Abstraction Refinement of Parallel Transition Systems in an SMT-based Timeline Planner

Preprint, August 30, 2022

BJØRNAR LUTEBERGET and SYNNE FOSSØY, SINTEF Digital AS, Norway

Timeline-based planning is a paradigm for domain-independent temporal planning, describing multiple values that change over time and the conditions for such changes. We present an algorithm for encoding a timeline-based planning problem into a satisfiability modulo theories (SMT) problem. Each timeline is represented by a separate bounded unrolling of its transition relation, giving a local state space representation per timeline, while synchronizations across timelines are represented as disjunctive temporal constraints on the scheduling variables. This combines previously known successful techniques of state space representations for classical planning problems with partial order representations for scheduling problems.

Compared to existing SMT translations, our approach is the first to use unsatisfiable cores to build the SMT formula in a conflict-directed way, producing smaller SMT problems that can be solved faster. In contrast to many other SMT-based planners, our approach requires only an off-the-shelf SMT solver's external interface, so that our implementation is relatively small, simple, and extensible, and can take advantage the full range of features and fine-tuned heuristics from high-performance SMT solvers. We have evaluated our approach by comparing performance using manual translations for benchmark problems from other SMT-based timeline solvers and IPC4 temporal domains.

1 INTRODUCTION

In the quest for higher level of autonomy in robotics, there is a need for efficient and integrated task planning of operations and actions. Task planning algorithms are becoming efficient enough to use onboard unmanned vehicles (whether surface, underwater, aerial, or ground vehicles), though there are still many challenges in the integration of planning and execution. If successful, such an integration would connect high-level decision-making all the way through to the guidance systems and control systems, enabling a higher degree of autonomy in a wide range of application areas, including off-shore installation campaigns, inspection and maintenance of subsea and marine installations, and emergency response operations.

In the mentioned application areas, there is a need for decision-making based on *when* tasks should be completed, for instance guaranteeing that a maintenance action is taken after an inspection action or that an emergency is attended to within a certain time window. Therefore, robotic planning models often need to be *temporal*, i.e. time-aware, and handle durative actions, concurrency and deadlines. Temporal planning problems, where actions may have durations and concurrency, represent both a planning problem (deciding which actions to take) and, at the same time, a scheduling problem (deciding when actions start and end).

A modelling formalism for planning problems called the Planning Domain Description Language (PDDL) ([14]) is dominating in the task planning community, and has also been extended to temporal planning. However, the formalisms and algorithms that work well for planning-heavy problems, especially classical (non-temporal) planning solved with state-based heuristic search, do not always give as good results when extended to models with a heavy temporal component.

An alternative modeling formalism more specifically focused on temporal planning is *timeline-based planning*, which has been successfully used in several planning systems, including EUROPA ([2]), and oRatio ([4]). Also, the planners IxTeT ([15]) and FAPE ([11]), use a formalism similar to timelines. In timeline-based planning, multi-valued state variables change over time and are

Authors' address: Bjørnar Luteberget, bjornar.lutberget@sintef.no; Synne Fossøy, synne.fossoy@sintef.no, SINTEF Digital AS, Forskningsveien 1, Oslo, Norway.

represented as *timelines*, consisting of a sequence of *tokens*, each token representing a time interval where a state variable takes a specific value. Constraints on the timelines are imposed as *synchronization rules*, which specify, for each token, other tokens that it depends on, and the temporal relationship between a token and its dependencies. For temporal planning problems where scheduling aspects and resource constraints play a prominent role, timelines-based formulations have been demonstrated to give a succinct and user-friendly representation for various task planning problems in e.g. space missions and marine robotics. Timelines planning is especially suitable when modeling nearly independent parallel processes, for instance when using more than one vehicle. The motivations and practical considerations around timelines modeling are described in more detail in [8, 12, 13].

