considering aspects related to planning (Helmert 2002).

## Conclusions

We have analyzed temporal planning by identifying a border between PSPACE-complete and EXPSPACE-complete planning problems which also indicates a border between classical and more expressive temporal planning problems.

Important special cases of the temporal planning problem include those with restrictions on plan lengths. Temporal planning with polynomial plan lengths can be easily seen to be NP-complete (enabling a reduction to SAT), and with exponentially long plans it is NEXPTIME-complete. NEXPTIME-hardness can be shown by a variant of the proof of Theorem 10. This proof can also be combined with some of the proofs by Rintanen (2004). For example, it is easy to show that nondeterministic temporal planning with full observability is 2-EXPTIME-hard.

Interestingly, the complexity of temporal planning with schematic operators coincides with that of schematic (nonground) classical planning: both are EXPSPACE-complete.

The work has implications on algorithm development for temporal planning. First, the restrictions which force the planning problem to PSPACE allow the use of classical planning algorithms: the temporal aspects of the planning problem can be reduced to the use of a polynomial number of counters. Second, more expressive temporal planning requires stronger techniques which may be different from the currently existing ones.

The granularity of discretizations may have a strong impact on the efficiency of plan search. The unit time in a problem instance with integer time may be impractically short and problem instances with rational time may yield too fine discretizations. An important research topic is the derivation of more practical discretizations, possibly in combination with altered action durations.

Although the reduction to classical planning is possible with only Boolean state variables, it may be more efficient if (bounded) integer-valued state variables can be used. This underlines the close connection between temporal planning and planning with numeric state variables. It seems that efficient algorithms for planning with numeric state variables should also be quite efficient for temporal planning. Most immediately interesting frameworks are integer programming IP and extensions of SAT for handling numeric variables.

## References

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artificial Intelligence* 69(1-2):165–204.

Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. S. 2007. When is temporal planning really temporal. In Veloso, M., ed., *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 1852–1859. AAAI Press.

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

Haslum, P., and Geffner, H. Heuristic planning with time and resources. In Cesta, A., ed., *Recent Advances in AI Planning. Sixth European Conference on Planning (ECP'01)*, Lecture Notes in Artificial Intelligence. Springer-Verlag. to appear.

Helmert, M. 2002. Decidability and undecidability results for planning with numerical state variables. In Ghallab, M.; Hertzberg, J.; and Traverso, P., eds., *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2002)*, 303–312.

Laborie, P., and Ghallab, M. 1995. IxTeT: an integrated approach for plan generation and scheduling. In *1995 INRIA/IEEE Symposium on Emerging Technologies and Factory Automation: Proceedings, ETFA '95, Paris, France, October 10-13, 1995*, 485–495.

Muscettola, N. 1993. HSTS: Integrating planning and scheduling. Technical Report CMU-RI-TR-93-05, Robotics Institute, Carnegie Mellon University.

Penberthy, J. S., and Weld, D. S. 1994. Temporal planning with continuous change. In *Proceedings of the 12th National Conference on Artificial Intelligence*, 1010–1015.

Pinto, J. 1998. Concurrent actions and interacting effects. In Cohn, A. G.; Schubert, L. K.; and Shapiro, S. C., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR '98)*, 292–303. Morgan Kaufmann Publishers.

Rintanen, J. 2004. Complexity of planning with partial observability. In Zilberstein, S.; Koehler, J.; and Koenig, S., eds., *ICAPS 2004. Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, 345–354. AAAI Press.

Smith, D. E., and Weld, D. S. 1999. Temporal planning with mutual exclusion reasoning. In Dean, T., ed., *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, 326–337. Morgan Kaufmann Publishers.