Strong Temporal Planning with Uncontrollable Durations[☆]

Alessandro Cimatti[a], Minh Do[b], Andrea Micheli[a], Marco Roveri[a], David E. Smith[b]

[a] *Fondazione Bruno Kessler, Istituto per la Ricerca Scientifica e Tecnologica*
Via Sommarive 18, 38123 Povo, Trento, Italy
[b] *NASA Ames Research Center*
M/S 269-2 Moffett Field, CA 94035-1000, USA

Abstract

Planning in real world domains often involves modeling and reasoning about the duration of actions. Temporal planning allows such modeling and reasoning by looking for plans that specify start and end time points for each action. In many practical cases, however, the duration of actions may be uncertain and not under the full control of the executor. For example, a navigation task may take more or less time, depending on external conditions such as terrain or weather.

In this paper, we tackle the problem of strong temporal planning with uncontrollable action durations (STPUD). For actions with uncontrollable durations, the planner is only allowed to choose the start of the actions, while the end is chosen, within known bounds, by the environment. A solution plan must be robust with respect to all uncontrollable action durations, and must achieve the goal on all executions, despite the choices of the environment.

We propose two complementary techniques. First, we discuss a dedicated planning method, that generalizes the state-space temporal planning framework, leveraging SMT-based techniques for temporal networks under uncertainty. Second, we present a compilation-based method, that reduces any STPUD problem to an ordinary temporal planning problem. Moreover, we investigate a set of sufficient conditions to simplify domains by removing some of the uncontrollability.

We implemented both our approaches, and we experimentally evaluated our techniques on a large number of instances. Our results demonstrate the practical applicability of the two techniques, which show complementary behavior.

Keywords: Strong Temporal Planning, Strong Controllability, Temporal Planning, Uncontrollable Durations, Forward State Space Planning, PDDL,

[☆] This paper is an extended version of [1] and [2], published at AAAI 2015 and at IJCAI 2015 respectively.

Email addresses: `cimatti@fbk.eu` (Alessandro Cimatti), `minh.do@nasa.gov` (Minh Do), `amicheli@fbk.eu` (Andrea Micheli), `roveri@fbk.eu` (Marco Roveri), `david.smith@nasa.gov` (David E. Smith)

1. Introduction

Planning in real world domains often involves modeling and reasoning about the duration of actions. For example, the activities of a planetary rover, such as navigation and data transmission, require non-negligible time to be completed, and their successful execution is subject to timing constraints. Both navigation and transmission actions must occur in time windows when sufficient solar power is available, and the transmission action must occur in a time window when a receiving satellite is in view.

An approach to tackle the problem consists in combining classical planning, where actions are assumed to be instantaneous, with a subsequent scheduling phase, where the durations are taken into account. Unfortunately, valid classical plans may turn out not to admit a feasible schedule.

Temporal planning [3] attempts to overcome this problem by allowing for modeling and reasoning about *durative actions*. In the domain description, actions are associated not only to preconditions and effects on the state, but also with constraints on the durations. For example, a navigation or drilling task may have lower and upper duration bounds. A temporal planner looks for plans where each action is associated with start and end time points. The idea can be described as effectively integrating the scheduling aspects within the search for the set of actions to be executed, checking at the same time for the existence of a schedule.

In some practical applications, however, the duration of actions may be uncertain and not under the control of the executor. For example, a navigation task to a given location may take more or less time, depending on external factors, such as terrain or weather conditions. The plan may specify when to start the operation, but the actual duration is controlled by the environment.

In this paper, we introduce the problem of *Strong Temporal Planning with Uncontrollable Durations* (STPUD). In this setting, the planner is only allowed to choose the start of the actions, while the duration of uncontrollable actions is chosen by the environment at run time, within known bounds. A solution plan is required to be temporally strong, i.e. robust, with respect to uncontrollable action durations, and to achieve the goal over all possible executions, despite the run-time choices of the environment. These plans are quite similar to temporal plans in PDDL, where a specific starting time and duration is specified for each action in the plan, with the difference being that we do not specify the duration of uncontrollable actions.

The STPUD problem is the lifting to the planning case of the strong controllability problem for temporal networks [4, 5] and thus does not allow conditional execution based on the observation of action durations at run-time. This means that all the uncertainty is resolved at planning time and the plans do not contain conditional branches. Such plans can be sub-optimal in terms of make-span, and one can construct examples where strong plans do not exist. However, for duration uncertainty, this seems to be less common than for other types of uncertainty (e.g. uncertainty in the outcome of actions). For example, with Mars rovers there is considerable duration uncertainty for driving operations, but the approach used in practice is to construct strong plans assuming maximum duration, and just wait if the rover gets to a location too early for the next desired activity. This is practical in this domain because there is often little penalty for waiting, except the delay of future actions. In general, unless a domain has a lot of intricate temporal constraints with both lower and upper bounds, there is a good chance that a strong plan exists. This plan may not be optimal if one is interested in minimizing make-span, but it usually exists. Moreover, in some situations the executor of the plan may not have the time or the computational power to perform a re-scheduling of the plan at run-time. A typical example is a spacecraft during an orbit insertion maneuver: the pace of the operations and communication limitations require that a plan be pre-computed with no run-time adjustment.

One distinct advantage of strong plans is understandability: a human can read, understand and check a STPUD plan, while the same operation can be extremely difficult for a conditional plan or a flexible plan represented for example as an STN(U) or even as a simple precedence graph. Finally, while this paper focuses on techniques for automatic synthesis of strong plans, the resulting plans can often be relaxed to partially-ordered plans before execution, similar to what can be done for deterministic PDDL plans to reduce make-span (e.g. [6]). This could improve the quality of such plans during execution, but adapting such techniques for the STPUD case is left for future work.

In this paper, we explore two complementary approaches to STPUD. First, we discuss a dedicated planning method to deal with uncontrollable durations, which generalizes the forward state-space temporal planning (FSSTP) framework introduced in [7] and used in [8, 9]. Intuitively, FSSTP applies classical planning over an abstraction of the temporal domain, where temporal precedences over events are taken into account at a qualitative level, to enumerate candidate plans. The quantitative aspects are then taken into account by solving the consistency problem of the induced Simple Temporal Network (STN) [10]. In order to deal with temporal uncertainty, we retain the main loop of FSSTP, dealing with the quantitative aspects by means of Temporal Networks with Uncertainty[1] (TNUs), leveraging recent scheduling techniques for TNU [11] based on Satisfiability Modulo Theories [12]. We call the derived technique S-FSSTP. This approach is far from being trivial; we show how simply replacing the STN in [9] with the corresponding STNU may result in an unsound technique.

Second, we present a compilation-based planning method, that reduces any STPUD problem to a temporal planning problem, where all the actions have controllable durations. The sound-and-complete reduction eliminates uncontrollable durations by introducing intermediate effects and conditions, which

[1]The term *uncertainty* is used here for historical reasons; we believe that *temporal uncontrollability* would be more appropriate.

are events that take place during an action execution.

We also investigate a domain transformation technique that is able to eliminate some of the uncontrollable durations by reasoning in terms of worst case execution. This technique preserves the space of plans/set of solutions, and can be used as a preprocessor for both the planning approaches.

The approaches described above have been implemented and experimentally analyzed. We considered a large number of instances obtained by extending the temporal planning domains available in the literature with uncontrollable durations. Our results demonstrate the practical applicability of our approaches, and provide interesting insights. First, the two approaches we present often exhibit complementary behaviors, so that further efficiency can be achieved using a portfolio approach. Second, the preprocessing technique has negligible costs, but it greatly pays off, in selected cases, for both planning approaches.

Paper structure. The paper is structured as follows. In Section 2 we discuss related work. Section 3 introduces and formalizes the Strong Temporal Planning with Uncontrollable Durations problem. In Section 4 we give a structured overview of the approaches. Section 5 proposes a generalization of the state-space temporal planning framework for solving the STPUD problem. In Section 6 we describe the compilation technique, which removes the duration uncertainty from a given STPUD. In Section 7, we describe the domain simplification technique for uncontrollability elimination. In Section 8 we experimentally evaluate the merits of all the proposed techniques. We conclude in Section 9 by reviewing the paper and proposing future research directions.

2. Related Work

A few works in the literature consider the problem of planning with uncontrollable durations, but none of them consider the STPUD problem as it is presented in this paper. The only framework available for planning with uncertain duration is probabilistic, while here we focus on exact techniques for Strong planning with interval durations. There is a significant corpus of papers concerning the pure scheduling problem, most notably works on temporal networks with uncertainty, but these do not consider plan generation from a model of a system, only the temporal allocation of activities. STPUD can be thought of as a combination of scheduling under temporal uncertainty with temporal planning.

2.1. Temporal Networks with Uncertainty

Temporal uncertainty is a well-understood concept in scheduling. It has been widely studied [13, 14, 15, 11]. Given a set of actions with uncontrollable durations subject to a set of temporal constraints, we are interested in solutions to schedule the actions such that all the temporal constraints are respected regardless of the actual duration the actions may take. Depending on the information available to the scheduler at run-time, different classes of problems have been defined [4].

The problem we address in this paper can be seen as a generalization of Strong Controllability for Temporal Networks [4, 5] to planning (rather than scheduling). Strong controllability is the problem of finding a fixed schedule for the controllable time points that fulfills all the constraints in each valid situation encoded by the network. Dealing with planning is much harder because the actions (and thus the time points associated with them) in a plan are not known a-priori and must be searched for. Moreover causal relationships in planning are much more complex than Temporal Network constraints.

2.2. Temporal Planning

In temporal planning, duration uncertainty is a known challenge [16], but few temporal planners address it explicitly. Some temporal planners [17, 18] cope with this issue by generating "flexible temporal plans": instead of fixing the execution time of each action, they return a (compactly represented) set of plans that must be scheduled at run-time by the plan executor. This approach cannot guarantee plan executability and goal achievement at runtime, because there is no formal modeling of the boundaries and contingencies in which the system is going to operate. Instead, our proposed approach requires a formal modeling of the problem uncertainty and guarantees plan executability and goal achievement under any modeled contingency. In addition, flexible plans require an executor able to solve constraints at runtime to schedule the activities. Flexibility and controllability are complementary: controllability provides guarantees with respect to the modeled uncertainty, while flexibility allows the plan to be adjusted during execution. In principle, we can use any temporal planner that can generate flexible plans in combination with our compilation to generate a flexible strong plan.

IxTeT [19] is the first attempt to lift to the planning case the results in temporal reasoning under uncertainty. The planner generates a strategy for scheduling the actions depending on contingent observations and encodes it as a dynamically controllable Simple Temporal Network with Uncertainty (STNU) [4]. This plan representation requires a powerful plan executor, able to dynamically schedule an STNU. These plans sometimes work in more situations than the Strong Plans we generate in our work, but they are also more complex to generate, understand, and execute. Strong Plans are frequently required for safety critical systems like space applications, where guarantees are needed, and computational power is limited.

There has also been work on temporal planning with probabilistic duration uncertainty, e.g. [20, 21, 22, 23, 24, 25, 26, 27]. This work assumes a probability distribution over action durations, and attempts to generate policies or conditional plans that branch on the observed action durations at run time. A good discussion of much of this work can be found in [25]. The primary limitations of these approaches is that they are either incomplete (only generate partial policies), or tend to suffer from scalability issues. Instead we assume that durations are bounded in convex intervals, and attempt to generate Strong Plans – that is, non-branching plans that guarantee goal achievement and plan executability in all the allowed contingencies.

There is a clear parallel between the problem we are solving and conformant planning [28]. In this sense, the work we describe in Section 6 is similar to [29] in which the authors transform conformant planning into deterministic planning, although the translation is very different.

Finally, the work in [30] presents a logical characterization of the STPUD problem for timelines with temporal uncertainty, as well as a first-order encoding of the problem having bounded horizon. In Section 5 of this paper, we generalize this framework, as we do not impose any bounded horizon for the problem and we consider a more expressive language allowing disjunctive preconditions, effects at arbitrary time points during actions, and durative conditions on arbitrary sub-intervals.

3. The STPUD Problem

In this section, we formally define the syntax and semantics of Strong Temporal Planning with Uncontrollable Durations (STPUD) and we discuss the issues that arise in comparison to temporal planning without uncertainty.

3.1. Syntax

In order to express planning problems with uncontrollable durations, we propose a rich language that includes timed-initial-literals (TILs), and multi-valued variables. In addition, we extend the language to allow conditions expressed over sub-intervals of actions, and effects at arbitrary time points during an action. These features turn out to be particularly useful for encoding many problems of interest, and for our translation[2].

In order to define the abstract syntax of our language we need to consider four kinds of intervals, namely closed, left-open, right-open and open. We will use these intervals to express durative conditions.

Definition 1. *Given two numeric expressions a and b, we define the four possible intervals having extremes a (lower bound) and b (upper bound) as: $[a, b]$ closed interval; $(a, b]$ left-open interval; $[a, b)$ right-open interval; and (a, b) open interval.*

We write $[\!(a, b)\!]$ to indicate an instance of the above possibilities without distinguishing the type, similarly $[\!(a, b]$ indicates an interval that can be open or closed on the left, but closed on the right, and so on. Moreover, we write $[a]$ to indicate the single-point interval $[a, a]$.

We can now define a STPUD planning problem P as a tuple $\langle V, I, T, A, G \rangle$ where:

[2]To simplify the presentation, we exclude some features that are orthogonal to our approach of handling temporal uncertainty, such as *numeric variables* and *domain axioms*. Our techniques will work whether or not those features are included.

- $V = \{f_1, \cdots f_n\}$ is a finite set of variables[3], each having a domain $Dom(f_i)$.
- I is the initial state: a complete assignment of values to each variable in V: for each variable $f \in V$, $I(f) \in Dom(f)$.
- T is a set of timed-initial-literals, each of the form $\langle [t]\ f := v \rangle$ with $f \in V$, $v \in Dom(f)$ and $t \in \mathbb{R}_{>0}$. The real number t is the wall-clock time at which f will be assigned the value v.[4]
- A is a set of durative actions each of the form $a \doteq \langle [l, u], C, E \rangle$ where[5]:
  - $l, u \in \mathbb{R}_{>0}$, with $l \leq u$ being the action duration bounds.
  - C is the set of disjunctive conditions; each element $c \in C$ is a condition of the form $\langle [\![st_c, et_c]\!]\ \bigvee_{i=1}^{n} f_i = v_i \rangle$ where each $f_i \in V$ and $v_i \in Dom(f_i)$. The expressions st_c and et_c indicate the start and end time points of the condition c and are restricted to be of the form $st_a + \delta$ or $et_a - \delta$ with $0 \leq \delta \leq l$.
  - E is a set of instantaneous effects; each $e \in E$ is of the form $\langle [t_e]\ f := v \rangle$ where $f \in V$, $v \in Dom(f)$ and $t_e \doteq st_a + \delta$ or $t_e \doteq et_a - \delta$ with $0 \leq \delta \leq l$.
- G is the set of disjunctive goal conditions, each of the form $\bigvee_{i=1}^{n} f_i = v_i$, where each $f_i \in V$ and $v_i \in Dom(f_i)$.

Intuitively, the syntactic tokens st_a and et_a indicate the start and end times of action a, and are used to specify timings of effects and conditions relative to the action timing. Note that for the sake of simplicity, we do not permit a condition c with $st_c = et_a - \delta_1$ and, simultaneously, $et_c = st_a + \delta_2$, while all other cases are allowed. In practice, this means that we cannot express a condition that starts at a time point relative to the end of the action and ends at a time point expressed relative to the start of the action. We believe this combination isn't particularly useful in practice and this limitation simplifies the semantics and the explanation of the techniques. However, there is no fundamental problem in extending all the proposed techniques to cover this case. Moreover, we do not require st_c to be lower than (or equal to, if the interval is closed) et_c: this is because such an interval would be empty (i.e. it would contain no time-point). As such, any condition expressed over an empty interval is imposed over no time-point, and thus is vacuously true. All the subsequent semantics and techniques are defined in such a way that an empty interval poses no constraints on the problem. For these reasons, we allow this degenerate case. Finally, when the interval for a condition c is a single point (i.e. $[st_c]$), the condition is instantaneous.

[3]We indicate the variables with the letter f (for "fluent") to avoid confusion with values that will be indicated with v.

[4]In this problem formalization, we fix the timing for timed-initial-literals. However, allowing a window of possible timings that can be uncertain or controllable doesn't complicate anything, and the techniques we propose can be easily extended to handle them. In fact, we can think of timed-initial-literals as end effects of actions starting at time 0.

[5]We use $\doteq$ to distinguish symbol definitions from equality constraints.

		Action Duration	
		Controllable Only	**Uncontrollable**
Condition/Effect Timing	**Extremes + Overall**	Ctrl-Extr (PDDL 2.1)	Unc-Extr
	Arbitrary Sub-Intervals	Ctrl-Arbit (ANML, NDL)	Unc-Arbit

Table 1: Classification of relevant problem sub-classes. For each case, we indicated the abbreviation name used in the paper and a planning language representative in written in parenthesis where available.

We assume that the set of actions A is partitioned into two sets A_c and A_u of controllable and uncontrollable actions, respectively. This is needed to distinguish between actions whose duration can be controlled by the agent, and those that have uncontrollable durations. In both cases, the bounds on the duration need to be satisfied, but analogously to TNU contingent links, if an action is uncontrollable, its duration is assumed to take a value in the specified bound.

The solution to a STPUD problem P is a plan π that assigns the starting of all the actions and specifies the duration only for controllable actions.

Definition 2. *A plan π of P is a finite set of tuples $\langle t, a, d \rangle$, in which actions $a \in A$, $t \in \mathbb{R}_{\geq 0}$, $d \in \mathbb{R}_{>0}$ if $a \in A_c$ and $d = \bot$ if $a \in A_u$.*

Intuitively, an element $\langle t, a, d \rangle$ indicates the start of an instance of the action a at time t, with duration d if a is a controllable action. The assignment of $d = \bot$ for uncontrollable actions reflects the intuitive notion that since the duration is not under the control of the plan executor, the plan cannot specify it.

3.1.1. Discussion

The planning problem formalism we describe includes several features that allow considerable expressiveness. In particular, we focus on the presence of uncertainty in the action durations, that constitutes the main objective of this work. We also allow the presence of "intermediate" effects and conditions: the possibility of having action effects at intermediate times, and to impose conditions in sub-intervals of the action execution. This latter feature is not new, languages such as ANML [31] or NDL [32] support it natively, but it is not natively supported in other languages. For example, PDDL 2.1 does not allow for intermediate effects nor for conditions at times different from the start, the end, or the entire action duration.

If we classify according to the presence or absence of these features, we obtain the landscape of planning problem sub-classes depicted in Table 1. The table shows four classes of problems. Clearly, every class with arbitrary intervals subsumes the class with extreme intervals having the same action controllability. Similarly, each class having action uncertainty is strictly more general than the class without it. Given these subsumption rules, the only two incomparable classes are Unc-Extr and Ctrl-Arbit: both subsume Ctrl-Extr, but no obvious relation is present between the two. Unc-Arbit is the most general class that subsumes every other case and coincides with the language we discussed in the previous section. The aim of this work is to tackle the Unc-Arbit problem class in its full complexity.

Some of the reported problem classes are supported in the literature and by dedicated planning tools. In particular, Ctrl-Extr is the temporal planning problem addressed by all the planners supporting the PDDL 2.1 language. Ctrl-Arbit is a sub-case of the features provided by the ANML language, therefore tools such as FAPE [33] can natively reason on instances of this class. Moreover, reduction techniques have been presented for transforming an instance of the Ctrl-Arbit class to a problem in Ctrl-Extr [34, 31].

3.2. Running Example

We give a small example problem that will be used throughout the paper.