Classical (non-temporal) planning problems are often solved by the *state space search* approach, where one represents a single assignment to all state variables at one point in time and then searches through possible actions and their resulting effects on the state. In contrast, timeline problems are often solved by search through possible plans, where plans are a set of tokens (time-varying values on a timeline) and their temporal dependencies, resulting in a set of timing variables and corresponding constraints. Roughly, finding the set of tokens and their dependencies corresponds to planning, and solving the resulting constraint problem over the timing variables corresponds to the scheduling. Rapidly solving these scheduling problems as the set of tokens changes during search is efficiently handled by constraint solvers (such as CSP, MILP, and SMT solvers).

The main approach to implementing timelines-based planners in the literature (e.g, [21]) is to build a custom constraints solver (i.e., a mathematical solver assigning values to variables under given constraints) and integrate it with a heuristic planning algorithm (i.e., a tree search algorithm that evaluates different state changes that can be made). Building such a solver can be a daunting task: most of the constraint-based timeline planners available today are medium to large code bases, e.g. oRatio at 12k LOC (lines of code), FAPE at 17k LOC, and EUROPA at over 100k LOC.

Knowing that the implementation of constraints solvers is a field of research in itself, we argue that making the best use of existing constraint systems, instead of implementing and maintaining bespoke solutions for one specific planning system, may give long-term benefits for timeline-based planners in terms of performance, and in added speed and flexibility when developing the planning system. A few temporal planners based on off-the-shelf SMT solvers have recently appeared, mainly LCP ([6]) and ANML SMT ([20]), where the planning problem is translated into variables and constraints representing timing and choices. In [6], the LCP planner is presented, which focuses on the partial-order plan representation, adding pairwise temporal constraints (non-overlap) between tokens on the same timeline.

In this paper, we present an approach to implementing a simple, flexible and extensible solver for timeline-based planning problems using a novel algorithm (Section 3). Compared to existing SMT-based approaches, the algorithm builds a smaller problem representation by expanding only those timelines where it is needed.

The key to more precise expanding of the formulation is a lesser-used feature of SMT solvers called *unsatisfiable core* analysis. We view each timeline as a miniature classical planning problem, similarly as in [9]. Each timeline has a *transition relation* which describes which sequence of values are allowed (as state variables are represented in SAT-based state space planning approaches [16, 18]). Figure 1 shows some examples of transitions relations drawn as graphs, and their corresponding *unrolling*, i.e., the conversion of the graphs into sequences of choices. In this representation, the non-overlap constraints on timelines become implicit, and it becomes easier to exploit the information in the transition relation to include only tokens that are reachable from the previous token.

In Figure 1, the Operation Timeline represents the state of an inspection task. Since there is only one possible transition in each state and the total number of states is fixed, so the unrolling

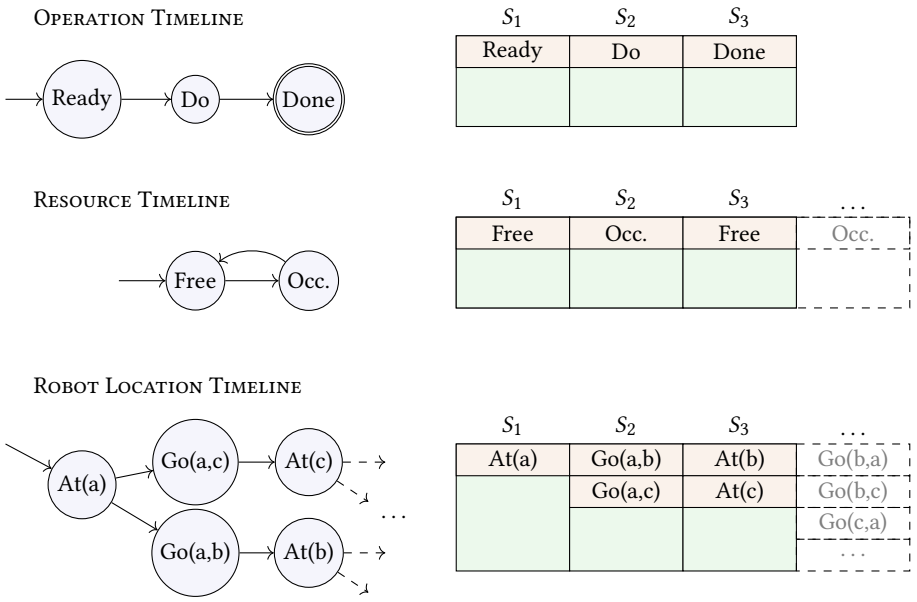

Fig. 1. Directed graphs corresponding to transition relations for different timelines (on the left) and their corresponding unrolled state sequences (on the right).

produces no choice variables (only timing variables). For the Resource Timeline and the Robot Location Timeline the number of states is unbounded and we can only unroll up to some limit. When this timeline then appears in the *unsatisfiable core*, we extend the number of states in the unrolling *of that timeline*.

Furthermore, using a high-performing off-the-shelf solver in a timeline-based planning system ensures that the latest research into constraints solving can be exploited without huge implementation efforts.

Our approach was motivated by work in two research projects: the SEAVENTION[1] project on autonomous subsea intervention using UUVs, and the ROBPLAN[2] project on autonomous mobile robot missions including inspection tasks on marine installations. Even so, the approach is fully domain-independent and can be used for any task planning application.

2 BACKGROUND

Definition 1. *A timeline t is a tuple (Π, Π_0, D, R) such that Π is a set of token types, $\Pi_0 \subseteq \Pi$ are the initial token types, $D : \Pi \to \mathbb{I}$ are the token types' duration intervals, $D(\pi) = [l_\pi, u_\pi]$, and $R \subseteq \Pi \times \Pi$ is a transition relation.*

A timeline represents a time-varying value by interpreting each token type as a different value. Each token type π must have a duration of at least l_π and at most u_π. If we have $(\pi_1, \pi_2) \in R$, then the end of a token of type π_1 can be followed by the start of a token of type π_2.

Definition 2. *A timeline planning problem P is a tuple $P = (\mathbb{T}, \mathbb{C}, F, G)$, where*

- *$\mathbb{T}$ is a set of timelines.*

[1] See https://www.sintef.no/en/projects/2018/seavention
[2] See https://www.sintef.no/en/projects/2021/robplan

- $\mathbb{C} = \{C_\pi\}$ *is a set of* synchronization conditions *for each token type* π*, where each* $c = (r_c, \Pi_c) \in C_\pi$ *consists of a temporal relation* r_c *and a set of target token types* Π_c*.*
- *The facts set* F *is a subset of token types* $F \subseteq \Pi$*, with given start and end times* τ_b, τ_e*.*
- *The goals set* G *is a subset of token types* $G \subseteq \Pi$*.*

The temporal relations typically consist of a numerical constraint involving the start and end times of tokens. In our planner implementation, we have used the Allen relations ([1]) which are often used in timeline-based planning ([17]), but the SMT encoding approach is flexible with regards to the type of temporal constraints used.

A *plan* for a timeline planning problem P consists of a set of tokens $H = \{\eta_1, \eta_2, \ldots\}$. Each token η belongs to a timeline t and is an instantiation of one of the timeline's token types $\phi(\eta) = \pi \in \Pi_t$. Each token has a start time $\tau_s^\eta \in \mathbb{R}$ and an end time $\tau_e^\eta \in \mathbb{R}$. For such a set of tokens to be a valid plan (a solution to the timeline planning problem P), the following conditions must be fulfilled:

- **Duration limits**: The duration limits of the token type are fulfilled:

$$\forall \eta \in H : l_\pi \leq \tau_e^\eta - \tau_s^\eta \leq u_\pi.$$

- **Timeline coherence**: The set of tokens belonging to the same timeline must be non-overlapping in time:

$$\forall \eta, \tilde{\eta} \in H : \left(\eta \neq \tilde{\eta} \wedge t = \tilde{t}\right) \Rightarrow \left(\tau_s^\eta \geq \tau_e^{\tilde{\eta}} \vee \tau_s^{\tilde{\eta}} \geq \tau_e^\eta\right).$$