A rover, initially at location l_1, needs to transmit some science data from location l_2 to an orbiter that is only visible in the time window $[14, 30]$. The rover can move from l_1 to l_2 using an action *move* that has uncontrollable duration between 10 and 15 time units. The data transmission action *trans* takes between 5 and 8 time units to complete. The goal of the rover is to transmit the data to the orbiter. Because of the harsh daytime temperatures at location l_2, the rover cannot be at l_2 until the sun goes behind the mountains at time 15. Figure 1 illustrates this scenario, which we encode as follows[6]:

$$\begin{aligned}
V &\doteq \{pos : \{l_1, l_2\}, visible : \{\mathtt{T}, \mathtt{F}\}, hot : \{\mathtt{T}, \mathtt{F}\}, sent : \{\mathtt{T}, \mathtt{F}\}\}\\
I &\doteq \{pos = l_1, visible = \mathtt{F}, sent = \mathtt{F}, hot = \mathtt{T}\}\\
T &\doteq \{\langle [14]\ visible := \mathtt{T}\rangle, \langle [30]\ visible := \mathtt{F}\rangle, \langle [15]\ hot := \mathtt{F}\rangle\}\\
G &\doteq \{(sent = \mathtt{T})\}\\
A_c &\doteq \emptyset\\
A_u &\doteq \{\langle [10, 15], C_{move}, E_{move}\rangle, \langle [5, 8], C_{trans}, E_{trans}\rangle\}\\
C_{move} &\doteq \{\langle [st_{move}]\ pos = l_1\rangle, \langle [et_{move}]\ hot = \mathtt{F}\rangle\}\\
C_{trans} &\doteq \{\langle [st_{trans}, et_{trans}]\ pos = l_2\rangle, \langle [st_{trans}, et_{trans}]\ visible = \mathtt{T}\rangle\}\\
E_{move} &\doteq \{\langle [et_{move}]\ pos := l_2\rangle\}\\
E_{trans} &\doteq \{\langle [et_{trans}]\ sent := \mathtt{T}\rangle\}
\end{aligned}$$

[6]In the formalization of the *move* action, the rover doesn't change its position until the end of the action. A more typical formulation sets the rover position to a special value "undefined" at the beginning of the *move* action. In a propositional formulation, like in PDDL, this is accomplished by simply deleting the rover position as a start effect. This is completely orthogonal to the uncertainty features of the language so we decided to keep the example as simple as possible.

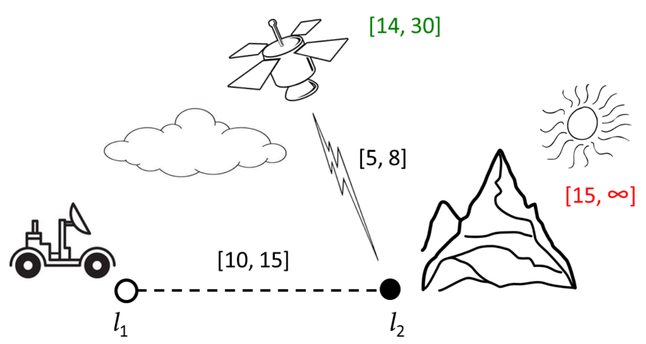


Figure 1: A graphical representation of the running example. The rover, initially at l_1 can move to l_2 where it can transmit data to a satellite. Action durations are indicated in black, the satellite visibility window is in green and the interval where the temperature in l_2 is favorable is indicated in red.

The *pos* variable indicates the position of the rover, *visible* is true if the satellite is available for data transmission, *hot* is true if the temperature at l_2 is too hot for the rover, and *sent* is set to `T` after the data is sent to the satellite.

Figure 2 graphically shows a valid strong plan π_{ex} for the running example. The plan is defined as follows.

$$\pi_{ex} \doteq \{\langle 6, move, \perp\rangle, \langle 22, trans, \perp\rangle\}$$

Note that all the actions in π_{ex} have uncontrollable duration; hence, the strong plan does not specify their duration (durations are replaced by $\perp$).

3.3. Semantics

We give the semantics of the planning language by defining the validity of a plan π for any given STPUD problem P. As usual, P admits a solution if there exists a valid plan, otherwise the problem is said to be unsolvable.

We start by defining the projection of a STPUD problem. Intuitively, in a projected problem, the durations of uncontrollable actions become controllable choices for the planner. For example, the *trans* action is originally assumed to last between 5 and 8 time units; in the projection this choice becomes controllable, but we retain the requirement that the duration lies within 5 and 8.

Definition 3 (Projected Problem)**.** *Given a STPUD problem $P \doteq \langle V, I, T, A, G\rangle$ with durative actions $A \doteq A_c \cup A_u$, the projected problem without uncertainty is a STPUD $ctrl(P) \doteq \langle V, I, T, A', G\rangle$ that is identical to P except for the partition of the set of actions that is $A' \doteq A'_c \cup A'_u$ with $A'_c \doteq A_c \cup A_u$ and $A'_u \doteq \emptyset$.*

The basic element of our semantics is a chronicle, that is used to assign a value to each variable in V for each time instant.

Definition 4 (Chronicle)**.** *A chronicle τ for a given STPUD problem instance $P \doteq \langle V, I, T, A, G\rangle$ is a set of functions $\tau_f : \mathbb{R}_{\geq 0} \to Dom(f)$, one for each $f \in V$.*

Given a plan, we can now define the chronicle induced by it. In our language we have three components that contribute to change the state of a variable, namely the initial state, the TILs and action effects. Apart from these events, each variable is assumed to maintain its value in the other time instants. To formalize this concept we start by collecting the set of change events in the execution of the plan.

Definition 5 (Set of Changes SoC)**.** *Given a projected planning problem instance $ctrl(P) \doteq \langle V, I, T, A, G\rangle$ and a plan $\pi \doteq \{\langle t_i, a_i, d_i\rangle \mid i \in [1, n]\}$, the multi-set of changes induced by π is a set $\textsc{SoC}(ctrl(P), \pi)$ defined as follows.*

- *for each $f \in V$, $\langle 0, f, I(f)\rangle \in \textsc{SoC}(ctrl(P), \pi)$;*
- *for each $\langle [t]\, f := v\rangle \in T$, $\langle t, f, v\rangle \in \textsc{SoC}(ctrl(P), \pi)$;*
- *for each $\langle t, a, d\rangle \in \pi$ with $a \doteq \langle [l, u], C, E\rangle$:*
  - *for each $\langle [st_a + \delta]\, f := v\rangle \in E$, $\langle t + \delta, f, v\rangle \in \textsc{SoC}(ctrl(P), \pi)$;*
  - *for each $\langle [et_a - \delta]\, f := v\rangle \in E$, $\langle t + d - \delta, f, v\rangle \in \textsc{SoC}(ctrl(P), \pi)$.*

Given a positive time $x \in \mathbb{R}_{>0}$ and a variable f, let PE_x^f be all the elements $\langle t, f, v\rangle$ of $\textsc{SoC}(ctrl(P), \pi)$ with $t < x$; let $maxPE_x^f$ be the maximum timing in PE_x^f (i.e. $max(t \mid \langle t, f, v\rangle \in PE_x^f)$). We indicate with $\textsc{PreC}(ctrl(P), \pi, x, f)$ the multi-set $\{v \mid \langle maxPE_x^f, f, v\rangle \in PE_x^f\}$.

We remark that, at this stage, we need to keep a multi-set of changes because it is possible for two effects to set the same variable to the same value. We consider such a situation illegal and we will catch it in Definition 8.

Now, we can define the chronicle induced by a plan by imposing that at each time point corresponding to a change in SoC, the chronicle changes its value; and between two successive changes, the chronicle maintains its value. In this sense, the chronicle is constrained to be a piecewise-constant function. The $\textsc{PreC}(ctrl(P), \pi, x, f)$ multi-set contains the changes on variable f that are applied immediately before time x. Note the strict inequality in the definition of $\textsc{PreC}(ctrl(P), \pi, x, f)$: we consider the value of the change immediately before, but not at the time x.

Definition 6 (Induced Chronicle)**.** *Given a projected planning problem instance $ctrl(P) \doteq \langle V, I, T, A, G\rangle$ and a plan π, a chronicle τ^π induced by π is defined as follows.*

For each variable f,

- *$\tau_f^\pi(0) = v$ with $\langle 0, f, v\rangle \in \textsc{SoC}(ctrl(P), \pi)$;*
- *for each $x \in \mathbb{R}_{>0}$, $\tau_f^\pi(x) = v$ with $v \in \textsc{PreC}(ctrl(P), \pi, x, f)$.*

Intuitively, each variable in each time point assumes the value imposed by the last change until another is applied. Note that, if no two effects on the same variable happen at the same time, then the induced chronicle is unique (otherwise, we will deem the execution invalid in Definition 8).

For example, consider an effect $\langle [et_a] f := v\rangle$ and suppose the action a ends at absolute time 10. The value of f is not changed at time 10 but immediately after. This means, that a condition requiring f to have value v is not satisfied at time 10, but a positive amount of time is required to pass. This view is practically compatible with the PDDL 2.1 specification[7] and also with the continuous time version of the ANML language.

As an auxiliary definition, we introduce the Absolute-Time Interval of a condition of an action appearing in a plan. The idea is to define the subset of the time points in which each condition is required to hold, given a plan.

Definition 7 (Absolute-Time Interval)**.** *Given a plan π and a plan element $\langle t, a, d\rangle \in \pi$, the absolute-time interval of a condition $c \in C$ of action a is a subset of the real numbers $ATI(c, \langle t, a, d\rangle)$ defined as follows.*

$$ATI(c, \langle t, a, d\rangle) \doteq \begin{cases} \llbracket t + \delta_s, t + \delta_e \rrbracket & \text{if } c = \langle \llbracket st_a + \delta_s, st_a + \delta_e \rrbracket\ \phi\rangle \\ \llbracket t + \delta_s, t + d - \delta_e \rrbracket & \text{if } c = \langle \llbracket st_a + \delta_s, et_a - \delta_e \rrbracket\ \phi\rangle \\ \llbracket t + d - \delta_s, t + d - \delta_e \rrbracket & \text{if } c = \langle \llbracket et_a - \delta_s, et_a - \delta_e \rrbracket\ \phi\rangle \end{cases}$$

We can now define the validity of a plan for a projected problem. Intuitively, we need to check three conditions to validate a plan. First, we require that there are not two or more changes of the same variable at the same time; second, we check that all the conditions of all the actions used in the plan are satisfied; and finally we require that all the goals are reached immediately after the end of the last action in the plan.

Definition 8 (Projected Problem Plan Validity)**.** *Given a projected problem $ctrl(P) \doteq \langle V, I, T, A, G\rangle$, a plan π is valid if the following conditions hold:*

1. *for each $t \in \mathbb{R}_{\geq 0}$ and each $f \in V$,*
   $|\{\langle t, f, v\rangle \mid \langle t, f, v\rangle \in \mathrm{SoC}(ctrl(P), \pi)\}| \leq 1$;
2. *for each $\langle t, a, d\rangle \in \pi$ with $a \doteq \langle [l, u], C, E\rangle$, the following holds:*
   - *$d \in [l, u]$;*
   - *for each $c = \langle \llbracket st_c, et_c \rrbracket \bigvee_{i=1}^{n} f_i = v_i\rangle \in C$, and each $x \in ATI(c, \langle t, a, d\rangle)$, $\bigvee_{i=1}^{n} \tau^{\pi}_{f_i}(x) = v_i$ holds;*
3. *for each goal condition $(\bigvee_{i=1}^{n} f_i = v_i) \in G$, $\bigvee_{i=1}^{n} \tau^{\pi}_{f_i}(x) = v_i$ holds, for any $x \in [t_{max}, t_{max} + \epsilon]$ with $t_{max} \doteq max(\{t + d \mid \langle t, a, d\rangle \in \pi\})$ and a sufficiently small $\epsilon \in \mathbb{R}_{>0}$.*

[7]In PDDL 2.1, effects are applied immediately, but a changed predicate or fluent cannot be inspected until immediately after. In addition, PDDL 2.1 requires a known minimal time quantum ϵ to pass between any pair of changing points (an instantaneous action or the start or end of a durative action) if the conditions/effects in such a pair of points are interfering [3].

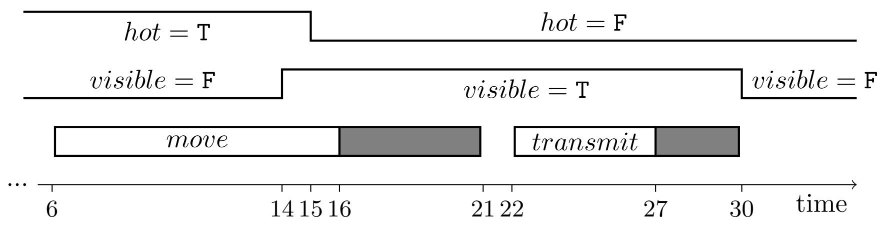


Figure 2: Graphical execution of π_{ex}. Filled regions represent the uncertainty in the action duration.

We remark that here we are defining the semantics of a valid plan for a projected planning problem, hence we reject as invalid any plan yielding an execution that violates any of the above constraints.

Finally, we can define the semantics of any STPUD problem P by imposing that each plan obtained by specifying a valid duration for each uncontrollable action is valid for the projected problem of P. This captures the intuitive notion of strong plan: regardless of the actual duration of each uncontrollable action specified in the plan, the execution is valid and all the goals are satisfied.

Definition 9 (STPUD Plan Validity). *Given a STPUD problem P, a plan $\pi \doteq \{\langle t_i, a_i, d_i \rangle \mid i \in [1, n]\}$ is valid, if all the plans $\pi' \in \{\langle t, a, d \rangle \mid \langle t, a, d \rangle \in \pi, a \in A_c\} \cup \{\langle t, a, k \rangle \mid \langle t, a, d \rangle \in \pi, a \in A_u, a \doteq \langle [l, u], C, E \rangle, k \in [l, u]\}$ are valid for $ctrl(P)$.*

A valid plan for a given STPUD is called a "strong plan".

3.4. Discussion

In the general case, finding a strong plan for a problem with uncontrollable durations is different from simply considering the maximum or the minimum duration for each action. Consider our rover example and its strong plan shown in Figure 2. The *move* action must end before the transmit action can start and, at the same time, *move* cannot end before time 15 due to the temperature constraint. If we only consider the lower-bound on the duration of *move* (i.e., planning with a fixed duration of 10 for *move*) then one valid plan is: $\pi_{lb} \doteq \{\langle 11, move \rangle, \langle 22, trans \rangle\}$. However, because of the uncertainty in the actual execution duration of *move*, it may actually take 14 time units to arrive at l_2. Thus, the rover would start transmitting at time 22 before it actually arrives at l_2 at time $11 + 14 = 25$. Thus, plan π_{lb} is not a valid strong plan. Similarly, if we consider only the maximal duration (i.e., planning with a fixed duration of 15), then one possible plan would be: $\pi_{ub} \doteq \{\langle 1, move \rangle, \langle 22, trans \rangle\}$. However, the actual execution of *move*, may take only 11 time units (and not the planned maximum of 15 time units) to arrive at l_2. This would violate the constraint that the rover should arrive at l_2 after $t = 15$ to avoid the sun, so π_{ub} is also not a valid plan.

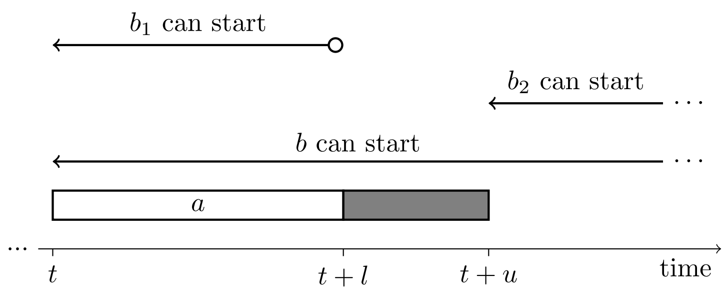


Figure 3: Example of situation where the classical compilation of disjunctive preconditions fails in presence of temporal uncertainty. Intervals in which b-actions can start are represented by solid lines: an arrow extreme indicates an open interval, a circle indicates a closed interval.

Disjunctive Conditions. In contrast to ordinary temporal planning, given a STPUD problem it is not possible to compile away disjunctive conditions using the action duplication technique [35]. This is because the set of satisfied disjuncts in the presence of uncertainty can depend on the contingent execution. For example, consider the situation depicted in Figure 3. An action a is starting at time t where two Boolean variables p_1 and p_2 are $\mathtt{F}$. The action a has uncontrollable duration in $[l, u]$, a starting effect $e_1 \doteq \langle [st_a]\, p_1 := \mathtt{T} \rangle$ and two ending effects $e_2 \doteq \langle [et_a]\, p_1 := \mathtt{F} \rangle$ and $e_3 \doteq \langle [et_a]\, p_2 := \mathtt{T} \rangle$. An at-start condition $p_1 \vee p_2$ of another action b is satisfied anywhere between the start of the action a and the next deletion of p_2. Thus, b can start anytime after t. However, by applying the compilation on this disjunctive condition we replace b with two actions b_1 and b_2, one with an at-start condition p_1 and the other with an at-start condition p_2. Now, b_1 is not executable within $(t+l, t+u]$ because there is no time point in d in which we can guarantee that $p_1 = \mathtt{T}$ (because a may take the minimum duration l and thus the at-end effect e_2 will occur at $t+l$ to set $p_1 = \mathtt{F}$). Similarly, we cannot start b_2 within $(t+l, t+u]$ because there is no guarantee that p_2 will be $\mathtt{T}$ during $(t+l, t+u]$ (this is because a may take the maximum duration u and thus e_3 that sets $p_2 = \mathtt{T}$ will not happen until $t+u$). Thus, compiling away disjunctive conditions as in temporal planning leads to incompleteness (some valid plans cannot be found) when actions with uncontrollable duration are present. For this reason, it is important to explicitly model disjunctive conditions in our language.

Computational Complexity. Considering decidability and computational complexity of the STPUD problem, we first note that the domain of the variables in a STPUD problem may be finite or infinite, but only finitely-many values from each domain are used in a problem instance, because we only allow for the equality relation in effects and conditions. So, each problem instance can be polynomially transformed, by means of a logarithmic encoding of the relevant values of each domain, into an equivalent one that only has Boolean variables. In this way, we can meet the general framework of Rintanen [36] for temporal planning.

The STPUD problem is decidable: this paper presents sound-and-complete solution algorithms. Concerning the computational complexity of STPUD, we present the following results.

Theorem 1. *The STPUD problem is EXPSPACE-Hard.*

Proof. (Sketch) STPUD trivially subsumes temporal planning (any temporal planning instance is just a STPUD instance with no action having uncontrollable duration). Because temporal planning is EXPSPACE-Complete [36], the STPUD problem is EXPSPACE-Hard. □

Theorem 2. *The STPUD problem is in 2-EXPSPACE.*

Proof. (Sketch) Any STPUD problem instance can be reduced to a corresponding temporal planning problem without uncontrollable durations of exponential size (see for example the compilation we present in Section 6). Since temporal planning without uncontrollable durations is EXPSPACE-Complete [36], the STPUD problem is in 2-EXPSPACE. □

We currently have no proof of membership in EXPSPACE nor a proof of 2-EXPSPACE-Hardness; we only know that the STPUD problem lies in between these two classes. We leave a detailed analysis of the computational complexity of the STPUD problem and its sub-classes for future work. In this paper we concentrate on practical solution techniques for the problem.

4. Overview of the Approaches

Given the problem classification in Table 1, here we give an outline of the techniques included in this paper that have been developed to address the more general Unc-Extr and Unc-Arbit class.

First, we focus on the Unc-Extr class and in Section 5 we present a dedicated solving technique that extends the Forward State-Space Temporal Planning (FSSTP) approach to deal with uncontrollable durations. We indicate this technique as S-FSSTP (for Strong FSSTP). We propose different variants of the technique that are sound for the Unc-Extr problem class; in addition, we prove that one of the approaches is also complete for the same problem class.

Second, we devise a compilation technique that transforms any instance of the Unc-Arbit class into an instance of the Ctrl-Arbit class, effectively removing the temporal uncertainty from the problem. The compilation is such that any plan of the controllable instance admits a plan if and only if the original Unc-Arbit planning problem has a valid strong plan. This compilation, discussed in Section 6, makes use of arbitrary-time conditions and effects; hence, even if applied on a Unc-Extr instance, it produces an equivalent Ctrl-Arbit instance.

We also present a simplification technique that is able to reduce (and in some cases, to remove) the temporal uncertainty in a planning instance. This

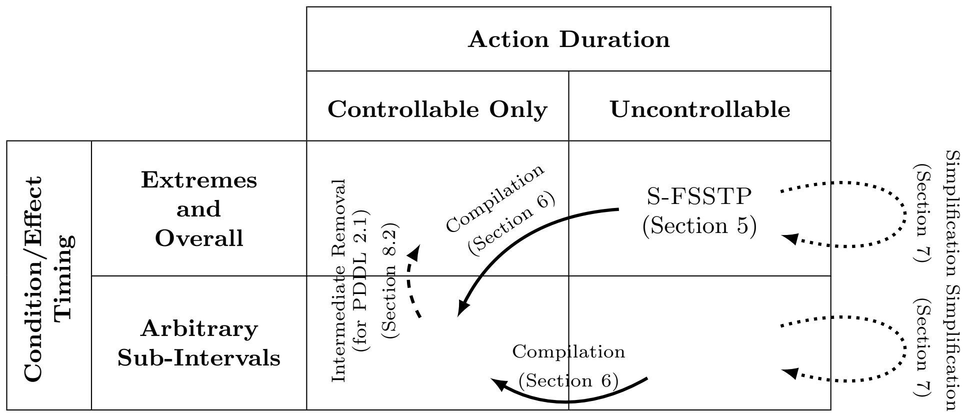


Table 2: Overview of the proposed techniques in the Strong Temporal Planning with Uncontrollable Durations problem sub-classes landscape. Arrows stand for compilation techniques, S-FSSTP indicates a dedicated technique for its cell. For each technique, the section where it is discussed is reported.

technique does not make use of intermediate effects nor conditions, hence it can be applied on both the Unc-Arbit and Unc-Extr classes without ending up in a different problem class.