- **Timeline transitions**: for consecutive tokens η_1, η_2 on the same timeline t, we have that $(\eta_1, \eta_2) \in R_t$ (transition relations).
- **Token conditions**: each token η of type π must fulfill each of its conditions $c \in C_\pi$ by satisfying the temporal relation r_c with another token $\tilde{\eta}$ of type $\tilde{\pi} \in \Pi_c$:

$$\forall \eta \in H : \forall c \in C_\pi : \bigvee_{\tilde{\eta} \in H | \phi(\tilde{\eta}) \in \Pi_c} r_c(\eta, \tilde{\eta})$$

  For example, the Allen relation *during*, requires that $\tilde{\eta}$ starts before η and ends after it, so if $r_c = \text{during}$, we get:

$$r_c(\eta, \tilde{\eta}) = \text{during}(\eta, \tilde{\eta}) = \left(\tau_s^{\tilde{\eta}} \leq \tau_s^\eta \wedge \tau_e^{\tilde{\eta}} \geq \tau_e^\eta\right)$$

- **Facts and goals**: for each fact f, a corresponding fixed token η_f must be included in the plan:

$$\forall f \in F : \eta_f \in H$$

  Goal tokens η_g for each goal g must be also included in the plan and their end times $\tau_e^{\eta_g}$ must be equal to the highest time value in the plan:

$$\forall g \in G : \eta_g \in H \wedge \tau_e^{\eta_g} = \max_{\tau_e^\eta | \eta \in H} \tau_e^{\eta_g}$$

See [19] for a thorough treatment of resource management in timelines-based planning. Other extended definitions of timelines problems include priced disjunctions ([4]), flexible plans and uncontrollability ([9, 17]). In this paper, we assume the planning domain to be fully deterministic. This means also that all tokens are *controllable*, i.e. their duration can be freely chosen anywhere within the duration interval. We also assume that the problems are grounded, i.e., all the token types, transition relations, and synchronization conditions are explicitly given without variables.

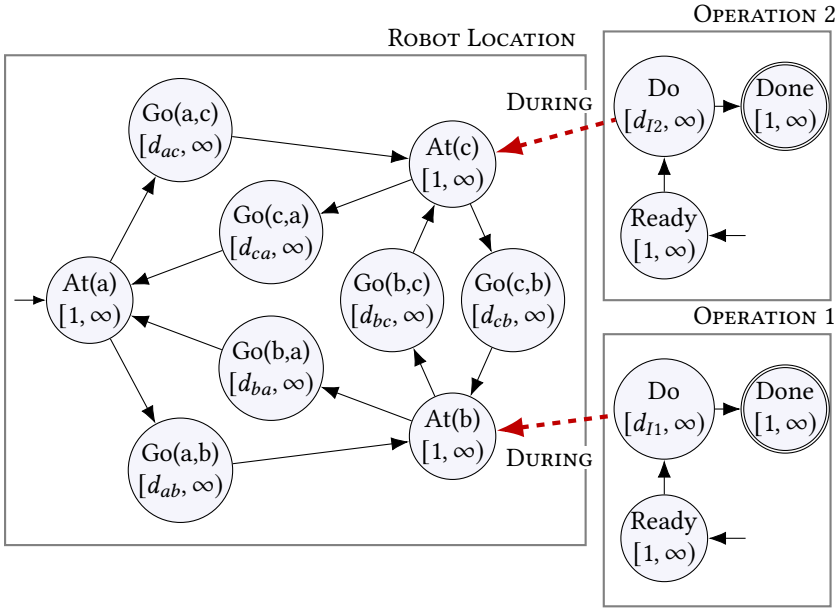

Fig. 2. A timelines representation of a planning problem with a mobile robot that needs to perform two operation tasks at different locations. Rectangular boxes indicate timelines, circles represent token types, solid arrows represent timeline transitions, and dashed arrows represent synchronization conditions.

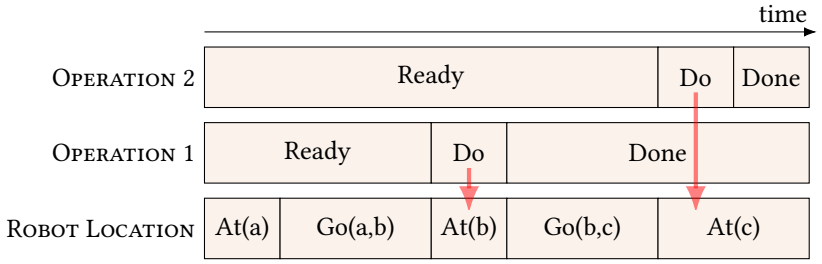

Fig. 3. A solution to the mobile robot planning problem drawn as a scheduling diagram. Timelines are on the vertical axis. Boxes correspond to tokens in the timeline and red arrows correspond to temporal dependencies that fulfil the synchronization conditions.

2.1 Mobile robot use case

The planning domain that motivated our approach to timeline-based planning was a mobile robot, e.g., an UUV or an USV, that has a set of tasks to perform at various locations, and a set of deadlines or time windows in which these tasks should be performed. This use-case is relevant for marine applications such as inspection and maintenance of equipment and off-shore installations or search and rescue.

The mobile robot's location is an illustrative example of a timeline, as the robot can only be in one place, and there is a non-trivial transition relation between the locations. A robot is either in a location that is a point of interest, `At(x)`, or it is under way from one location to another, `Go(x,y)`. The duration of an `At(x)` value is unconstrained, while the duration of a `Go(x,y)` is the time it

takes to travel from location `x` to location `y`. The corresponding transition graph is shown in the left part of Figure 2. The goals given to the robot might be to perform some actions at certain locations, such as delivering a parcel or inspecting a piece of equipment. These goals are simpler timelines with fixed tokens. The `Doing` state has a condition that it should happen while the robot is at a certain location indicated by the dashed arrows in Figure 2. A solution to a simple mobile robot problem instance is shown in Figure 3.

From this basic model, we have extended our representation of the mobile robot in various ways for different use cases, including:

- More complex **state**, for example picking up a piece of equipment from one location before performing inspection, are modelled by adding timelines and conditions.
- **Deadlines and time windows**, for example a forecast on weather conditions restricting access to certain locations during exposed operations, are modelled by adding facts, i.e. tokens with fixed start and/or end times.
- For **reusable resource** constraints, such as transferring data using an exclusive access to the communication module of the robot during remote operations, we have extended the formalism (and the input format) with resource constraints, allowing us to handle resource constraints specially in the solver. The implementation consists of adding a task-indexed resource constraint using a pseudo-Boolean encoding (see [7]). Further details about resource handling are out of scope for this paper.
- **Non-reusable resource** constraints, such as battery capacity, may also be handled similarly to reusable resources.

The mobile robot example illustrates how some timelines have complex transition graphs, and some have trivial ones. The operation timelines' simple transition relations give only fixed tokens, while the robot location timeline has is more complex.

2.2 Satisfiability Modulo Theories

Satisfiability Modulo Theories (SMT) solvers determine whether a first-order logical formula is satisfiable. By satisfiable, it is meant whether or not there exist some assignment for the formula's free variables making the logical formula true. SMT formulas consist of Boolean expressions connected by logical connectors such as negations($\neg$), conjunction($\wedge$), disjunctions($\vee$) and implications($\Rightarrow$). An example of a Boolean logical formula is $(a \wedge b) \vee (c \Rightarrow d)$. In addition to Boolean variables, the Boolean expressions of an SMT formula may be statements of different kinds, using variable types such as integers, real numbers, vectors etc., and the connectives may for instance be linear arithmetic constraints such as $a + 2b \leq 2$ or $a = c$. See [3] for an introduction to SMT.