Finally, in the experimental evaluation section (Section 8), we adapt existing techniques for removing intermediate conditions and effects in the context of PDDL 2.1. We discuss and extend existing techniques to obtain an efficient compilation for the removal of intermediate effects and conditions when no actions having uncontrollable duration are present.

Table 2 gives an overview of the aforementioned techniques and reductions in the problem classes landscape.

5. STPUD via Forward State-Space Search

In this section, we focus on the Unc-Extr problem class: for each action a, effects can only be specified at times st_a and et_a, and conditions are limited to timings $[st_a]$, $[et_a]$ or (st_a, et_a). We propose a dedicated approach for solving the STPUD problem, based on an extension of the Forward State-Space Temporal Planning (FSSTP) approach. We first introduce the relevant background on Temporal Networks with Uncertainty and we formalize the Forward State-Space Temporal Planning (FSSTP) approach, then we generalize it to handle temporal uncertainty. The resulting technique, that handles the Unc-Extr problem class, is referred to as S-FSSTP.

5.1. FSSTP Background

5.1.1. Temporal Networks with Uncertainty

In temporal planning, we need to reason not only about the ordering of actions in time, but also about their duration. A common framework to represent and reason about these kinds of constraints in scheduling (i.e. when the set of activities is known and only a suitable timing for the activities is sought) is the Temporal Network (TN) [10, 37]. The TN formalism is used to represent temporal constraints over time-valued variables representing time points. Each constraint is a disjunction of atoms in the form $x - y \in [l, u]$, where x and y are time points and $l, u \in \mathbb{R} \cup \{\infty, -\infty\}$. Formally, a TN is a tuple $\langle X, C\rangle$ where X is a set of time points and C is a set of temporal constraints. Time points are typically used to represent the start or the end of actions; causal relations are represented as precedence constraints, forcing one event to happen before another; and metric constraints allow for the encoding of action durations by constraining the difference between start and end of an action. A solution to a TN is an assignment of real values to the time points that satisfies all the constraints. A TN is said to be consistent if it has a solution.

In order to deal with temporal uncontrollability, TNs with uncertainty (TNUs) have been proposed [4, 5].

Definition 10 (Temporal Network with Uncertainty)**.** *A Temporal Network with Uncertainty (TNU) is a tuple $\langle X_c, X_u, C_c, C_f\rangle$, where X_c and X_u are sets of time points; $C_c \doteq \{cc_1, \cdots, cc_m\}$ is a set of constraints of the form $cc_i \doteq \bigvee_j x_i - y_i \in [l_{i,j}, u_{i,j}]$ with $x_i \in X_u$, $y_i \in X_c \cup X_u$ and $l_{i,j}, u_{i,j} \in \mathbb{R} \cup \{\infty, -\infty\}$; and $C_f \doteq \{cf_1, \cdots, cf_n\}$ is a set of constraints of the form $cf_i \doteq \bigvee_j x_{i,j} - y_{i,j} \in [l_{i,j}, u_{i,j}]$ with $x_{i,j}, y_{i,j} \in X_c \cup X_u$ and $l_{i,j}, u_{i,j} \in \mathbb{R} \cup \{\infty, -\infty\}$.*

In a TNU some of the time points can be controlled (assigned values) by the solver (X_c), while the others (X_u) are controlled (assigned values) by the environment. Similarly, the constraints are divided according to requirements (called free constraints, C_f) and assumptions (called contingent constraints, C_c). As such, TNUs can be seen as a form of game between the solver and an adversarial Nature [11].

Given a TNU, different kinds of queries are possible. In this paper, we focus on strong controllability [4, 5]. A TNU is strongly controllable if there exists an assignment (called a *strong schedule*) of real values to each controllable time point, such that all free constraints are satisfied for every possible assignment of the uncontrollable time points satisfying the contingent constraints.

Depending on the structure of the constraints, various classes of TNUs have been identified [5]. We focus on two classes of TNUs: Simple Temporal Networks with Uncertainty (STNUs) and Disjunctive Temporal Networks with Uncertainty (DTNUs). An STNU is a TNU where each constraint has exactly one disjunct (i.e. it is conjunctive), while DTNUs allow for arbitrary Boolean combinations in the constraints.

Several approaches to check strong controllability of a TNU have been proposed. In [4], the authors show that the strong controllability problem for an STNU is polynomial-time, while for a DTNU it is NP-hard [5]. Recently, new techniques to solve strong controllability for DTNUs have been presented [11]: they rely on the reduction of strong controllability to a Satisfiability Modulo Theory [12] problem.

5.1.2. FSSTP

The idea behind the FSSTP approach is an interplay between a state-based forward search planner to generate an abstract plan and a temporal reasoner to check its temporal feasibility [7, 9]. Intuitively, a temporal plan is a set of action instances scheduled at specific times with specific durations. FSSTP works by encoding each durative action as a pair of instantaneous, classical actions: a classical planner is employed to generate sequences of such instantaneous action instances that are sound from the propositional point of view, but might violate some temporal constraint. Then, a scheduler is used to check temporal feasibility and to associate a proper time-stamp to each action instance. If the scheduling succeeds, a valid temporal plan is constructed, otherwise the sequence of classical action instances is refused and another must be found.

Most existing FSSTP planners are designed to operate on the PDDL 2.1 language. PDDL only allows effects at the start (st_a) or at the end (et_a) of an action, instantaneous conditions can be specified at the start or at the end of the action and durative conditions are possible only on the whole interval (st_a, et_a). This coincides with the Ctrl-Extr problem class we defined in Section 4.

In FSSTP, each durative action a is expanded into a pair of classical planning actions called *snap actions*: a_{st} encoding the starting event of a, and a_{et} corresponding to the ending of a. The action a_{st} has the starting conditions and effects of a as preconditions and effects, and, similarly, a_{et} has the ending conditions and effects of a as preconditions and effects. FSSTP planners like Colin force the planner to instantiate snap actions in pairs (each start is coupled with exactly one end) and forbid any action threatening the overall condition of a between the snap actions for a [7].

Similarly, timed initial literals are treated as instantaneous actions that can be instantiated without preconditions and have the TIL as effect. For a TIL t, we indicate with $TA(t)$ such an instantaneous action.

We can now define a Domain Abstraction as a classical planning problem derived from a Ctrl-Extr STPUD planning instance.

Definition 11 (Domain Abstraction)**.** *Given a temporal planning problem $P = \langle V, I, G, T, A\rangle$ restricted to the* Ctrl-Extr *problem class, the abstraction of P (written $abs(P)$) is a classical (non-temporal) planning problem defined as $\langle V, I, G, \bigcup_{a\in A}\{a_{st}, a_{et}\} \cup \{TA(t) \mid t \in T\}\rangle$.*

The (highly simplified) pseudo-code of a forward state-space temporal planner (FSSTP) is shown in Algorithm 1. A classical planner (implemented as a forward state-space search) is used as an iterator[8] over all the valid plans of the abstract problem $abs(P)$. The planner keeps a totally-ordered partial plan χ composed of abstract action instances we call steps. We indicate a step corresponding to an instance of action a as s^a. Each time an action instance is added to the partial plan, the scheduling check is invoked to assess the temporal

[8]This behavior can be efficiently implemented in a forward state-space search planner by keeping the open and closed sets between calls to avoid the need for plan blocking constraints.

Algorithm 1 The FSSTP Approach

```
1: procedure FSSTP($P$)
2:     for all partial $\chi$ generated while solving $abs(P)$ do
3:         $D \leftarrow$ DURATIONS($\chi$, $P$)
4:         $P \leftarrow$ PRECEDENCES($\chi$, $P$)
5:         if TN.ISCONSISTENT($\langle \chi, D \cup P \rangle$) then
6:             $\mu \leftarrow$ TN.CONSISTENTSCHEDULE($\langle \chi, D \cup P \rangle$)
7:             if ISCOMPLETE($\chi$) then
8:                 return BUILDTEMPORALPLAN($\mu$, $\chi$)
9:             else
10:                CONTINUE( )
11:        else
12:            REJECT($\chi$)
13:    return $\perp$
```

consistency of the partial plan. The scheduling check builds a TN[9] that has the steps of χ as time points and has a set of constraints composed of duration constraints D and precedence constraints P. Duration constraints (created by the DURATIONS function) are used to bind pairs of snap action instances $(s_i^{a_{st}}, s_j^{a_{et}})$, forcing the duration of each action instance to obey the domain specification ($a_{et} - a_{st} \in [l, u]$, being $a \doteq \langle [l, u], C, E \rangle$). In addition, each TIL $t \doteq \langle [k]\, f := v \rangle$ is forced to happen at the predefined time by imposing a temporal constraint[10] $TA(t) = k$. Precedence constraints (created by the PRECEDENCES function) are used to maintain causality in the plan. If a step s_i is needed to achieve a precondition for another step s_j, we must impose a precedence among the two steps ($s_j - s_i > 0$), in order to inform the scheduler of the causal constraint[11]. Similarly, precedence constraints are used to ensure that the overall conditions for an action a are maintained.

When the TN is found to be consistent, two situations can occur. If χ is a plan achieving the goal in $abs(P)$, each $s_i^{a_{st}}$ is followed by a corresponding $s_j^{a_{et}}$ and all TILs appear in the plan (ISCOMPLETE returns true), we can terminate the procedure, otherwise we continue the search in the abstract domain (CONTINUE). To terminate, we build a temporal plan π from a consistent schedule μ of the TN: we write $\mu(x)$ to indicate the value assigned to x by μ. Each pair of snap actions steps $s_i^{a_{st}}$, $s_j^{a_{et}}$ in χ is a step in π, the time for the step is $\mu(s_i^{a_{st}})$ and the duration is $\mu(s_j^{a_{et}}) - \mu(s_i^{a_{st}})$. Instead, if the TN is not consistent, the classical planner is required to generate a new plan, as χ is not temporally sound and cannot be further extended.

[9]In this work, we only consider purely temporal planning. Other works cope with numeric fluents and continuous effects by using linear programs instead of Temporal Networks [9].

[10]In practice, we introduce a reference time point z marking the beginning of time and we impose a proper TN constraint $TA(t) - z \in [k, k]$

[11]In the PDDL 2.1 semantics, interfering steps must be separated by at least ϵ time; therefore, one would impose the constraint $b - a \geq \epsilon$.

Algorithm 2 The S-FSSTP Approach

```
1: procedure S-FSSTP($P$)
2:     for all partial $\chi$ generated while solving $abstract(P)$ do
3:         $X_u \leftarrow$ UncontrollableSteps($\chi$, $P$)
4:         $D_c \leftarrow$ ControllableDurations($\chi$, $P$)
5:         $D_u \leftarrow$ UncontrollableDurations($\chi$, $P$)
6:         $P \leftarrow$ Precedences($\chi$, $P$)
7:         if TNU.IsStrongControllable($\langle \chi \setminus X_u, X_u, D_u, D_c \cup P \rangle$) then
8:             $\mu \leftarrow$ TNU.StrongSchedule($\langle \chi \setminus X_u, X_u, D_u, D_c \cup P \rangle$)
9:             if IsComplete($\chi$) then
10:                return BuildStrongPlan($\mu$, $\chi$)
11:            else
12:                continue( )
13:        else
14:            reject($\chi$)
15:    return $\bot$
```

In the rest of this section, we use the following notation. Given a step s_i of χ (that is an instance of a classical planning action), we write *effects*(s_i) to indicate the set of its effects, each of the form $\langle f := v \rangle$. Moreover, we write *conditions*(s_i) to indicate the set of preconditions of s_i, each in the form $\bigvee_{i=1}^{n} f_i = v_i$.

5.2. S-FSSTP

The general idea we pursue is to substitute the scheduling steps of Algorithm 1 to solve a strong controllability problem for a TNU instead of consistency for a TN. The generalization of the framework to the STPUD case is shown in Algorithm 2.

As in FSSTP, we first consider the abstract domain enumerating the "discrete" plans that are a solution of the abstract problem. The key difference with respect to the FSSTP schema is that to accommodate uncontrollable durations we substitute the TN with a TNU (checking strong controllability instead of consistency), and we consider different formulations for the precedence constraints.

Thanks to the limitation to the Unc-Extr class of problems, we can consider all the time points corresponding to the starting of actions as controllable (the planner decides if and when an action should be started), while ending time points are controllable if the corresponding action is controllable, otherwise they are uncontrollable.

The duration constraints are built analogously to the plain temporal case, but are divided in two sets: D_c are the duration constraints for controllable actions, D_u are the ones for uncontrollable actions.

Building a strong plan σ from a strong schedule for the TNU is analogous to the plain temporal planning case: each pair of corresponding $\langle s_i^{a_{st}}, s_j^{a_{et}} \rangle \in \chi$ is a step of σ, the time for the step is $\mu(s_i^{a_{st}})$ and, if a is controllable, the duration is

$\mu(s_j^{a_{et}}) - \mu(s_j^{a_{st}})$. We do not set the duration for uncontrollable durative action instances.

The encoding of the precedence constraints $Precedences(\chi, P)$ is crucial, because in the presence of uncontrollability not all the techniques presented in the temporal planning literature for the controllable case [9, 8] are complete. In the following, we consider two different encodings proposed in the temporal planning literature and we show that they are incomplete for solving the STPUD problem. Then, we borrow the idea of reordering from [6] and we derive the first sound and complete approach for the STPUD problem in the Unc-Extr class.

5.2.1. Total Order Encoding

A simple way of building the ordering constraints is to maintain the total order (TO) of the partial plan χ. Forcing this total order clearly maintains the causal soundness but, as noted in [8], is heavily dependent on the order of action instances chosen by the classical planner. Nevertheless, this encoding is complete for temporal planning without duration uncontrollability and is adopted in the Colin [9] and Crikey 3 [7] planners.

We call $Precedences_{TO}(\chi, P)$ the set of precedence constraints for a given totally ordered plan $\chi \doteq \langle s_1, \dots, s_n \rangle$, and we define it as follows:

$$Precedences_{TO}(\chi, P) \doteq \{s_i < s_{i+1} \mid i \in [1, n-1]\} .$$

We note that no disjunction is created, hence the encoding results in an STNU[12]. Unfortunately, this encoding is incomplete in the presence of temporal uncertainty. Indeed, at each step of Algorithm 2, it might be the case that no total order produces a strongly controllable TNU, even if there exists a strong plan for the given problem. Thus, the planner can explore the complete search space and declare the problem unsolvable even if there exists a solution. Nonetheless, the approach is sound: if a solution is returned, it is a valid strong plan.

In our running example, reported in Section 3.2, the TO approach can terminate yielding the strong plan π_{ex} when the following abstract plan is generated by the classical planner and the relative total order is considered.

$$\chi_{ex} \doteq \langle s_1^{move_{st}}, s_2^{visible:=\mathtt{T}}, s_3^{hot:=\mathtt{F}}, s_4^{move_{et}}, s_5^{trans_{st}}, s_6^{trans_{et}}, s_7^{visible:=\mathtt{F}} \rangle$$

This very same example of an abstract plan that works for finding the plan π_{ex} also works in the other approaches in this section.

As an example of the approach incompleteness, consider the situation depicted in Figure 4. Suppose that both actions a and b must be started at the same time[13]. Action a is uncontrollable and b must end between the earliest and the latest possible ends of a. Literals p and q are initially true and

[12]Formally, strict inequalities are not expressible in STNU, but the techniques in [11] support this extension. In a PDDL 2.1 context we can exploit the minimum quantum ϵ and rewrite $x > y$ as $x + \epsilon \geq y$.

[13]We just need that the end of b is forced to overlap with the interval in which a can uncontrollably end.

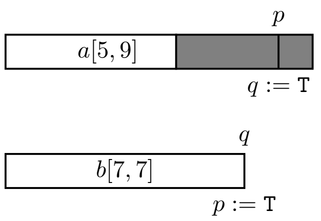


Figure 4: Example problem for which the TO and LAD encodings cannot find a plan, while a strong plan exists. The propositions p and q are both initially true, and if a and b must start at the same time, b will end between the earliest and latest end times for a.

no action falsifies them. Let us focus on the relative order of $s_i^{a_{et}}$ and $s_j^{b_{et}}$. If $\chi = \langle \cdots, s_i^{a_{et}}, \cdots, s_j^{b_{et}}, \cdots \rangle$, then we (transitively) impose the constraint $s_i^{a_{et}} < s_j^{b_{et}}$. However, the resulting STNU is not strongly controllable because if a takes longer than 7 time units, a_{et} can happen after b_{et}, violating the constraint. If $\chi = \langle \cdots, s_j^{b_{et}}, \cdots, s_i^{a_{et}}, \cdots \rangle$, then the situation is reversed and again the STNU is not strongly controllable. Therefore, in both cases χ is rejected and the planner returns $\bot$. This is incomplete, because there exists a simple strong plan for the problem: start both actions at time 0 (the two actions are non-interfering and all the conditions are satisfied as p and q are never falsified).

5.2.2. Last Achiever Deordering Encoding

Another possible precedence encoding lifts totally ordered plans to partially ordered plans [8]. The underlying idea is to use the greedy algorithm proposed in [38] to reconstruct the causal links as precedence links. For each action instance in the plan requiring a literal l as precondition, the algorithm searches for the last achiever of that literal in the totally ordered plan, and imposes a precedence link between the two action instances. In this way, it builds a partial order plan as a deordering [6] of χ and possibly reduces the commitment on the specific input ordering. Note that, similar to the previous encoding, we never introduce disjunctions, hence the resulting TNU is an STNU[14].

We now define the Last Achiever Deordering (LAD). Using a common trick in partial order planning, we consider two fictitious steps, s_0 and s_{n+1}, representing the initial state and the goal condition, respectively. Step s_0 has no preconditions and has the initial state I as effect. Step s_{n+1} has the goal as precondition and no effect.

Given a variable f and a value v, we denote with $ach(f, v)$ the subset of steps in χ that set the variable f to value v and with $del(f, v)$ the set of action

[14]Compared to [8], we expanded the precedence constraints to cope with disjunctive conditions in our language. In particular, we consider the last achiever of at least one disjunct of any step and we ensure that the overall condition of each action is respected on at least one disjunct.

instances that set f to a value different from v. Formally, $ach(f, v)$ is defined as the set of steps $\{s_i \in \chi | \langle f := v \rangle \in \mathit{effects}(s_i)\}$ and $del(f, v)$ as the set $\bigcup_{v' \in Dom(f), v' \neq v} ach(f, v')$.

Given a step s_k and a disjunctive condition $c \doteq \bigvee_{i=1}^{n} f_i = v_i$, we denote with $last(c, s_k)$ the step s_j such that $s_j \in \bigcup_{i=1}^{n} ach(f_i, v_i)$ and j is the maximum index strictly lower than k. Intuitively, $last(c, s_k)$ is the last action instance that sets an f_i to its desired value v_i before s_k in χ.

The precedence constraints $Precedences_{LAD}(\chi, P)$ of this approach are defined as follows.

Definition 12. *Given $\chi = \langle s_0, \cdots, s_{n+1} \rangle$, $Precedences_{LAD}(\chi, P)$ is as follows.*

1. *$\{(s_0 < s_i), (s_i < s_{n+1}) \mid i \in [1, n]\} \subseteq Precedences_{LAD}(\chi, P)$.*
2. *For each $s_k^a \in \chi$ and for each $c \in conditions(a)$, $(last(c, s_k^a) < s_k^a) \in Precedences_{LAD}(\chi, P)$.*
3. *For each $s_k^{a_{st}} \in \chi$ and for each overall condition $\langle (st_a, et_a) \bigvee_{i=1}^{n} f_i = v_i \rangle$ of action a, $\{(s_j < s_k^{a_{st}}) \mid s_j \in \bigcup_{i=1}^{n} del(f_i, v_i), j < k\} \subseteq Precedences_{LAD}(\chi, P)$.*
4. *For each $s_k^{a_{et}} \in \chi$ and for each overall condition $\langle (st_a, et_a) \bigvee_{i=1}^{n} f_i = v_i \rangle$ of action a, $\{(s_k^{a_{et}} < s_j) \mid s_j \in \bigcup_{i=1}^{n} del(f_i, v_i), k < j\} \subseteq Precedences_{LAD}(\chi, P)$.*

For example, consider the following abstract plan for the running example of Section 3.2.

$$\chi_{ex}^2 \doteq \langle s_1^{move_{st}}, s_3^{hot:=\mathtt{F}}, s_4^{move_{et}}, s_2^{visible:=\mathtt{T}}, s_5^{trans_{st}}, s_6^{trans_{et}}, s_7^{visible:=\mathtt{F}} \rangle$$

The abstract plan is the same as χ_{ex} defined in the previous section, but we shuffled the positions of $s_2^{visible:=\mathtt{T}}$, $s_3^{hot:=\mathtt{F}}$ and $s_4^{move_{et}}$. For this abstract plan, the Total Order approach would construct an STNU that is not strongly controllable (nor Consistent) because the $visible := \mathtt{T}$ TIL must happen before the $hot := \mathtt{F}$ one. However, there is no causal relation between the two TILs (they operate on different variables) nor between the changes to the hot variable and the starting of the $trans$ action. In this sense, the algorithm in [38] would not introduce a precedence requirement from $s_3^{hot:=\mathtt{F}}$ to $s_2^{visible:=\mathtt{T}}$, nor from $s_4^{move_{et}}$ to $s_2^{visible:=\mathtt{T}}$; leaving to the scheduler the task of finding a suitable order. In this view, the example abstract plan would correctly generate the plan π_{ex}. In Figure 5, we show the STNU produced by the LAD approach.

This encoding is able to find a plan in many situations even in the presence of uncertainty, but it is not complete in general. For example, it fails on the problem of Figure 4: the encoding greedily assumes that the last achiever is the one that must be preserved in the form of a causal link; in reality there may be other achievers that could be used instead. Just as in the previous case, if $\chi = \langle \cdots, s_i^{a_{et}}, \cdots, s_j^{b_{et}}, \cdots \rangle$, then we impose the constraint $s_i^{a_{et} < b_{et}}$ because $s_i^{a_{et}}$ is the last achiever of p (required by $s_j^{b_{et}}$). Instead, if $\chi = \langle \cdots, s_j^{b_{et}}, \cdots, s_i^{a_{et}}, \cdots \rangle$, we impose the constraint $s_j^{b_{et}} < s_i^{a_{et}}$ because $s_j^{b_{et}}$ is the last achiever of q (required by $s_i^{a_{et}}$).

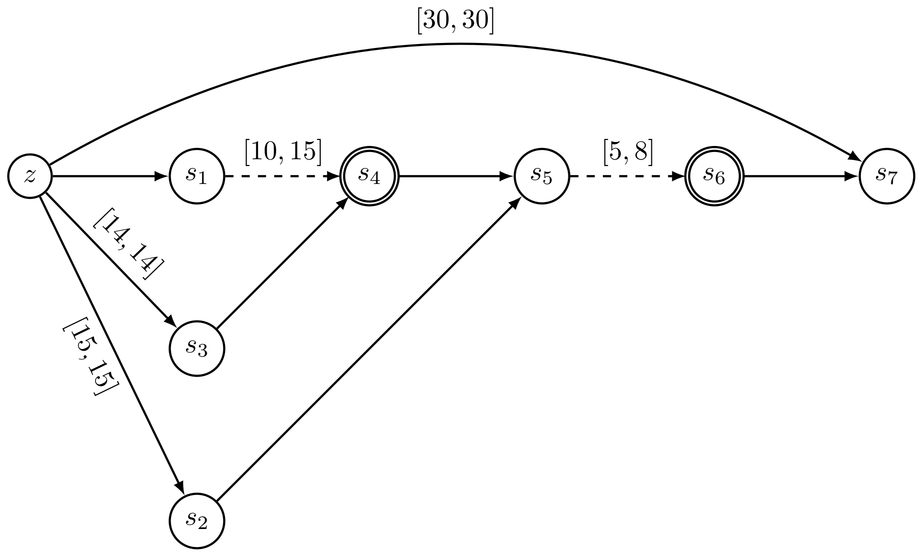


Figure 5: The STNU induced by the LAD approach on the χ^2_{ex} abstract plan. Nodes represent time points and edges are the constraints. In the picture, doubly circled nodes are uncontrollable, dashed edges are contingent constraints and unlabeled edges are precedence constraints (the implicit label would be $[0, \infty)$). The resulting STNU is strongly controllable, with π_{ex} being a valid strong schedule.

5.2.3. Disjunctive Reordering Encoding

In order to obtain sound and complete reasoning, we need to relax the total order produced by the state-space search, retaining the precedence constraints needed to ensure plan validity. However, we must be careful to not over-constrain the TNU, otherwise we may discard valid plans. A solution is to consider all the reorderings [6] of the given plan that are causally sound: we build a set of (disjunctive) precedence constraints in such a way that all the orderings fulfilling the constraints are causally sound. We call the approach using these precedence constraints *Disjunctive Reordering* (DR) and formally define it below. We show that, given a partial plan χ, using DR to construct the precedences in Algorithm 2, yields a complete technique for STPUD.

Given a variable f, a value v and a pair of action instances a and r of χ, we define the disjunctive temporal constraint $\rho(f, v, a, r)$ as follows.

$$\rho(f, v, a, r) \doteq (a < r \wedge \bigwedge_{s_i \in ach(f,v) \setminus \{a,r\}} (s_i < a \vee s_i > r))$$

Intuitively, for a condition $c \doteq \langle f = v \rangle$, if a is an achiever of c and r is a step having c as precondition, $\rho(f, v, a, r)$ holds if a was the last achiever of c before r. We now define the precedence constraints induced by the DR approach, indicated as $Precedences_{DR}(\chi, P)$. As before, s_0 and s_{n+1}, represent the initial state and the goal condition, respectively.

Definition 13. *Given $\chi = \langle s_0, \cdots, s_{n+1} \rangle$, $Precedences_{DR}(\chi, P)$ is as follows.*

1. $\{(s_0 < s_i), (s_i < s_{n+1}) \mid i \in [1, n]\} \subseteq Precedences_{DR}(\chi, P)$.
2. *For each $s_k^a \in \chi$ and for each $\bigvee_{i=1}^n f_i = v_i \in conditions(a)$, the following constraints belong to $Precedences_{DR}(\chi, P)$:*

   (a) $\bigvee_{i=1}^n \bigvee_{s_j \in ach(f_i, v_i) \setminus \{s_k^a\}} \rho(f_i, v_i, s_j, s_k^a)$;

   (b) $\bigvee_{i=1}^n (\bigwedge_{s_j \in ach(f_i, v_i)} (\rho(f_i, v_i, s_j, s_k^a) \rightarrow \bigwedge_{s_t \in del(f_i, v_i) \setminus \{s_k^a\}} (s_t < s_j \vee s_t > s_k^a)))$.
3. *For each $s_k^{a_{st}} \in \chi$ and its corresponding $s_w^{a_{et}} \in \chi$ and for each overall condition $\langle (st_a, et_a) \bigvee_{i=1}^n f_i = v_i \rangle$ of the action a,*
   $\bigvee_{i=1}^n \bigwedge_{s_j \in \bigcup_{i=1}^n del(f_i, v_i)} ((s_j < s_k^{a_{st}}) \vee (s_j > s_w^{a_{et}})$.

Intuitively, constraint 2a says that at least one step s_j having as effect one of the disjuncts of the precondition of step s_k^a occurs before s_k^a. Constraint 2b says that there is at least one precondition disjunct $c \doteq \langle f = v \rangle$ of action a such that if s_j is the last achiever for c, between s_j and s_k^a there must be no action instance falsifying c. Overall conditions cannot be canceled between their extremal time point markers (constraint 3).

The following theorem states that DR is sound and complete. We give the full proof of the theorem in Appendix A.

Theorem 3 (DR Completeness)**.** *Given a STPUD problem P admitting a valid strong plan σ, if DR is used, Algorithm 2 terminates with a valid strong plan.*

The intuition is that in DR, the disjunctions encode all reorderings that are causally sound in the form of a DTNU, allowing the scheduler to re-arrange the action instances independently of the total ordering of χ. In fact, the precedence constraints generated by this approach are independent of the order of the steps in χ; practically χ is treated as a multi-set. Therefore, it is possible that this approach will generate and reject the same DTNU multiple times. In principle, it would be possible to cache these DTNUs and require that the underlying plan enumeration avoid the same multi-set of actions directly, but we leave these technical considerations and improvements for future work.

We note that the set of precedence constraints generated by the DR approach, conjoined with the duration constraints, yields a TNU that is not formally a DTNU. This is because of the use of strict inequalities and negations that are not expressible in the DTNU framework. However, if we take the PDDL 2.1 semantics, this is not a problem because this semantics prescribes that there is always a minimal time quantum (called ϵ) that is required to separate two steps: each constraint $s_i > s_j$ can then be rewritten as $s_i - s_j \geq \epsilon$, and negations can be handled by reversing the inequalities (e.g. $\neg(s_i > s_j)$ is equivalent to $(s_i \leq s_j)$). Therefore, we can encode these constraints as a proper DTNU. Moreover, we remark that the strong controllability techniques presented in [11] are applicable even in the presence of strict inequalities.

The strong controllability of a DTNU is an NP-Hard problem [5] (and the same is true for the generalized DTNU with strict inequalities), thus the use of this encoding is quite costly; however, DR is important as it overcomes the incompleteness limitation of the other encodings.

6. Compiling STPUD into Temporal Planning

In this section, we present our compilation technique, which can be used to reduce any planning instance P having duration uncertainty into a temporal planning instance P' in which all actions have controllable durations. The translation guarantees that P is solvable if and only if P' is solvable, and it fully supports the Unc-Arbit problem class. Moreover, given any plan for P' we can derive a plan for P. This approach comes at the cost of duplicating some of the variables in the domain, but allows for the use of off-the-shelf temporal planners.

6.1. Formal Compilation

The intuition behind the translation is that we are representing the uncertainty about duration as uncertainty about the values of variables during certain action intervals. In practice, we use additional variables to encode in a single execution all the possible uncertain executions of the STPUD. In a sense, this is analogous to the expansion of a universal quantifier in logic: we make sure that all the possible realizations of the quantified variable are satisfied by a single model.

Consider for example the *transmit* (i.e., *trans*) action in our example, and suppose it is scheduled to start at time k. Let v be the value of *sent* at time $k + 5$; since *transmit* has an at-end effect $\langle [et_{trans}]\ sent := \mathtt{T} \rangle$, we know that the value of the variable *sent* during the interval $(k + 5, k + 8]$ will be either v or $\mathtt{T}$ depending on the duration of the action. After time $k + 8$ we are sure that the effect took place, and we are sure of the value of *sent* until another effect is applied[15]. Since in STPUD the plan cannot take advantage of observed durations at run-time, we need to consider the uncertainty in the value of *sent* and produce a plan that works for all the possible uncertain values. Since *sent* could appear as a condition of another action (or as a goal condition, as in our example) we must rewrite such conditions to be true only if the value of *sent* is certain to be the required value, or if either value will satisfy the condition. To achieve this, we create an additional variable $sent_\sigma$ (called the *shadow variable of sent*). This secondary variable stores the alternative value of *sent* during uncertainty periods. When there is no uncertainty in the value of *sent*, both *sent* and $sent_\sigma$ will have the same value. In this way, all the conditions involving *sent* can be rewritten in terms of *sent* and $sent_\sigma$ to ensure they are satisfied by both the values.

[15]Note that there cannot be another concurrent action in the plan having an effect on *sent* during the interval $[k+5, k+8]$ because this would allow for the possibility of two concurrent effects on the same variable, which is forbidden in our semantics.

In general, our translation rewrites a STPUD problem $P \doteq \langle V, I, T, G, A\rangle$ into a new planning instance $P' \doteq \langle V', I', T', G', A'\rangle$ that does not contain actions with uncontrollable duration.

6.1.1. Uncertain Variables

The first step is to identify the set of variables $L \subseteq V$ that appear as effects of uncontrollable actions and are executed at a time depending on the end of the action.

$$L \doteq \{f \mid a \in A_u, \langle [t]\ f := v\rangle \in E_a, t = et_a - \delta\}$$

Intuitively, this is the set of variables that can possibly have uncertain value during plan execution. A variable that is modified only at times linked to the start of actions or by timed initial literals cannot be uncertain: neither the starting time of actions nor the timed initial literals can be uncertain in our model. In our running example, the set L is the set $\{sent, pos\}$.

We now define the set V' as the original variables V plus a shadow variable for each variable appearing in L.

$$V' \doteq V \cup \{f_\sigma \mid f \in L\}$$

We use the pair of variables f and f_σ to represent uncertainty: if $f = f_\sigma$ we know that there is no uncertainty in the value of f, while if $f \neq f_\sigma$ we know that the actual value of f in the original problem is either f or f_σ.

6.1.2. Disjunctive Conditions

At the end of Section 3, we outlined the reason why existing approaches for compiling away disjunctive conditions will not work with uncontrollable action durations. To rewrite a condition $c \doteq \langle [\![st_c, et_c]\!] \bigvee_{i=1}^{n} f_i = v_i\rangle$ we need to ensure that at each time in which the condition must hold, the set of disjuncts for at least one variable f_i in c is satisfied by the values of both f_i and $f_{i\sigma}$. When there is only one disjunct $f_i = v_i$ for the variable f_i, this requires that both f_i and $f_{i\sigma}$ have the value v_i. However, if c contains multiple disjuncts for a variable f_i, say $f_i = v_{i1} \vee f_i = v_{i2}$, this condition will be satisfied whenever both f_i and $f_{i\sigma}$ have values in the set $\{v_{i1}, v_{i2}\}$. To accomplish this, we define an auxiliary function $\theta(\psi)$ that takes a single disjunctive condition without timing information and returns a set of untimed disjunctive conditions.

$$\theta(\psi) \doteq \theta(\psi, L)$$

with

$$\theta(\psi, X) \doteq \begin{cases} \{\psi\} & \text{if } X = \emptyset \\ \theta(\psi, X') \cup \{\phi[f \to f_\sigma] \mid \phi \in \theta(\psi, X')\} & \text{if } X = \{f\} \cup X' \end{cases}$$

where $\phi[f \to f_\sigma]$ indicates logical substitution of variable f with variable f_σ in formula ϕ.

Using this, the condition of the *trans* action, $pos = l_2$, is translated as the two conditions $pos = l_2$ and $pos_\sigma = l_2$. Analogously, assuming that both f

and g are in L, a given condition $(f = \mathtt{T}) \vee (g = \mathtt{F})$ in P is translated by function θ as the set of conditions $\{(f = \mathtt{T}) \vee (g = \mathtt{F}),\ (f_\sigma = \mathtt{T}) \vee (g = \mathtt{F}), (f = \mathtt{T}) \vee (g_\sigma = \mathtt{F}), (f_\sigma = \mathtt{T}) \vee (g_\sigma = \mathtt{F})\}$ in P'. The intuition behind θ is as follows. Semantically, a condition is satisfied at time t if the evaluation of the variables at time t satisfies the condition. If some variables are uncertain, then the condition must be satisfied by all the possible values of the variables at time t. In our rewriting, we know that at each time the value of an uncertain variable f in the original problem is uncertain between the value of f and the value of f_σ in the rewritten problem. So, we must ensure that a condition is valid for every possible combination of both the f value and the f_σ value, and θ produces a set of conditions that exactly ensures this property.

6.1.3. Uncertain Temporal Intervals

We also need to identify the temporal interval in which the value of a given variable can be uncertain. Given an action a with uncertain duration d_a in $[l, u]$, let $\lambda(t)$ and $\nu(t)$ be the earliest and latest possible times at which an effect at $t \doteq et_{a'} - \delta$ may happen, i.e. $\lambda(t) \doteq st_{a'} + l - \delta$ and $\nu(t) \doteq st_{a'} + u - \delta$. Both functions are equal to $st_{a'} + \delta$ if $t \doteq st_a + \delta$.

For example, consider the effect $e_1 \doteq \langle [et_{trans}]\, sent := \mathtt{T}\rangle$ of action *trans*. We know that the duration of transmit is uncertain in $[5, 8]$, therefore the effect could occur at any time between $\lambda(et_{trans}) \doteq st_{trans'} + 5$ and $\nu(et_{trans}) \doteq st_{trans'} + 8$ and the *sent* variable has an uncertain value within that interval.

6.1.4. Uncontrollable Actions

For each uncontrollable action $a \doteq \langle [l, u], C_a, E_a\rangle)$ in A_u in the original model we create a new action $a' \doteq \langle [u], C_{a'}, E_{a'}\rangle$ in A'_c. Specifically, we first fix the maximal duration u as the only allowed duration for a' and then add appropriate effects and conditions *during* the action to capture the uncertainty.

The effects $E_{a'}$ are partitioned in two sets $E^l_{a'}$ and $E^u_{a'}$ to capture possible values within the uncertain action execution duration. The conditions $C_{a'}$ are also composed of two elements: the rewritten conditions $C^R_{a'}$ and the conditions added to protect the new effects $C^E_{a'}$ (thus $C_{a'} \doteq C^R_{a'} \cup C^E_{a'}$).

Rewritten conditions $C^R_{a'}$. Controllable conditions are compiled by rewriting existing action conditions by means of the θ function. The intervals specifying the duration of the conditions are preserved as they are written in the original model. However, since the action duration is now set to its maximum, the effective duration of conditions is "stretched" to match the maximal duration (this is because temporal anchors such as st_a and et_a are interpreted over the new maximal duration of the action).

$$C^R_{a'} \doteq \{\langle [\![\lambda(t_1), \nu(t_2)]\!]\ \alpha\rangle \mid \alpha \in \theta(\psi), \langle [\![t_1, t_2]\!]\ \psi\rangle \in C_a\}$$

Here we keep the interval type: for example a $[t_1, t_2)$ interval gets translated into $[\lambda(t_1), \nu(t_2))$.

For example, the set $C^R_{trans'}$ for the *trans* action is: $\{\langle [st_{trans'}, st_{trans'} + 8] pos = l_2\rangle, \langle [st_{trans'}, st_{trans'} + 8] pos_\sigma = l_2\rangle, \langle [st_{trans'}, st_{trans'} + 8] visible = \mathtt{T}\rangle\}$.

This requires variables $visible$, pos and pos_σ to be true throughout the execution of $trans'$.

Compiling action effects. The effects on variables in L of the original action are duplicated: both the affected variable f and its shadow f_σ are modified, but at different times. We first identify the earliest and latest possible times at which an effect can happen due to the duration uncertainty. We then apply the effect on f_σ at the earliest time point $\lambda(t)$, and at the latest time point $\nu(t)$ we re-align f and f_σ by also applying the effect on f:

$$E^l_{a'} \doteq \{\langle [\lambda(t)]\, f_\sigma := v \rangle \mid \langle [t]\, f := v\rangle \in E_a\}$$

$$E^u_{a'} \doteq \{\langle [\nu(t)]\, f := v \rangle \mid \langle [t]\, f := v\rangle \in E_a\}$$

For example, the $trans$ action has $E^l_{trans'} \doteq \{\langle [st_{trans'} + 5]\; sent_\sigma := \mathtt{T}\rangle\}$ and $E^u_{trans'} \doteq \{\langle [st_{trans'} + 8]\; sent := \mathtt{T}\rangle\}$.

Additional conditions $C^E_{a'}$. Let $t \doteq et_a - \delta$ be the time of an at-end effect that affects the value of f. In order to prevent other actions from changing the value of f during the interval $(\lambda(t), \nu(t)]$ where the value of f is uncertain, we add a condition in $C^E_{a'}$ to maintain the value of f_σ throughout the uncertain duration $(\lambda(t), \nu(t)]$.

$$C^E_{a'} \doteq \{\langle (\lambda(t), \nu(t)]\; f_\sigma = v\rangle, \mid \langle [t]\, f := v\rangle \in E_a\}$$

Since the effect on f_σ (belonging to $E^l_{a'}$) is applied at time $\lambda(t)$, the condition is satisfied immediately after the effect and we want to avoid concurrent modifications of either f or f_σ until the uncertainty interval ends at $\nu(t)$.[16]

For example, the $trans$ action gets the added condition $C^E_{trans'} \doteq \{\langle (st_{trans} + 5, st_{trans} + 8]\; sent_\sigma = \mathtt{T}\rangle\}$. The compilation of the $trans$ action is depicted in Figure 6.