Timeline-based planning problems involve both logical structure and temporal constraints on real variables. As such, SMT solvers are capable of handling the types of constraints needed in timeline-based planning. However, the unknown number of tokens in a solution plan makes it impossible to specify the whole planning problem directly as a (quantifier-free) SMT formula. Building the formula *incrementally* has been shown to work well for planning, especially in SAT-based classical planning (see [18]). This means that we solve a constraint system and then afterwards add more variables and constraints and solve again until the formula is able to represent a valid solution. In this context it is also possible to give the SMT solver a set of *assumptions*, which are Boolean variables that are set to a given value only temporarily, for a single solve call. When that solve call returns that the formula is unsatisfiable, the solver also reports a subset of the given assumptions that make the formula unsatisfiable. Such a subset is called an *unsatisfiable core*.

If we have a propositional logic disjunction,

$$a_1 \vee \ldots \vee a_{i-1} \vee a_i \vee \ldots \vee a_n,$$

we can represent it equivalently as the conjunction of two separate disjunctions by adding a fresh variable x:

$$a_1 \vee \ldots \vee a_{i-1} \vee x$$
$$\neg x \vee a_i \vee \ldots \vee a_n$$

Then, using a SMT solver with the assumptions interface, we may first add only the first disjunction and give the assumption $\neg x$ when solving. If x shows up in the unsatisfiable core, we can add the second disjunction as well (and remove x from the assumptions), and we are left with a problem equivalent to just adding the original disjunction from the beginning. We apply this technique to our timeline planning encoding below to extend the disjunctions for goals and synchronization conditions.

3 PARALLEL TRANSITION SYSTEMS ENCODING

The formulas in our encoding of the timeline planning problem into SMT are built from two main parts: the timeline state sequence, and the synchronization conditions. All actions happen inside the planning horizon $[T_0, T_\Omega]$.

We construct a sequence of states $\mathcal{S}_t = \langle S_t^0, S_t^1, \ldots \rangle$ for each timeline t, representing the unrolling of the transition graph of t. Note that we are not using the word *state* in classical planning sense, where the state represents the whole world: here, a state is local to an individual timeline (this is called a *stream* in [9]). Each state S_t^i represents a choice between token types $\pi_1, \pi_2, \ldots \in \Pi_t$, producing a set of corresponding tokens $\eta_1, \eta_2, \ldots$. A Boolean variable $s_{t,i}$ represents whether the state i of timeline t exists in the plan, and for each token η, a Boolean variable v_η represents whether that token is chosen for that state or not. We denote the state that contains a token η as $\zeta(\eta) = S$.

The first state S_t^0 of each timeline t is a choice between one of the initial token types Π_t^0, while the subsequent states are a choice between the token types $\tilde{\pi}$ in the transition relation $R_t(\pi, \tilde{\pi})$ for any token type π in the previous state. Since each possible token type belongs to a specific state S, we use start and end time variables ρ_s^S, ρ_e^S associated with the states instead of creating separate time variables for each of the tokens themselves.

If the set of goal tokens includes a token type $g \in G$ from timeline t, then one of the states must choose this goal value, and have an end time equal to the end of the planning horizon, T_Ω. We create a Boolean variable $\gamma_{g,\eta}$ for each token η of token type g, representing whether the token is chosen as goal state, and we require that one of tokens η is chosen as the last state of the state sequence. Also, to be able to add further alternative goal tokens, we use a sequence of extension variables $\gamma_g^{+1}, \gamma_g^{+2}, \ldots$ to make an incrementally encoded disjunction (see Section 2.2). These variables are used as assumptions in the abstraction refinement algorithm described in the section below.