6.1.5. Controllable Actions

Controllable actions are much simpler. For each $a \doteq \langle [l,u], C_a, E_a\rangle \in A_c$ we introduce a replacement action $a' \doteq \langle [l,u], C_{a'}, E_{a'}\rangle \in A'_c$, in which: (1) each condition in C containing variables in L is rewritten to check the values of both those variables and their shadows, and (2) each effect on a variable in L is applied to the variable and its shadow.

$$C_{a'} \doteq \{\langle [\![t_1, t_2]\!]\; \alpha\rangle \mid \alpha \in \theta(\psi), \langle [\![t_1, t_2]\!]\; \psi\rangle \in C_a\}$$

$$E_{a'} \doteq E_a \cup \{\langle [t]\, f_\sigma := v\rangle \mid f \in L, \langle [t]\, f := v\rangle \in E_a\}$$

[16]Note that it is sufficient to include a preservation condition for f_σ to ensure that neither f nor f_σ are modified during the uncertain interval. This is because every action that modifies f also modifies f_σ and has a similar preservation condition for f_σ if the action has uncertain duration. Any problematic overlap of another action that modifies f would therefore result in the assignment to the shadow variable for one of the actions overlapping with the preservation constraint of the shadow variable for the other action, which violates the semantics.

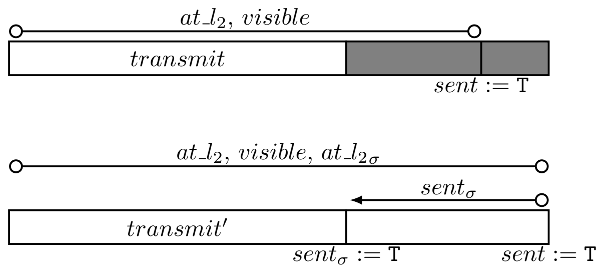


Figure 6: Graphical view of the original *transmit* action (top) and its compilation (bottom). Durative conditions are represented over the actions, while effects are reported under each action. We indicate closed interval extremes with circles and open ends with arrows. For example, the condition $sent_\sigma$ is open to the left and closed to the right.

6.1.6. Initial State I

The initial state is handled by initializing variables and their corresponding shadow variables in the same way as in the original problem.

$$I' \doteq I \cup \{f_\sigma := v \mid f \in L, I(f) = v\}$$

Intuitively, each initial condition on original variables is kept, and all shadow variables are initialized exactly like the corresponding original variable.

For example, the initial state of our running problem is the original initial state plus $\{sent_\sigma = \mathtt{F}, pos_\sigma = l_1\}$.

6.1.7. Timed Initial Literals

Timed Initial Literals T' are set similarly to controllable effects.

$$T' \doteq T \cup \{\langle [t]\, f_\sigma := v\rangle \mid f \in L, \langle [t]\, f := v\rangle \in T\}$$

In our example, we do not have timed initial literals operating on uncertain variables, thus $T' \doteq T$.

6.1.8. Goal Conditions

The goal conditions G are augmented to consider both the original and shadow variables.

$$G' \doteq \bigcup_{g \in G} \theta(g)$$

In our example, the set G' becomes $\{(sent = \mathtt{T}), (sent_\sigma = \mathtt{T})\}$.

6.2. Example

The compiled STPUD obtained by applying the compilation approach to the rover running example is as follows.

$$\begin{aligned}
V' &\doteq V \cup \{pos_\sigma : \{l_1, l_2\}, sent_\sigma : \{\mathtt{T}, \mathtt{F}\}\} \\
I' &\doteq I \cup \{pos_\sigma = l_1, sent_\sigma = \mathtt{F}\} \\
T' &\doteq \{\langle [14]\, visible := \mathtt{T}\rangle, \langle [30]\, visible := \mathtt{F}\rangle, \langle [15]\, hot := \mathtt{F}\rangle\} \\
G' &\doteq \{(sent = \mathtt{T}), (sent_\sigma = \mathtt{T})\} \\
A'_c &\doteq \{\langle [15], C_{move'}, E_{move'}\rangle, \langle [8], C_{trans'}, E_{trans'}\rangle\} \\
C_{move'} &\doteq \{\langle [st_{move}]\, pos = l_1\rangle, \langle [st_{move}]\, pos_\sigma = l_1\rangle, \\
&\qquad \langle [et_{move}]\, hot = \mathtt{F}\rangle, \langle (st_{move} + 10, st_{move} + 15]\, pos_\sigma = l_2\rangle, \\
C_{trans'} &\doteq \{\langle [st_{trans}, et_{trans}]\, pos = l_2\rangle, \langle [st_{trans}, et_{trans}]\, pos_\sigma = l_2\rangle, \\
&\qquad \langle [st_{trans}, et_{trans}]\, visible = \mathtt{T}\rangle, \langle (st_{trans} + 5, st_{trans} + 8]\, sent_\sigma = \mathtt{T}\rangle, \\
E_{move'} &\doteq \{\langle [st_{move} + 10]\, pos_\sigma := l_2\rangle, \langle [st_{move} + 15]\, pos := l_2\rangle\} \\
E_{trans'} &\doteq \{\langle [st_{trans} + 5]\, sent_\sigma := \mathtt{T}\rangle, \langle [st_{trans} + 8]\, sent := \mathtt{T}\rangle\}
\end{aligned}$$

6.3. Discussion

This compilation is sound and complete. Theorem 4 states that the original problem is solvable if and only if the resulting problem is solvable.

Theorem 4 (Soundness and Completeness)**.** *Let $P \doteq \langle V, I, T, G, A\rangle$ be a planning instance and $R \doteq \langle V', I', T', G', A'\rangle$ be its translation. P has a strong plan π if and only if R has a temporal plan σ.*

The proof we give in Appendix B is constructive and shows that any plan for the rewritten temporal planning problem is automatically a strong plan for the original problem (with the obvious mapping from the rewritten to the original actions).

The compilation produces a problem that has: (i) at most twice the number of variables of the original problem, (ii) at most twice the initial and timed assignments and (iii) exactly the same number of actions. The only point in which the compilation might produce exponentially large formulae is in the application of the θ function, which is exponential in the number of disjuncts constraining variables appearing in L. Since this only happens for disjunctive conditions, and the number of disjuncts is typically small, this is normally not a serious issue in practice.

Finally, we remark that any technique can be used to solve the compiled temporal planning problem, and any valid plan corresponds to a strong plan. Therefore, this technique allows for the mix of controllability and flexibility: if we employ a planner that is able to produce flexible solutions (i.e. an STN of possible solutions), all those solutions will be valid strong plans. This is an example of flexible-strong planning.

7. Worst-case Simplification

As we discussed in Section 3.4, it is in general impossible to solve the STPUD problem by considering uncontrollable actions as taking either the maximal or minimal duration of their duration interval: this motivated the development of S-FSSTP and the compilation approaches we presented in Section 6. However, under some specialized conditions, it is possible to soundly remove uncertainty simply by fixing the duration of an action to its maximal or minimal duration. In this section, we report a set of sufficient conditions that can be statically checked on a planning instance to simplify the uncertainty of an uncontrollable action.

7.1. Maximal-Duration Simplification

In order to soundly lengthen an uncontrollable action $a \doteq \langle [l, u], C, E\rangle$ to its maximum duration u without changing its conditions or its effects, we need to make sure that there are no other action conditions or effects (or TILs) that could interfere with conditions or effects of the action a that are specified relative to the end time of a. This means that any such threatening action conditions or effects (or TILs) must be impossible during the time window when the end-relative condition or effect might occur. For example, if a has duration in $[l, u]$ and has an end-relative effect $\langle [et_a - \delta]\ f := v\rangle$, no other action could have a condition or effect on f that might occur within $[l - \delta, u - \delta]$. Likewise, no TIL on f could occur within $[l - \delta, u - \delta]$.

To make this more precise, let a' be the extension of action a to its maximal duration u. For this extension to be sound, we require:

1. For each end-relative effect $\langle [et_{a'} - \delta]\ f := v\rangle$ of a':
   - no other action can have an effect on f or a condition containing f that could occur within $[l - \delta, u - \delta]$ of a';
   - no TIL on f can occur within $[l - \delta, u - \delta]$ of a'.
2. For each end-relative condition $\langle [et_{a'} - \delta_1, et_{a'} - \delta_2 \rrbracket\ \phi\rangle$:
   - no other action can have an effect on any variable appearing in ϕ or a condition inconsistent with ϕ that could occur within $[l - \delta_1, u - \delta_1)$ of a';
   - no TIL on any variable in ϕ can occur within $[l - \delta_1, u - \delta_1)$ of a'.
3. For each end-relative condition $\langle (et_{a'} - \delta_1, et_{a'} - \delta_2 \rrbracket\ \phi\rangle$:
   - no other action can have an effect on any variable appearing in ϕ or a condition inconsistent with ϕ that could occur within $(l - \delta_1, u - \delta_1]$ of a';
   - no TIL on any variable in ϕ can occur within $(l - \delta_1, u - \delta_1]$ of a',

Some notes are in order. First, the difference between cases (2) and (3) above is whether the condition interval is closed or open on the left, which determines whether the restriction needs to be open or closed on the right.

Whether the condition interval is closed or open on the right does not affect the requirements. Second, it might seem like the interval in condition (2) above should be $[l-\delta_1, u-\delta_2\rrbracket$ rather than the sub-interval $[l-\delta_1, u-\delta_1)$. However the extended action a' still has the condition ϕ over the interval $[et_{a'}-\delta_1, et_{a'}-\delta_2\rrbracket$ and since a's duration is u, the interval from $[u-\delta_1, u-\delta_2\rrbracket$ is covered by the new action. As a result, we only need to guard against a threat that might occur within $[l-\delta_1, u-\delta_2)$. Similarly for case (3). Finally, it might seem that we also need to consider conditions of the form $\langle\llbracket st_a+\delta_1, et_a-\delta_2\rrbracket\ \phi\rangle$. However, this is not needed because the condition for a' is still over the interval $\llbracket st_{a'}+\delta_1, et_{a'}-\delta_2\rrbracket$ which covers the required interval. This also applies to overall conditions since the extended action a' still has an overall condition that covers the maximum possible duration of the original action a.

While the above criteria are fairly general, they are difficult to apply. In particular, showing that some condition or effect of another action cannot occur within some specific subinterval of action a' may require computationally complex reasoning about the ways in which the actions can overlap. A more restrictive, but more practical set of conditions is:

1. For each end-relative effect $\langle[et_{a'}-\delta]\ f := v\rangle$ of a':
   - no other action b with an effect on f or a condition containing f can overlap with a';
   - there are no TILs on f.
2. For each end-relative condition $\langle\llbracket et_{a'}-\delta_1, et_a-\delta_2\rrbracket\ \phi\rangle$ of a':
   - no other action with an effect on any variable appearing in ϕ or a condition inconsistent with ϕ can overlap with a';
   - there are no TILs on any variable in ϕ.

An action b cannot overlap with action a' if any of the following hold:

- b has an overall condition inconsistent with an overall condition of a';
- b cannot start during a' and a' cannot start during b. This holds if a start condition or effect of b is *incompatible* with an overall condition of a' and a start condition or effect of a' is *incompatible* with an overall condition of b;
- b cannot end during a' and a' cannot end during b. This holds if an end condition or effect of b is *incompatible* with an overall condition of a' and an end condition or effect of a' is *incompatible* with an overall condition of b.

Two conditions are *incompatible* if they are logically inconsistent. An effect $\langle f := v\rangle$ is *incompatible* with a condition ϕ if ϕ contains f. Two effects $\langle f_1 := v_1\rangle$ and $\langle f_2 := v_2\rangle$ are *incompatible* if $f_1 = f_2$.

These conditions can be checked syntactically for the actions in a domain, making this easy and efficient to implement.

7.1.1. Example

To clarify the rules for this simplification, let us consider the running example of Section 3.2. We show how the *transmit* action in the example can be simplified by considering its maximal duration, while the *move* action cannot be simplified.

The *transmit* action has only one effect: $\langle [et_{trans}]\ sent := \mathtt{T}\rangle$ and no other action nor TIL can change the *sent* variable. Moreover, no action has a condition that depends on *sent*, hence general condition (1) is satisfied. Finally, general conditions (2) and (3) are trivially satisfied because the *transmit* action has no ending conditions. Therefore, all the conditions are satisfied and the example problem can be simplified as follows.

$$\begin{aligned}
V &\doteq \{pos : \{l_1, l_2\}, visible : \{\mathtt{T}, \mathtt{F}\}, hot : \{\mathtt{T}, \mathtt{F}\}, sent : \{\mathtt{T}, \mathtt{F}\}\}\\
I &\doteq \{pos = l_1, visible = \mathtt{F}, sent = \mathtt{F}, hot = \mathtt{T}\}\\
T &\doteq \{\langle [14]\ visible := \mathtt{T}\rangle, \langle [30]\ visible := \mathtt{F}\rangle, \langle [15]\ hot := \mathtt{F}\rangle\}\\
G &\doteq \{(sent = \mathtt{T})\}\\
A_c &\doteq \{\langle [8], C_{trans}, E_{trans}\rangle\}\\
A_u &\doteq \{\langle [10, 15], C_{move}, E_{move}\rangle\}\\
C_{move} &\doteq \{\langle [st_{move}]\ pos = l_1\rangle, \langle [et_{move}]\ hot = \mathtt{F}\rangle\}\\
C_{trans} &\doteq \{\langle [st_{trans}, et_{trans}]\ pos = l_2\rangle, \langle [st_{trans}, et_{trans}]\ visible = \mathtt{T}\rangle\}\\
E_{move} &\doteq \{\langle [et_{move}]\ pos := l_2\rangle\}\\
E_{trans} &\doteq \{\langle [et_{trans}]\ sent := \mathtt{T}\rangle\}
\end{aligned}$$

The resulting simplified problem now has one uncontrollable action and one controllable action, without any additional variables or conditions. Moreover, each plan for the resulting problem corresponds to a strong plan for the original problem (with the obvious removal of durations for the simplified actions in the plan).

Let us now consider the *move* action. In this case, condition (2) is violated, because the TIL $\langle [15]\ hot := \mathtt{F}\rangle$ affects the variable *hot* that appears in the condition $\langle [et_{move}]\ hot = \mathtt{F}\rangle$. In fact, a plan starting the move action at time 1 would be valid in the simplified problem, but it does not yield a strong plan because the uncontrollable action *move* could end at time 11 and the ending condition $\langle [et_{move}]\ hot = \mathtt{F}\rangle$ would be unsatisfied. For this reason, the *move* action cannot be simplified to its maximal duration.

7.2. Minimal-Duration Simplification

In order to soundly shorten an uncontrollable action $a \doteq \langle [l, u], C, E\rangle$ to its minimum duration l, we need to make sure that there are no other action conditions or effects (or TILs) that could interfere with conditions or effects of the action a that are specified relative to the end time of a. This means that any such threatening action conditions or effects (or TILs) cannot occur during the time window when the end-relative condition or effect might actually occur if a takes more time than the minimum. For example, if a has duration in $[l, u]$

and has an end-relative effect $\langle [et_a - \delta]\ f := v\rangle$, no other action could have a condition or effect on f that might occur within $[l - \delta, u - \delta]$. Likewise, no TIL on f could occur within $[l - \delta, u - \delta]$.

To make this more precise, let a' be the shortening of action a to its minimum duration l. For this simplification to be sound, we require:

1. For each end-relative effect $\langle [et_{a'} - \delta]\ f := v\rangle$ of a':
   - no other action can have an effect on f or a condition containing f that could occur within $[l - \delta, u - \delta]$ of a';
   - no TIL on f can occur within $[l - \delta, u - \delta]$ of a'.
2. For each end-relative condition $\langle [\![t, et_{a'} - \delta_2]\ \phi\rangle$:
   - no other action can have an effect on any variable appearing in ϕ or a condition inconsistent with ϕ that could occur within $(l - \delta_2, u - \delta_2]$ of a';
   - no TIL on any variable in ϕ can occur within $(l - \delta_2, u - \delta_2]$ of a'.
3. For each end-relative condition $\langle [\![t, et_{a'} - \delta_2)\ \phi\rangle$:
   - no other action can have an effect on any variable appearing in ϕ or a condition inconsistent with ϕ that could occur within $[l - \delta_2, u - \delta_2)$ of a';
   - no TIL on any variable in ϕ can occur within $[l - \delta_2, u - \delta_2)$ of a'.

Again, some notes are in order, similar to those for the maximal duration simplification case. First, the difference between cases (2) and (3) above is whether the condition interval is closed or open on the right, which determines whether the restriction needs to be open or closed on the left. Whether the condition interval is closed or open on the left does not affect the requirements. Likewise, whether the start of the condition interval is relative to the start or end of the action does not matter. Second, it might seem like the interval in condition (2) above should be $[\![t, u-\delta_2]$ rather than the sub-interval $(l-\delta_2, u-\delta_2]$. However the shortened action a' still has the condition ϕ over the interval $[\![t, et_{a'} - \delta_2]$ and since a's duration is l, the interval from $[\![t, l - \delta_2]$ is covered by the new action. As a result, we only need to guard against a threat that might occur within $(l - \delta_2, u - \delta_2]$. Similarly for case (3). Finally, the intervals mentioned in the above requirements often extend beyond the duration of the action a'. For example, if action a has an end effect $\langle [et_a]\ f := v\rangle$ and is shortened to its minimum duration, the safety interval $[l, u]$ starts at the end of the shortened action a'.

7.2.1. Example

As an example, consider the classical match-fusebox example. An action a lights a match producing the light needed to change a fuse by an action b as illustrated in Figure 7. Suppose that the duration of a is uncertain in $[10, 15]$ (because different matches burn differently), and suppose that b has an overall condition of having light (hence b must be contained in a). In this simple

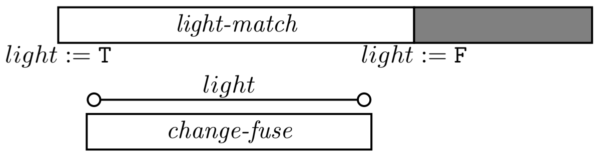


Figure 7: A simple fuse-changing example. The change-fuse action must fit inside the light-match action to ensure success. Shortening *light-match* to its minimum duration is fine in this case.

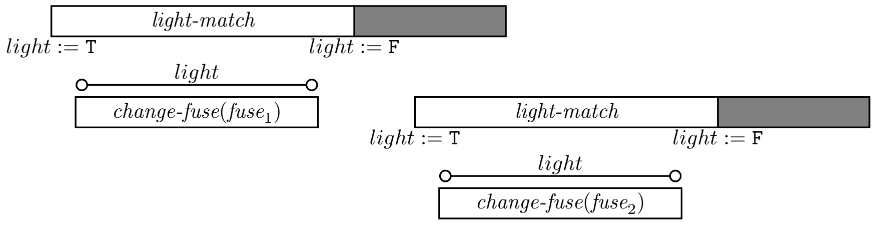


Figure 8: A double fuse-changing example. This plan is not sound because the second *light-match* action should not take place before the first is guaranteed to finish.

scenario, a can be simplified to its minimal duration because there are no other actions with threatening effects, and the only threatening condition is the overall condition of b, but it cannot happen during $[10, 15]$ because b is now forced to occur within the shortened action a', hence b must start after the start of a' and must end before the end of a' at time $st_{a'} + 10$.

The catch here is that we need to ensure that the *light-match* action a' can only happen once. If a' ends at time $t + 10$ and we were to execute a second light-match a'_2 at time $t + 12$, it's possible that the end-effect $light := false$ of a', could actually happen at the same time as the start-effect $light := true$ of a'_2, violating the semantics (which prohibits simultaneous effects on a variable). In this case, the second *light-match* action a'_2 would be violating condition (1) above. If there were two fuses to change, and a single match is only sufficient for changing one fuse, the minimum duration simplification is problematic. If we used the shortened *light-match* action, we might think that the plan shown in Figure 8 is valid. However, the first *light-match* could end after the second *light-match* starts, resulting in $light := false$ during the second *change-fuse* action. It might seem that this could be fixed by a more sophisticated modeling of $light$ as a metric quantity that is increased at the beginning of *light-match*, and decreased at the end of *light-match*. However, even with this modeling, we would have to show that there is no upper bound on the amount of light that can occur, or that the maximum number of possible overlapping *light-match* actions cannot possibly exceed this bound. We would also need to show that no other possible action (like developing film) depends on the light being below a certain level. All of this illustrates the sophistication of the reasoning that may be required to shorten an action to its minimum duration.

7.3. Discussion

The conditions we listed above are sufficient to soundly simplify away uncontrollable durations to either the maximal or minimal duration.

The lengthening conditions are much easier to satisfy than the conditions for shortening an action to its minimum duration. The reason for this is that the lengthening process extends overall conditions of an action a, which tends to prevent bad things from happening during $[l, u]$. In contrast, the conditions for shortening an action are much harder to satisfy because they impose some outside-of-action requirements that are rarely met in practice, or are much more difficult to prove. A common source of problems for the minimum duration simplification is when the action being shortened can appear more than once in a plan, because such actions often interfere with themselves.

8. Experimental evaluation

We now empirically evaluate the approaches and simplifications we presented. First we discuss the experimental set-up, then we explain how to use PDDL 2.1 planners for dealing with the compilation output, and finally we discuss the results.

8.1. Experimental Set-Up

In order to enable the specification of durative actions with uncontrollable durations, we extended the PDDL 2.1 planning language syntax with the addition of the keyword `uncontrollable-durative-action`: the construct is analogous to the usual `durative-action` construct, but marks the action as uncontrollable. We refer to this extension of the language as PDDL-U.

For testing, we adopted the temporal planning domains from the temporal track of the International Planning Competition in 2011 [39]: these domains are written in the PDDL 2.1 language. We modified them by making some actions uncontrollable, enlarging the duration intervals of actions, and created several versions of each domain. The resulting benchmark set is composed of a total of 901 planning problem instances. Since both the compilation and the simplification techniques manipulate the domain specifications and automated planners are quite sensitive to the input, we implemented a tool that randomly "scrambles" PDDL input problems. The tool always generates a problem instance that is logically equivalent to the original one, but it changes the order of actions, conditions and effects in the concrete specification. Using this tool, we performed each experiment 5 times (with different random seeds to obtain different "scramblings"), considering each run as a separate experiment. Thus, we are left with a virtual benchmark set of 4505 planning instances. We chose this approach because we realized that minimal variations in the ordering of actions or in the ordering of the conditions within an action produced very different performance. This sensitivity issue is highlighted in several papers in the literature [40]; we use this scrambling augmentation procedure to limit the bias due to sensitivity to the syntactic ordering in the problem.

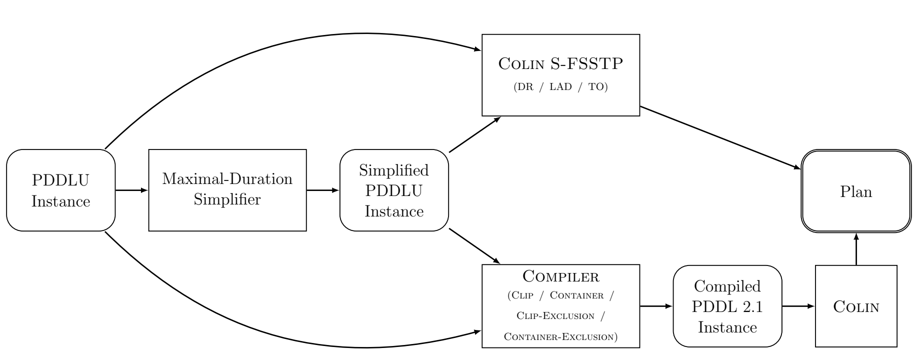

Colin S-FSSTP
(DR / LAD / TO)
PDDLU Instance
Maximal-Duration Simplifier
Simplified PDDLU Instance
Plan
Compiler
(Clip / Container / Clip-Exclusion / Container-Exclusion)
Compiled PDDL 2.1 Instance
Colin

Figure 9: Block diagram representation of the implemented tools for the experimental evaluation.

We implemented the direct approaches for the Unc-Extr problem class (described in Section 5) as an extension of the Colin planner [9]. The parser and the internal structures of Colin were modified to support the processing of this extension. We added three new schedulers to Colin, each implementing one of the defined encodings. The temporal solvers for strong controllability were implemented in C++, following the approach presented in [11], using the MathSAT [41] SMT solver as workhorse. Finally, the heuristic for the forward search planner was left unchanged.
In the following, we write TO to refer to the encoding based on Total Ordering, LAD for the Last Achiever Deordering and DR for the Disjunctive Reordering.

The compilation described in Section 6 was implemented as a Java translator that takes as input a PDDL-U specification and produces a plain PDDL 2.1 temporal planning problem. The formal translation is designed to generate a Ctrl-Arbit problem even when starting from an Unc-Extr instance. For this reason, the translator is equipped with a technique to remove intermediate effects and conditions resulting in a PDDL 2.1 description. We discuss this technique in detail in Section 8.2. Since the direct approaches have been implemented using a modification of the Colin planner, we also used Colin to solve the temporal planning instances produced by the compilation.

Finally, the maximal-duration simplification has been implemented as a Java re-writer that takes as input a PDDL-U instance and produces a simplified PDDL-U specification. We note that in some cases the simplifier is able to remove all the temporal uncertainty from the problem and, thanks to the way PDDL-U is defined, a plain PDDL 2.1 planning problem is produced by the simplifier. In these experiments we did not attempt to use the minimal-duration simplification.

The implemented tools and the possible data-flows are depicted in Figure 9.

All the experiments were executed on a Scientific Linux 64 bit, 12 core Intel Xeon at 2.67GHz, with 96GB RAM. We used a timeout of 10 minutes, and a memory limit of 8GB.

All the tools and the benchmark set can be downloaded from `http://es.fbk.eu/people/amicheli/resources/aij-stpud`.

8.2. Dealing with Intermediate Effects and Conditions

Given a PDDL-U planning instance, the compilation described in Section 6 produces an instance having no uncontrollable durations, but with intermediate effects and conditions. In particular, for the UNC-EXTR case that is expressible by PDDL-U, the algorithm introduces an intermediate effect and a durative condition for each uncontrollable action. Unfortunately, the PDDL 2.1 language does not support these features natively. Therefore, we need to get rid of these characteristics before being able to use PDDL 2.1 planners on the result of our compilation.

In the following we assume a PDDL semantics: we require a minimum time quantum called ϵ forcing each pair of time points in the plan to be separated by at least ϵ time.

The papers [42, 34, 31] present some ideas on how to encode intermediate events in PDDL 2.1. In particular, [34] assumes actions have a fixed duration and proposes some "standard" constructs to encode several features as polynomial transformations of the PDDL domain. The work in [31] uses the construction in [42] to develop a more general translation from ANML to PDDL.

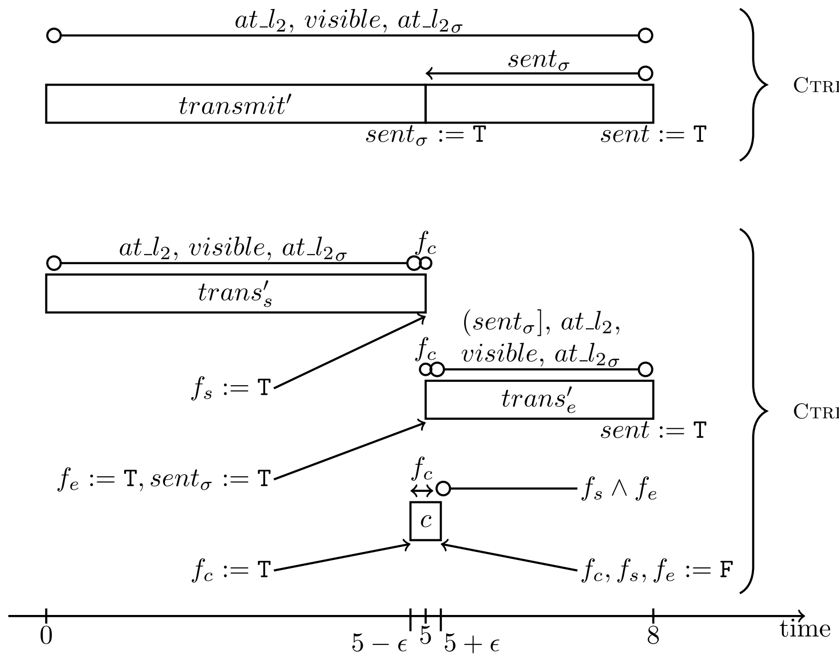
The clip-action construction described in [34] allows one to force two or more time points (either the start or the end of actions) to happen simultaneously or to be separated by exactly ϵ. The construction uses one additional action with duration 2ϵ (or 3ϵ in case an ϵ separation is required). Each time point p_i being clipped requires a special effect $\langle f_{p_i} := \mathtt{T}\rangle$ where f_{p_i} is a fresh Boolean variable. The clip action for clipping the points $\{p_1, \cdots, p_n\}$ is defined as follows.

$$\begin{aligned}
&clip \doteq \langle [2\epsilon], C_{clip}, E_{clip}\rangle\\
&C_{clip} \doteq \{\langle (st_{clip}, et_{clip})\ f_s = \mathtt{T}\rangle\} \cup \{\langle [et_{clip}]\ f_{p_i} = \mathtt{T}\rangle \mid i \in [1, n]\}\\
&E_{clip} \doteq \{\langle [st_{clip}]\ f_s := \mathtt{T}\rangle, \langle [et_{clip}]\ f_s := \mathtt{F}\rangle\} \cup \{\langle [et_{clip}]\ f_{p_i} := \mathtt{F}\rangle \mid i \in [1, n]\}
\end{aligned}$$

Intuitively the clip is an action that sets a fresh, dummy variable f_s to true when it starts and resets it upon termination. We can now clip time points to impose the condition $f_s = \mathtt{T}$ exactly at those time points. Since no two events can happen with distance lower than ϵ, the clip guarantees simultaneous execution. If the time points being clipped have mutually-exclusive effects, we need a clip of duration 3ϵ to achieve an ϵ separation of the two time points. The special effects on the f_{p_i} variables are needed to prevent a clip from being instantiated without clipping all the needed time points.

Figure 10 shows an example of the construction for the action τ' derived from the running example action τ by means of the uncertainty compilation.

This construction can be used to encode intermediate effects and conditions by splitting an action duration in pieces, one for each sub-interval of the action duration delimited by an intermediate effect or condition bound. This technique works perfectly for fixed duration actions (note that this is enough for the



at_l_2, $visible$, $at\_l_{2\sigma}$
$sent_\sigma$
$transmit'$
$sent_\sigma := \mathtt{T}$
$sent := \mathtt{T}$
Ctrl-Arbit
at_l_2, $visible$, $at\_l_{2\sigma}$
f_c
$trans'_s$
$(sent_\sigma]$, at_l_2,
f_c
$visible$, $at\_l_{2\sigma}$
$f_s := \mathtt{T}$
$trans'_e$
$sent := \mathtt{T}$
Ctrl-Extr
$f_e := \mathtt{T}, sent_\sigma := \mathtt{T}$
f_c
$f_s \wedge f_e$
c
$f_c := \mathtt{T}$
$f_c, f_s, f_e := \mathtt{F}$
0
$5-\epsilon$
5
$5+\epsilon$
8
time

Figure 10: Clip action construction example: the action $trans'$ derived from the running example action $trans$ by means of the compilation is rewritten as an equivalent triplet of actions. The condition f_c that is the basis of the construction is imposed at time 5 at the end of the action $trans'_s$ and at the begin of the action $trans'_e$; moreover it is an overall condition of the clip action c. The literals f_s and f_e are required to ensure that the clip cannot be instantiated without clipping the two actions. For space reasons, we indicated the condition $\langle (5, 8]\ sent_\sigma = \mathtt{T} \rangle$ together with the other conditions in the interval $[5, 8]$.

outcome of the translation as each uncontrollable action is compiled as a fixed-duration action and no other intermediate effects or conditions are artificially added).

The clip action is not the only construction that can be used to encode intermediate effects and conditions. A second relevant technique [42, 31] uses a container action that spans the whole duration of the original action and exactly contains a number of sub-actions, one for each sub-interval. This paradigm is depicted in Figure 11.

Both these techniques have significant overhead: to remove an intermediate effect or condition, a single action is replaced with three actions, increasing the plan length; moreover, additional variables are added to the problem description to support the construction, widening the search space.

We improved these two classical constructions by developing a modification to guide the planners and limit this overhead. The fundamental observation is that both these encodings are designed in such a way that when the first action

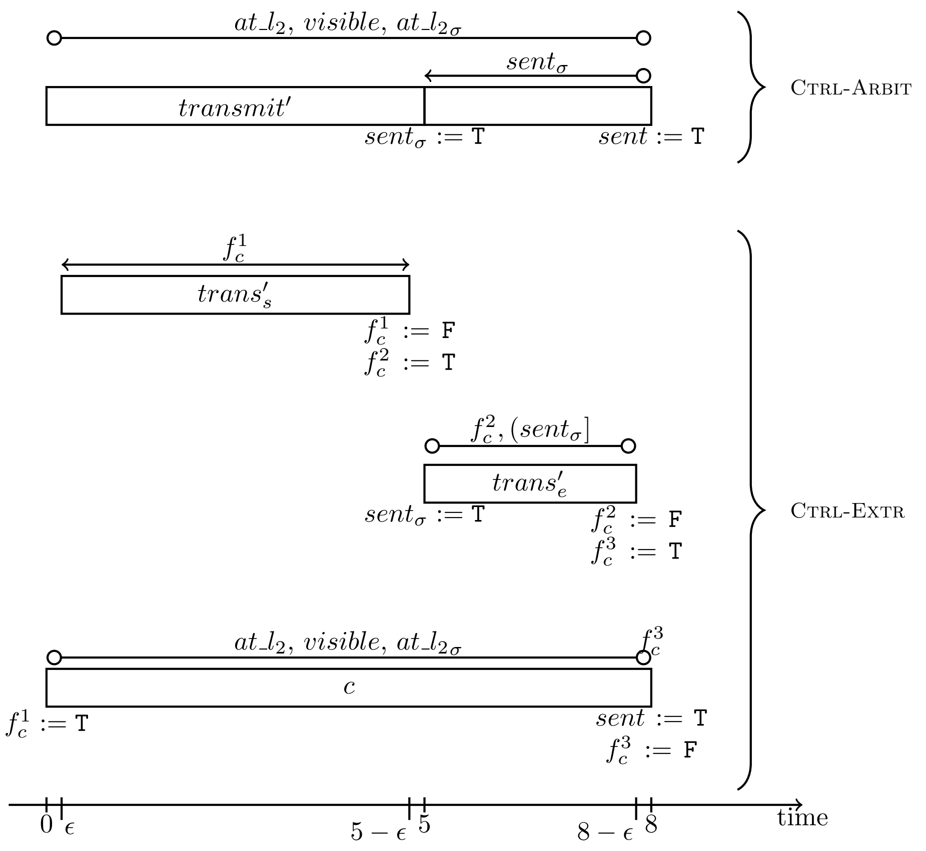


Figure 11: Container action construction example: the action *trans′* derived from the running example action *trans* by means of the compilation is rewritten as an equivalent triplet of actions.

of the construction is started (the first piece in the clip action construction and the container in the other), the planner has no choice but to instantiate the rest of the construction. However, nothing is telling the planner to avoid useless search on branches where the structure has not been correctly instantiated. We can exploit the PDDL 2.1 features to force the planner into the correct choice: we call the resulting technique “exclusion literal simplification”.

We start by describing the idea on the clip-action construction. Suppose we are clipping two actions a and b with a clip c. We introduce a fresh Boolean variable f_c^e that is initially true and is falsified during the execution of the clip c with appropriate start and end effects. Then, f_c^e becomes a condition for starting and terminating each action in the problem except for the clip c and the two clipped actions a and b. Moreover f_c^e is a condition for starting a and for terminating b. In this way, once c is inserted in the plan, the search has no choice but to immediately start b, because no other successor is possible given that b is the only action that can be started during c. This additional construction is applied for every clip-action in the instance to prune the search in all the cases.

The same idea can also be applied to the container construction by imposing mutual exclusion on the "holes" within the container action to force the planner to immediately expand the next action in the sequence. All these techniques have been implemented as a post-processing of the compilation result and are evaluated in the following sections.

8.3. Overall Results

In the following, Colin-Clip refers to the compilation approach solved with the Colin planner using the clip technique to remove intermediate effects. Colin-Clip-MuxLiterals indicates the same but with the addition of mutual exclusion literals to guide the solver. Similarly, Colin-Container and Colin-Container-MuxLiterals refer to the compilation techniques using the container construction to remove intermediate effects. The S-FSSTP techniques are referred to as S-FSSTP-TO, S-FSSTP-LAD and S-FSSTP-DR, for the TO, LAD and DR techniques, respectively.

Figure 12 gives an overview of the performance of the presented techniques in our experiments. First, we consider the direct approach obtained extending the FSSTP planning framework. We note how the LAD approach performs much better than the TO approach, which in turn performs better than the DR approach. In fact, LAD is able to solve almost twice as many instances as DR. However, both the LAD and TO techniques are not complete in general: it might be the case that a problem is reported as unsolvable when a strong plan exists. This situation never occurred in our experiments: in each instance the planner either returned a valid strong plan or did not terminate within the allowed time limit.

For the compilation technique, we note how the performance dramatically depends on the technique used to compile away intermediate effects and conditions. The clip construction proved to be more effective than the container construction. Moreover, both the techniques benefit from the mutual exclusion improvement.

In order to compare the compilation technique with the direct approach, we report in Figure 13 the result of the best-performing compilation technique with the two native approaches. The results are mixed with no clear winner. This means that the two techniques exhibit a complementary behavior: on some instances the direct approach is vastly superior, on others it is beaten by the compilation. This is confirmed by the performance of the Virtual Best Solver (VBS) in Figure 12: the performance of the VBS is obtained by taking the results of the fastest solver for each instance. The VBS line in the plot solves far more instances than any actual solver; hence, the various solvers that contribute to the VBS are able to solve different sets of instances.

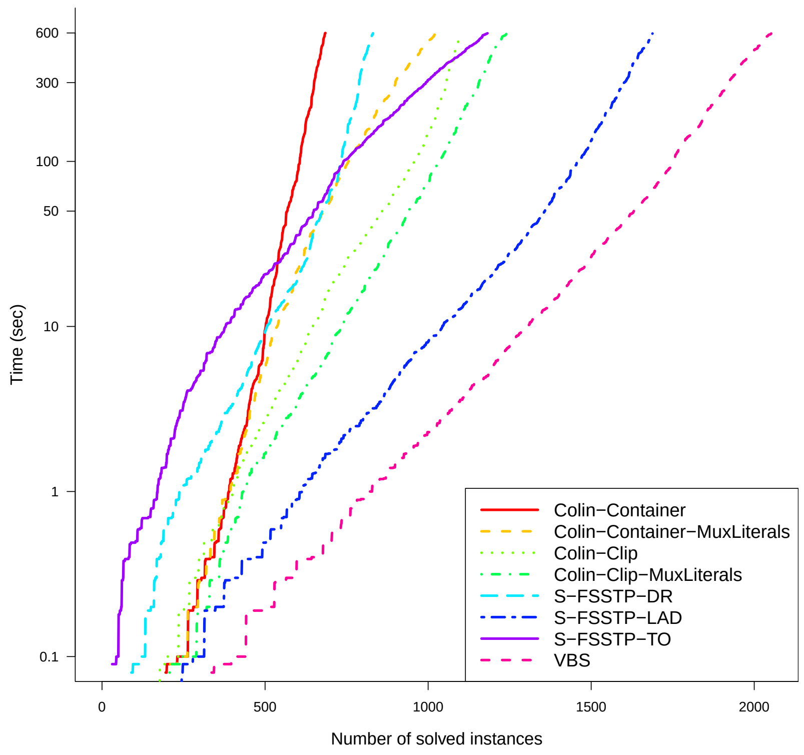


Figure 12: Log-scale cactus plot showing the performance of the solvers. The "VBS" line is the Virtual Best Solver that is computed taking for each instance the performance of the fastest solver among all the others.

We analyzed several features of the instances without finding a clear correlation with the better solver. We leave the use of automated data mining techniques for the best solver prediction as future work.

8.3.1. Impact of Worst-Case Simplification

We implemented the simplifier following the maximal-duration simplification described in Section 7. Since the simplification requires a mutual exclusion generator, we provided a simple generator based on syntactic rules. Moreover, to strengthen the mutual exclusion predicate we manually checked for all the domains that all the uncontrollable actions were mutually exclusive with themselves, and we forced this information in the mutual-exclusion generator.

The pseudo-code of the generator is reported in Algorithm 3. The mutual exclusion rules are purely syntactic and only exploit the overall condition of the actions. The entry function MaxDurationSimplifiable returns $\top$ if the

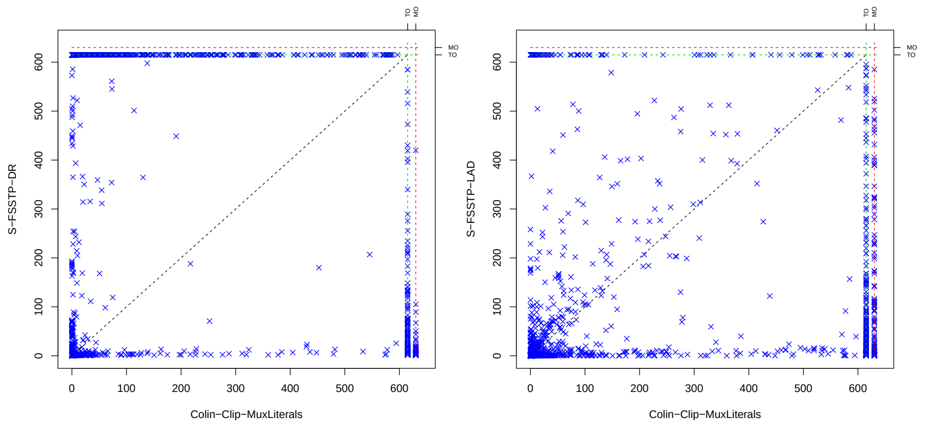


Figure 13: Scatter plot comparing the best technique for solving the compilation approach with the complete DR method (a) and the incomplete LAD method (b). Axes indicate run-time in seconds, each point represents the comparison of time required by the two solvers being compared. The points at the top and right edges are problems where one technique could solve the problem in the allotted time, but the other could not.

given action a can be simplified to its maximal duration. The function ConflictingActions returns the set of actions that have a condition or an effect conflicting with an ending effect of the given uncontrollable action. CheckMutexAssume implements the policy we described: all action are mutually exclusive with themselves. Finally, CheckMutex performs the syntactic check: in the code StartConditions, OverallConditions, and EndConditions return the set of starting, overall and ending conditions, respectively. Moreover, StartEffects and EndEffects return the set of starting and ending effects for the given action.

We note that the generator is sound but incomplete: if two actions are marked as mutually exclusive, they are, but it might be the case that two actions are mutually exclusive but the generator fails to recognize this. This is acceptable in our framework, because if an uncontrollable action is not mutually exclusive with all the actions threatening its ending effects or conditions, the simplification is not applied and a sound and complete technique for dealing with uncertainty is used.

We ran the benchmark tests with the simplification used as a pre-processor for all the techniques: the results are reported in Figures 14 and 15. We differentiate a technique from its version in which simplification is applied as preprocessing by adding a "+ Simp" to the technique name.

The simplification turns out to be useful in all the cases but the LAD and TO approaches where it has virtually no effect. This is because there is little

Algorithm 3 Syntactic simplification procedure: an action $a \in A_u$ can be simplified by fixing its maximal duration if MaxDurationSimplifiable(a) returns $\top$. In the pseudocode, we distinguished the syntactic mutual exclusion checking (CheckMutex) from the chek that ssumes that an action is mutually exclusive with itself (CheckMutexAssume) that we used in the experiments.

```
procedure CheckMutex($a$, $b$)
    if $\exists c_1 \in$ OverallConditions($a$).$\exists c_2 \in$ OverallConditions($b$).$c_1 \wedge c_2 \models \bot$ then
        return $\top$
    if ($\exists c_1 \in$ OverallConditions($a$).$\exists c_2 \in$ StartConditions($b$).$c_1 \wedge c_2 \models \bot$) $\wedge$
        ($\exists c_1 \in$ OverallConditions($b$).$\exists c_2 \in$ StartConditions($a$).$c_1 \wedge c_2 \models \bot$) then
        return $\top$
    if ($\exists c_1 \in$ OverallConditions($a$).$\exists c_2 \in$ EndConditions($b$).$c_1 \wedge c_2 \models \bot$) $\wedge$
        ($\exists c_1 \in$ OverallConditions($b$).$\exists c_2 \in$ EndConditions($a$).$c_1 \wedge c_2 \models \bot$) then
        return $\top$
    if ($\exists c_1 \in$ OverallConditions($a$).$\exists c_2 \in$ StartEffects($b$).$c_1 \wedge c_2 \models \bot$) $\wedge$
        ($\exists c_1 \in$ OverallConditions($b$).$\exists c_2 \in$ StartEffects($a$).$c_1 \wedge c_2 \models \bot$) then
        return $\top$
    if ($\exists c_1 \in$ OverallConditions($a$).$\exists c_2 \in$ EndEffects($b$).$c_1 \wedge c_2 \models \bot$) $\wedge$
        ($\exists c_1 \in$ OverallConditions($b$).$\exists c_2 \in$ EndEffects($a$).$c_1 \wedge c_2 \models \bot$) then
        return $\top$
    return $\bot$

procedure CheckMutexAssume($a$, $b$)
    if $a = b$ then
        return $\top$
    else
        return CheckMutex($a$, $b$)

procedure MaxDurationSimplifiable($a$)
    for all $b \in$ ConflictingActions($a$) do
        if $\neg$ CheckMutexAssume($a$, $b$) then
            return $\bot$
    return $\top$
```

overhead in solving an STNU with a DTNU solver. The simplification technique removes some of the uncertainty, but only changes some time points from uncontrollable to controllable while keeping the same constraint structure.

On the other hand, the simplification is very beneficial for the compilation approaches: this is because of two factors. First, converting a single uncontrollable action from an instance results in a reduction of the plan length in the compiled domain. In fact, each uncontrollable action gets translated into three controllable actions in the compilation. Second, all the additional variables needed for the intermediate effect construction of an uncontrollable action are removed if the action is simplified as controllable.

In order to show the impact of the simplification on the solvers, we show a scatter plot of the DR and Colin-Clip-MuxLiterals approaches with and without the simplification pre-processing step (Figure 16). The plots highlights that the simplification is either beneficial or neutral with very few detrimental cases.

We also analyzed in detail the performance of the best-performing solvers for the compilation and direct approaches, all with the addition of simplification. We show the scatter plots in Figure 17. As before, there is evident complementary behavior for the two approaches.

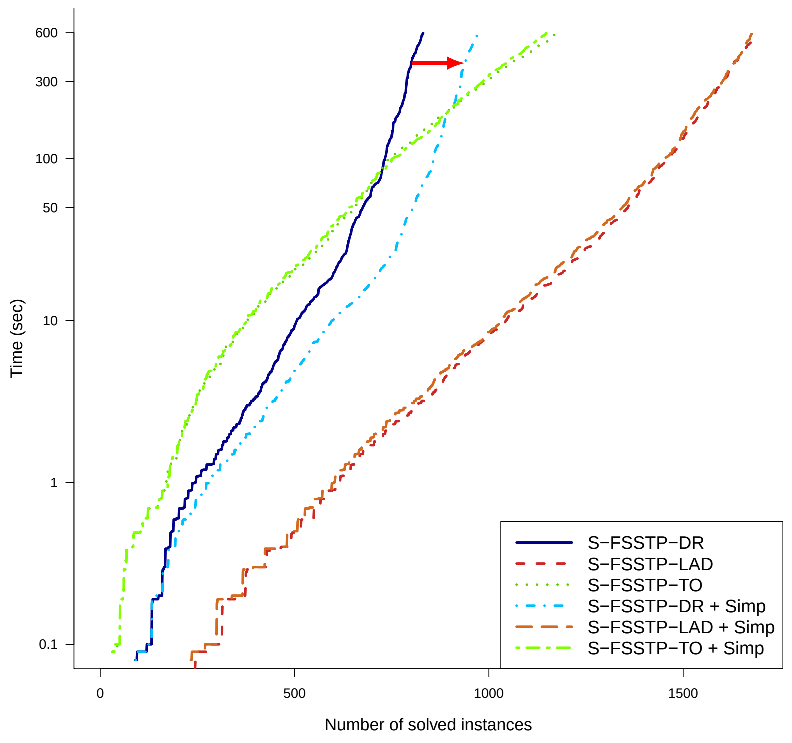


Figure 14: Log-scale cactus plot showing the impact of simplification in our experiments on the FSSTP techniques. The red arrows highlight the impact of simplification by linking a solver without simplification pre-processing with the corresponding pre-processed solver.

Finally, Figure 18 shows a comparison of the performance of all the solvers with the simplification enabled. The VBS + Simp line is calculated by taking, for each instance, the best performing solver among the others shown in the plot. Comparing this plot with Figure 12, it is evident that the simplification technique is very effective for the compilation approaches, which are now able to beat the native techniques. Moreover, we still see that the virtual best solver is far more effective than any other technique; hence, there is still a strong partitioning of the instances in terms of the best solving technique. This plot also shows that taking a portfolio approach with the simplification, we can solve more instances than without the simplification. In fact, while the VBS solver is able to solve a total of 2052 instances, the VBS + Simp solver can handle 2597 instances within the timeout.

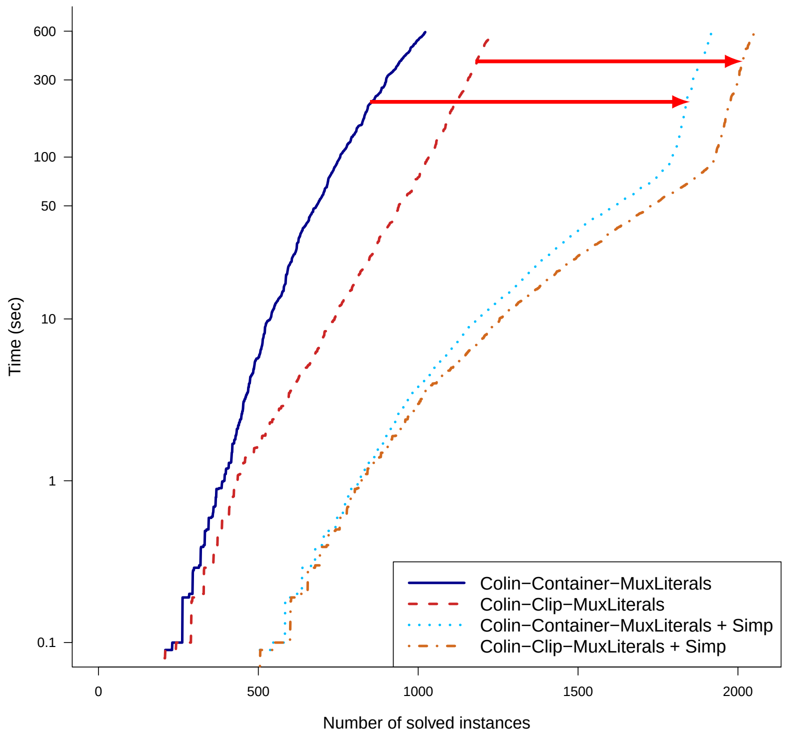


Figure 15: Log-scale cactus plot showing the impact of simplification in our experiments on the compilation approach. The red arrows highlight the impact of simplification by linking a solver without simplification pre-processing with the corresponding pre-processed solver.

9. Conclusions and Future Work

In this paper we introduced the Strong Temporal Planning with Uncontrollable Durations (STPUD) problem, which extends Temporal Planning to deal with actions having uncontrollable duration. We search for temporally strong solutions, i.e. plans that are guaranteed to achieve the goal regardless of the actual duration of the actions that are under the control of the execution environment.

We presented two complementary approaches. The first one is based on the integration of a "classical" temporal planner with a solver for temporal networks with uncertainty. The second approach reduces any STPUD problem to a "classical" temporal planning problem, where the actions have controllable durations. Finally, we proposed a technique that is able to eliminate some of the uncontrollable durations by reasoning in terms of worst case execution. We

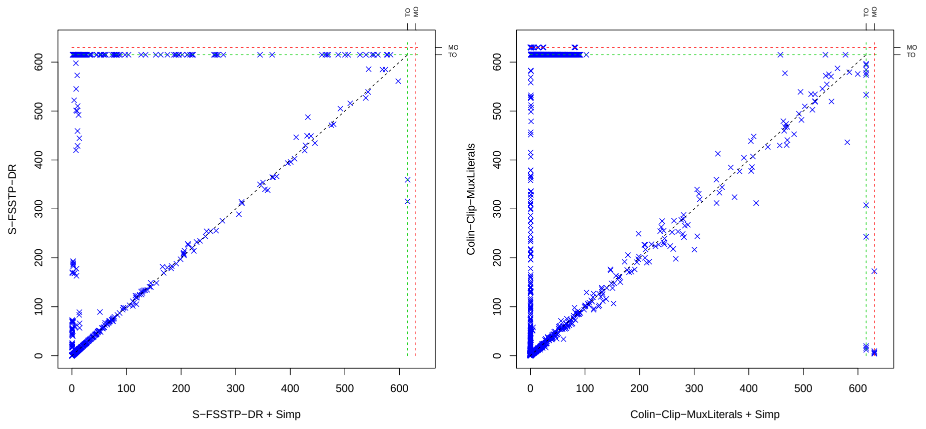


Figure 16: Impact of simplification on the DR and the Colin-Clip-MuxLiterals techniques: the scatters compare each technique with and without the simplification pre-processing on the whole benchmark set. Axes indicate run-time in seconds, each point represents the comparison of time required by the two solvers being compared. The points at the top and right edges are problems where one technique could solve the problem in the allotted time, but the other could not.

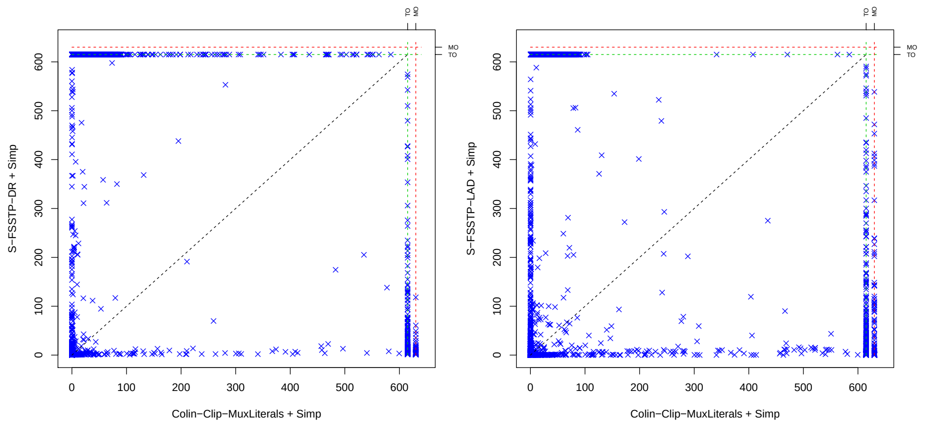
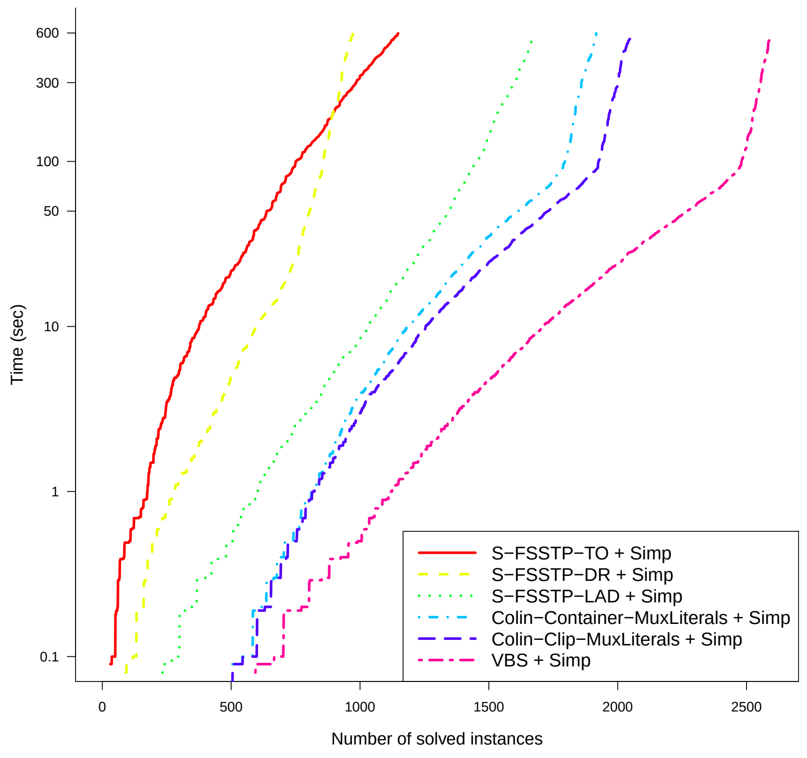


Figure 17: Comparison of DR and LAD approach with the Colin-Clip-MuxLiterals approach when pre-processing simplification is applied. Axes indicate run-time in seconds, each point represents the comparison of time required by the two solvers being compared. The points at the top and right edges are problems where one technique could solve the problem in the allotted time, but the other could not.



;

Figure 18: Log-scale cactus plot showing a comparison of the different techniques with simplification pre-processing enabled.

implemented and experimentally evaluated the approaches. The results demonstrate the complementarity of the two planning methods, and of the effectiveness of the proposed optimizations.

This work is a first step in a more comprehensive research effort on real-wold planning with durative actions. The target is a richer domain description language based on ANML [31], extended with uncertain durations, uncertain resource usage, and uncontrollable effects, all of which allow more natural representation of real-world domains and overcome the many limitations of PDDL. In particular, several directions can be explored.

First, some interesting features, such as timed initial literals with uncontrollable time windows and conditional effects, are not included in the abstract language used in this paper because they are not fundamental to explain the characteristics of the approach. Timed initial literals have been modeled as fixed-time effects that happen exogenously in every plan, but we can allow TILs having a time window in which they can controllably or uncontrollably occur.

Such a feature could be useful to model unpredictable events and can be easily handled in our framework by considering each TIL as the end effect of an action (controllable or uncontrollable) that must be started at time 0. In this view, both the approaches we presented can be extended to deal with non-fixed or even uncontrollable TILs. Conditional effects are another useful feature that can be included in our language. Unlike uncertainty in TILs, this feature requires more adaptation in order to be included in our framework. The compilation approach we presented is based on the assumption that at each point in time while executing a plan, each variable can be either certain, meaning that we know the exact value, or uncertain between exactly two values. Moreover, we know that outside of uncertainty intervals, all the variables are certain. The introduction of conditional effects changes this situation: if a conditional effect with a condition on variable f takes place while f is uncertain, the outcome of the effect is uncertain. This uncertainty is not removed at the end of the uncertainty intervals and multiple conditional effects can increase the cardinality of the set of possible uncertain values. One possible idea to deal with this problem is to extend our compilation to allow a bounded number of uncertain values for each variable, so that we can accommodate a finite number of uncertain effect occurrences. This approach would maintain the soundness of the compilation, but sacrifice completeness.

Another interesting question is whether a compilation approach can be developed to handle (uncertain) resource usage. In this paper we do not consider resources, but they are important in many planning applications. Resources can be uncertain due to different factors. Either the production or consumption of a resource depends on the duration of some uncontrollable action (E.g. the amount of fuel consumed in a trip depends on its duration, which is uncontrollable because of uncontrollable traffic conditions), or the amount being added or removed for a resource is uncertain by itself (E.g. the amount of energy produced by a solar panel depends on uncontrollable weather conditions). One idea to extend our work in this direction is to encode the resource profile with the upper and lower bounds and modifying conditions accordingly. This compilation would probably be sound but incomplete, but more research is needed to fully understand the issues.

In terms of planning problems, another challenge is to deal with the ability of the executor to observe the end of actions with uncontrollable duration. This amounts to lifting to the level of planning the TNU notion of dynamic controllability. We would like to produce plans that are able to change their course of action based on the duration of activities observed at run-time. Notice that in STPUD the start of actions is decided a-priori, and plans must achieve the goal "blindly" (which is the planning counterpart of strong controllability in TNU).

On the experimental side, we would like to better understand the impact of the intermediate effect constructions (such as clip-actions) by modifying a PDDL temporal planner to natively understand such constructions. Another direction we would like to pursue is to study the compilation technique performance using a native ANML planner.

Finally, we intend to study the plan- and domain-validation problems for the case of STPUD, by using techniques based on formal verification. In fact, validating a plan when uncertainty is present is no trivial task, and so is the development of domain models for complex applications. We want to investigate automated techniques that can help domain experts in the creation and debugging of planning models to foster the applicability of the planning technologies presented in this paper.

Acknowledgements

We thank Maria Fox and Derek Long for discussion and assistance with improving the performance of COLIN on the translation of intermediate conditions and effects. Partial support for this research was provided by the Safe Autonomous Systems Operations (SASO) Project of the NASA Airspace Operations and Safety Program, the Autonomous Systems and Operations Project (ASO) of the NASA Advanced Exploration Systems Program and by EIT Digital under activities “iLAADR” and “iLEVATOR”.

References

[1] A. Cimatti, A. Micheli, M. Roveri, Strong temporal planning with uncontrollable durations: a state-space approach, in: AAAI, 2015, pp. 3254–3260.

[2] A. Micheli, M. Do, D. E. Smith, Compiling away uncertainty in strong temporal planning with uncontrollable durations, in: IJCAI, 2015, pp. 1631–1637.

[3] M. Fox, D. Long, PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains, JAIR 20 (2003) 61–124.

[4] T. Vidal, H. Fargier, Handling contingency in temporal constraint networks: from consistency to controllabilities, J. Exp. Theor. Artif. Intell. 11 (1) (1999) 23–45.

[5] B. Peintner, K. Venable, N. Yorke-Smith, Strong controllability of disjunctive temporal problems with uncertainty, in: CP, 2007, pp. 856–863.

[6] C. Bäckström, Computational aspects of reordering plans, J. Artif. Intell. Res. (JAIR) 9 (1998) 99–137.

[7] A. Coles, M. Fox, K. Halsey, D. Long, A. Smith, Managing concurrency in temporal planning using planner-scheduler interaction, Artif. Intell. 173 (1) (2009) 1–44.

[8] A. Coles, A. Coles, M. Fox, D. Long, Forward-chaining partial-order planning, in: ICAPS, 2010, pp. 42–49.

[9] A. Coles, A. Coles, M. Fox, D. Long, COLIN: Planning with continuous linear numeric change, JAIR 44 (2012) 1–96.

[10] R. Dechter, I. Meiri, J. Pearl, Temporal constraint networks, Artif. Intell. 49 (1-3) (1991) 61–95.

[11] A. Cimatti, A. Micheli, M. Roveri, Solving strong controllability of temporal problems with uncertainty using SMT, Constraints 20 (1) (2015) 1–29.

[12] C. W. Barrett, R. Sebastiani, S. A. Seshia, C. Tinelli, Satisfiability modulo theories, in: Handbook of Satisfiability, IOS Press, 2009, pp. 825–885.

[13] P. Morris, A structural characterization of temporal dynamic controllability, in: CP, 2006, pp. 375–389.

[14] P. Santana, B. Williams, A bucket elimination approach for determining strong controllability of temporal plans with uncontrollable choices, in: AAAI, 2012, pp. 2453–2454.

[15] C. Muise, C. Beck, S. McIlraith, Flexible execution of partial order plans with temporal constraints, in: IJCAI, 2013, pp. 2328–2335.

[16] J. Bresina, R. Dearden, N. Meuleau, S. Ramakrishnan, D. Smith, R. Washington, Planning under continuous time and resource uncertainty: A challenge for AI, in: UAI, 2002, pp. 77–84.

[17] J. Frank, A. Jónsson, Constraint-based Attribute and Interval Planning, Constraints 8 (4) (2003) 339–364.

[18] A. Cesta, G. Cortellessa, S. Fratini, A. Oddi, R. Rasconi, The APSI Framework: a Planning and Scheduling Software Development Environment, in: ICAPS Application Showcase, 2009.

[19] M. Ghallab, H. Laruelle, Representation and control in IxTeT, a temporal planner, in: AIPS, 1994, pp. 61–67.

[20] R. Dearden, N. Meuleau, S. Ramakrishnan, D. E. Smith, R. Rich Washington, Incremental contingency planning, in: ICAPS'03 Workshop on Planning under Uncertainty and Incomplete Information, 2003.

[21] H. L. S. Younes, R. Simmons, Policy generation for continuous-time stochastic domains with concurrency, in: ICAPS, 2004, pp. 325–333.

[22] H. L. S. Younes, R. Simmons, Solving generalized semi-markov decision processes using continuous phase-type distributions, in: AAAI, 2004, pp. 742–748.

[23] I. Little, D. Aberdeen, S. Thiébaux, Prottle: A probabilistic temporal planner, in: AAAI, 2005, pp. 1181–1186.

[24] A. Coles, A. Coles, A. Clark, S. Gilmore, Cost-sensitive concurrent planning under duration uncertainty for service-level agreements, in: ICAPS, 2006, pp. 34–41.

[25] Mausam, D. Weld, Planning with durative actions in stochastic domains, JAIR 31 (2008) 33–82.

[26] E. Beaudry, F. Kabanza, F. Michaud, Planning for concurrent action executions under action duration uncertainty using dynamically generated bayesian networks, in: ICAPS, 2010, pp. 10–17.

[27] E. Beaudry, F. Kabanza, F. Michaud, Planning with concurrency under resources and time uncertainty, in: ECAI, 2010, pp. 217–222.

[28] M. Ghallab, D. Nau, P. Traverso, Automated planning - Theory and Practice, Morgan Kaufmann, 2004.

[29] H. Palacios, H. Geffner, Compiling uncertainty away in conformant planning problems with bounded width, JAIR 35 (2009) 623–675.

[30] A. Cimatti, A. Micheli, M. Roveri, Timelines with temporal uncertainty, in: AAAI, 2013, pp. 195–201.

[31] D. Smith, J. Frank, W. Cushing, The ANML language, in: The ICAPS-08 Workshop on Knowledge Engineering for Planning and Scheduling (KEPS), 2008.

[32] J. Rintanen, Impact of modeling languages on the theory and practice in planning research, in: AAAI, 2015, pp. 4052–4056.

[33] F. Dvorak, A. Bit-Monnot, F. Ingrand, M. Ghallab, A flexible ANML actor and planner in robotics, in: ICAPS Planning and Robotics Workshop, 2014, pp. 12–19.

[34] M. Fox, D. Long, K. Halsey, An investigation into the expressive power of PDDL2.1, in: ECAI, 2004, pp. 328–342.

[35] B. Gazen, C. Knoblock, Combining the expressiveness of UCPOP with the efficiency of Graphplan, in: ECP, Springer-Verlag, New York, 1997, pp. 221–233.

[36] J. Rintanen, Complexity of concurrent temporal planning, in: Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007, 2007, pp. 280–287.

[37] I. Tsamardinos, M. E. Pollack, Efficient solution techniques for disjunctive temporal reasoning problems, Artificial Intelligence 151 (1–2) (2003) 43 – 89. doi:10.1016/S0004-3702(03)00113-9.

[38] M. Veloso, A. Perez, J. Carbonell, Nonlinear planning with parallel resource allocation, in: Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control, Morgan Kaufmann, 1990, pp. 207–212.

[39] A. J. Coles, A. Coles, A. G. Olaya, S. J. Celorrio, C. L. López, S. Sanner, S. Yoon, A survey of the seventh international planning competition, AI Magazine 33 (1) (2012) 83–88.

[40] A. E. Howe, E. Dahlman, A critical assessment of benchmark comparison in planning, J. Artif. Intell. Res. (JAIR) 17 (2002) 1–3.

[41] A. Cimatti, A. Griggio, B. J. Schaafsma, R. Sebastiani, The MathSAT5 SMT Solver, in: TACAS, 2013, pp. 93–107.

[42] D. E. Smith, The case for durative actions: A commentary on PDDL2.1, J. Artif. Intell. Res. (JAIR) 20 (2003) 149–154.

Appendix A. Proof of Theorem 3

In this section we prove Theorem 3.

Lemma 1. *Given a strong plan σ, let $\chi = \bigcup_{s \in \sigma} snap(action(s))$. σ is valid for a STPUD if and only if the DTNU K created by algorithm 2 with DR is strongly controllable.*

Proof. Clearly, K is defined over all and only the snap actions of the action appearing in σ.

First, we prove that if σ is valid, then K is strongly controllable. Let μ be the assignment to the controllable time points of K, defined as follows.

$$\mu(x) = \begin{cases} t(x) & \text{if } x = a_{st} \text{ for some } a \\ t(a_{st} + \delta(a)) & \text{if } x = a_{et} \text{ for some controllable } a \end{cases}$$

We now prove that μ is a strong schedule for K. For the sake of contradiction, suppose it is not. Then, there exists a duration for the uncontrollable actions for which one of the free constraints in K is violated. It is impossible to violate a duration constraint, therefore one of the three constraints in Definition 13 must be violated for some $\bar{a}$. This is impossible, because σ is a valid plan and if we violate constraint 2a or constraint 2b, it means that the preconditions of the action in σ corresponding to $\bar{a}$ are unsatisfied, if we violate constraint 3, then the overall conditions of the action in σ corresponding to $\bar{a}$ are unsatisfied.

Now, we prove that if K is strongly controllable, σ is valid. Reversing the argument before, we assume we have a strong schedule μ for K, and prove that setting each step s of σ as follows, yields a valid strong plan.

- $t(s) = \mu(a_{st})$
- $\delta(s) = \mu(a_{et} - \mu(a_{st})$, if s is controllable.

where a_x indicates the snap-actions for the action s. For the sake of contradiction, assume that σ defined as above is not a valid strong plan. Then there exists a temporal plan $\pi \in I_\sigma$ that is not a valid plan for the domain in which we removed temporal uncertainty as per Definition 9. If π is invalid, it is either causally unsound (inapplicable in the initial state, not simulable, not leading to the goal state) or it violates some temporal constraint of the domain. But π cannot be causally unsound, because it fulfills all the constraints of Definition 13; and it cannot violate a temporal constraint, because the only temporal constraints in the plan are the durations of actions that are encoded in K and fulfilled by μ. □

The proof of Theorem 3 follows directly from Lemma 1.

Theorem 3 (DR Completeness)**.** *Given a STPUD problem P admitting a valid strong plan σ, if DR is used, Algorithm 2 terminates with a valid strong plan.*

Proof. We assume that the classical planner employed in Algorithm 2 is sound and complete. Therefore, sooner or later it will produce the abstract plan $\chi = \bigcup_{s \in \sigma} snap(action(s))$ as it is a plan achieving the goal $abs(P)$. Then, by Lemma 1, we know that the DR approach yields a strongly controllable DTNU, and therefore the algorithm terminates with a valid strong plan. $\square$

Appendix B. Proof of Theorem 4

In this appendix we prove Theorem 4.

Appendix B.1. Plan Mapping

Consider a plan σ for R, with actions a_i at time t_i and having duration d_i. We call π_σ the *regression plan* for P when it has actions a_i^π corresponding to the actions a_i in σ such that:

- a_i^π also starts at time t_i.
- a_i^π has duration d_i if a_i is controllable,
- otherwise the duration of a_i^π is unspecified.

Analogously, we call σ^π the *projection plan* for R of a strong plan π for P obtained by fixing the duration of each uncontrollable action in π to its maximum.

Appendix B.2. Plan Execution

Given a temporal plan, an *execution* ϵ^X of a temporal planning instance X, is a set of changes applied to the variables in time: $\epsilon \doteq \{(t_1, f_1, v_1), \cdots (t_n, f_n, v_n)\}$.

An element $(t_i, f_i, v_i) \in \epsilon^X$ means that at time t_i the variable f_i takes value v_i. Analogously to ANML, we take the view that at time t_i the change is not yet visible: the value v_i is taken immediately after t_i. Such an element of ϵ^X can be caused by either: (1) an initial condition (i.e. $t_i = 0$); (2) by a Timed Initial Literal or (3) by an action effect. Therefore, for the rest of this proof we refer to *execution elements* as either initial conditions, timed initial literals or action effects. Moreover, $\epsilon^X(f, t)$ represents the value v of f at time t during the execution ϵ^X.

Given a strong plan π for P, we have a set of possible executions, one for each possible duration of each uncontrollable action in π. For a given execution ϵ^P, then $\epsilon^P(f, t)$ indicates the value of variable f at time t during this particular execution ϵ^P.

We now need to compare the executions of plans for P and R. The next theorem states that if σ is a valid plan for R and its corresponding regression plan for R is π_σ, then the variables in each pair of executions are aligned at each time in which f_σ is equal to f during the execution of R.

Lemma 2. *Given a valid plan σ for R and its corresponding regression plan π_σ, for each execution ϵ^P of π_σ: if $\epsilon^R(f_\sigma, t) = \epsilon^R(f, t)$, then $\epsilon^R(f, t) = \epsilon^P(f, t)$, for each variable $f \in L$ and each $t \in \mathbb{R}_{\geq 0}$.*

Proof. For the sake of contradiction, let us focus on an execution ϵ^P in which there is $t \in \mathbb{R}_{\geq 0}$ such that $\epsilon^R(f_\sigma, t) = \epsilon^R(f, t)$ but $\epsilon^R(f, t) \neq \epsilon^P(f, t)$.

Let $E^R_{f_\sigma} \doteq (t^R_\sigma, f_\sigma, v^R_\sigma)$, $E^R_f \doteq (t^R, f, v^R)$ and $E^P_f \doteq (t^P, f, v^P)$ be the latest execution elements involving f_σ and f in ϵ^R and ϵ^P. By our hypothesis, $v^R \neq v^P$. Moreover, due to the translation constraints, we know that $v^R_\sigma = v^R$ and $t^R_\sigma \leq t^R$. In fact, each time we change f in R we also change f_σ either at the same time or at the beginning of an uncertain interval d and we prevent any other change on f_σ until d is over.

We prove that this hypothesis is impossible by considering all possible cases.

1. E^R_f and E^P_f are both initial conditions: since all the initial conditions are copied from P to R, we must have $v^P = v^R$.
2. E^R_f and E^P_f are both timed initial literals (TIL): since all TILs are copied from P to R, either $v^P = v^R$ or there are two TILs at the same time on the variable f, which is not allowed.
3. E^R_f is either a TIL or an initial condition and E^P_f is the effect of action a:
   - If a is uncontrollable, there is a TIL (t^R, f, v^R) also in P corresponding to E^R_f. Since by hypothesis E^P_f is the last effect on f in P, we have $t^R < t^P$. However, since E^P_f is an uncontrollable effect, there is an effect on f in R corresponding to the maximal duration of a. We indicate the time of such an effect with $t*$ and we know that $t^P \leq t*$ because $t*$ is set to the maximal duration of a that starts at the same time in both ϵ^P and ϵ^R. Now, $t*$ must also be strictly smaller than t^R, because E^R_f is the last effect on f, so we have that $t^P \leq t* < t^R$ and $t^R < t^P$, which is a contradiction.
   - If a is controllable, then there should be an effect corresponding to E^P_f for R at time t^P. Therefore, either E^R_f or E^P_f is not the latest effect modifying f in the executions of R and P, respectively.
4. E^P_f is either a TIL or an initial condition and E^R_f is an action effect: since TIL are copied from P to R, there is a TIL analogous to E^P_f in R. Hence $t^R \geq t^R_\sigma > t^P$. However, since t^R and t^R_σ are the two extremes of an uncertainty interval, there must be another effect on f in P during this interval. Hence, E^P_f is not the last effect changing f in P.
5. E^P_f and E^R_f are both effects of two action instances a and a'; there are four possible sub-cases:
   - Both E^P_f and E^R_f are effects of controllable actions (or effects of uncontrollable actions that happen at times independent of the duration): they must be the same effect since the controllable actions in P are kept in R without changing the duration nor the effect times.

- Both E^R_f and E^P_f are effects of uncontrollable actions: E^R_f and E^P_f must be the same effect of the same uncontrollable action, because of the condition imposed on f_σ. If this were not the case, then the condition preserving the value of f_σ for the whole uncertainty interval would have been violated in ϵ^R.
- E^R_f is an effect of a controllable action (or effect of an uncontrollable action happens at a time independent of the uncertain duration) while E^P_f comes from an uncontrollable action. Since E^P_f is an uncontrollable effect, there is a corresponding pair of effects on f and f_σ for the execution in R. If $t^R > t^P$ then there would be an effect on f equivalent to E^R_f that happens after t^P but before t. If E^P_f happens after E^R_f, then either $\epsilon^R(f_\sigma, t) \neq \epsilon^R(f, t)$ or E^P_f is not the latest effect on f.
- E^P_f is an effect of a controllable action (or effect of an uncontrollable action happens at a time independent of the uncertain duration) while E^R_f comes from an uncontrollable action. This is analogous to the previous case.

□

Lemma 3. *Given a strong plan π for P and its corresponding projection plan σ^π for R, let ϵ^R be the execution of σ^π. For each variable $f \in L$, each $t \in \mathbb{R}_{\geq 0}$ and each execution of P: if $\epsilon^R(f_\sigma, t) = \epsilon^R(f, t)$ then $\epsilon^R(f, t) = \epsilon^P(f, t)$,*

Proof. We apply the same reasoning as in the proof of the previous lemma. Cases 1 and 2 are identical to the previous proof, the other cases are changed as follows.

3. E^R_f is either a TIL or an initial condition and E^P_f is the effect of an action a.

   - If a is uncontrollable, there should be a TIL (t^R, f, v^R) in P corresponding to E^R_f. Given that E^P_f is an action effect, there is an execution of P where t^P happens before t^R and another in which it happens after (no two effects of the same action on the same variable can happen at the same time). Hence, π is not a valid strong plan.
   - If a is controllable, then there is an effect corresponding to E^P_f and also for R at time t^P, therefore either E^R_f or E^P_f is not the latest effect modifying f in the executions of R and P.

4. E^P_f is either a TIL or an initial condition and E^R_f is an action effect. Since TILs are copied from P to R, there must be a TIL analogous to E^P_f in R. Hence $t^R \geq t^R_\sigma > t^P$. But there are two executions of P in which $t^R = t^P$ and $t^R_\sigma = t^P$ (t^R and t^R_σ are the extremes of an uncertainty interval). Hence, E^P_f is not the last effect changing f in P.

5. E^P_f and E^R_f are both effects of two action instances a and a'.

   Four sub-cases are possible:

- Both E_f^P and E_f^R are effects of controllable actions (or effects of uncontrollable actions that happen at times independent of the uncertain action duration): they must be the same effect, as the controllable actions in P are all kept in R without changing the duration or the effect times.
- Both E_f^R and E_f^P are effects of uncontrollable actions: E_f^R and E_f^P must be the same effect of the same uncontrollable action, because actions are started at the same time in the two executions and no pair of effects on the same variable are allowed if the plan is strong.
- E_f^R is an effect of a controllable action (or happens at a time independent of the duration) while E_f^P comes from an uncontrollable action. Since E_f^R is an uncontrollable effect, there is an original effect in P that is concurrent with E_f^P. If $t^R > t^P$ then there would be an effect on f equivalent to E_f^R after t^P and before t. If E_f^P happens after E_f^R, then either $\epsilon^R(f_\sigma, t) \neq \epsilon^R(f, t)$ or E_f^P is not the latest effect on f.
- E_f^P is an effect of a controllable action (or happens at a time independent of the duration) while E_f^R comes from an uncontrollable action. This is analogous to the previous case.

□

Theorem 4 (Soundness and Completeness)**.** *Let $P \doteq \langle V, I, T, G, A\rangle$ be a planning instance and $R \doteq \langle V', I', T', G', A'\rangle$ be its translation. P has a strong plan π if and only if R has a temporal plan σ.*

Proof. Let π be a strong plan for P. σ^π is a valid temporal plan for R because:

- It achieves the goal G' of R, because all the original goals in G are achieved by π and by σ in the same way, and the goals on the shadow variables must be achieved because π is a strong plan. In fact, π achieves the goals regardless of the concrete durations of the actions, therefore it achieves them outside of the uncertainty intervals, where the variables and the shadow variables are aligned because of Lemma 3.
- Each action a' is executable in R, because each $a \in \pi$ is executable in P regardless of the action durations. Thus the possible uncertainty introduced by the durations is irrelevant for the executability of a (all the conditions are satisfied and variables and the shadow variables are aligned because of Lemma 3). In the translated instance R, all the conditions are also satisfied because the conditions are imposed via the χ function that only checks that both the variable and its shadow fulfill the original condition.
- No conflicting effects are possible because of the conditions added in $C_{a'}^E$ that prevents any modification of the interested shadow variables during the uncertainty intervals.

Similarly, let σ be a plan for R. Then π_σ is a valid strong temporal plan for P because:

- It achieves the goal G, because σ achieves the goal G' that is a super-set of G, and the variables are aligned because of Lemma 2.
- Each action a is executable in P regardless of the action duration, because the variables are aligned because of Lemma 2, $a' \in \sigma$ is executable in R and the conditions in the translated actions are a super-set of the ones in the original action, because of the χ function.
- No conflicting effects are possible regardless of the uncertain duration, because each effect at time t can be uncertain only between $\lambda(t)$ and $\nu(t)$ and we guarantee no other effect is possible in that interval by means of $C^{E}_{a'}$.

$\square$