Formalizing the above, the SMT formulas for state variable constraints are as follows:

- At most one token is chosen for every state

$$\forall \eta, \tilde{\eta} | (\zeta(\eta) = \zeta(\tilde{\eta})) : \neg v_\eta \vee \neg v_{\tilde{\eta}} \tag{1}$$

- If a state exists, then one of the tokens is chosen:

$$s_{t,i} \Rightarrow \bigvee_{\eta | \zeta(\eta) = S_{t,i}} v_\eta \tag{2}$$

- If state S_t^i exists, then the previous state S_t^{i-1} also exists:

$$\forall i > 0 : s_{t,i} \Rightarrow s_{t,i-1} \tag{3}$$

- If a token η of type π is chosen for its state, the token type's duration constraints are satisfied:

$$v_\eta \Rightarrow (l_\pi \leq \rho_e^{\zeta(\eta)} - \rho_s^{\zeta(\eta)} \leq u_\pi) \tag{4}$$

- The transition conditions are fulfilled:
$$\forall \eta : v_\eta \Rightarrow \bigvee_{\tilde{\eta} | R(\phi(\eta), \phi(\tilde{\eta}))} v_{\tilde{\eta}} \tag{5}$$
- For each goal g, at least one of its tokens $\eta_1, \eta_2, \ldots$ is selected, or the disjunction is extended:
$$\gamma_{g,\eta_1} \vee \gamma_{g,\eta_2} \vee \gamma_{g,\eta_3} \vee \ldots \vee \gamma_g^{+1} \tag{6}$$
- Selecting the goal state activates the token, and the goal state ends at the end of the planning horizon:
$$\forall \eta | \phi(\eta) \in G : \gamma_{g,\eta} \Rightarrow \left(v_\eta \wedge (\rho_e^{\zeta(\eta)} = t_\Omega) \right) \tag{7}$$

The use of state-space-style planning for each timeline has the advantage that the sequencing of the tokens is implicit and pairwise non-overlap constraints are not required. Also, instead of constraining the a state's start time to be equal to the previous state's end time, we can re-use the variable to make this implicit, making the number of scheduling variables for the tokens in the timeline $n + 1$ instead of $2n$, where n is the number of states instantiated in the timeline.

For each token η of type π that appears in the unrolling of the transition relation of timeline t, we create a Boolean variable $\lambda_{\eta,c}^{\tilde{\eta}}$ that represents whether condition $c \in C_\pi$ is satisfied by the token η_j of type $\tilde{\pi} \in \Pi_c$. Also, to be able to add further tokens $\tilde{\eta}$, we use a sequence of extension variables $\lambda_{\eta,c}^{+1}, \lambda_{\eta,c}^{+2}, \ldots$ to make an incrementally encoded disjunction (see Section 2.2). These variables are used as assumptions in the abstraction refinement algorithm described below.

The conditions on $\lambda_{\eta,c}^{\tilde{\eta}}$ are:

- If a token η is chosen, for each of its conditions $c \in C_{\phi(\eta)}$, a synchronization condition is required, or, the disjunction needs to be extended.
$$v_\eta \Rightarrow \lambda_{\eta,c}^{\tilde{\eta_1}} \vee \lambda_{\eta,c}^{\tilde{\eta_2}} \vee \lambda_{\eta,c}^{\tilde{\eta_3}} \vee \ldots \vee \lambda_{\eta,c}^{+} \tag{8}$$
- Enabling the synchronization condition applies a constraint between the two tokens:
$$\lambda_{\eta,c}^{\tilde{\eta}} \Rightarrow \left(v_{\tilde{\eta}} \wedge r_c(\eta, \tilde{\eta}) \right) \tag{9}$$
For example, if r_c is the Allen relation *during*, we get, in the place of $r_c(\eta, \tilde{\eta})$:
$$(\tau_s^{\zeta(\tilde{\eta})} \leq \tau_s^{\zeta(\eta)} \wedge \tau_e^{\zeta(\eta)} \geq \tau_e^{\zeta(\tilde{\eta})})$$

In combination, equations (1)-(9) describe the SMT encoding for a given fixed number of states for each timeline.

3.1 Abstraction Refinement Algorithm

The planner algorithm builds an SMT formula incrementally, based on equations (1)-(9), to handle arbitrary-length plans. Figure 4 illustrates the representations of the *encoding objects* (states, tokens, and conditions) used in the algorithm.

Algorithm 1 describes the SMT encoding algorithm. After initialization (line 1-2), the fact and goal tokens are added to their respective sequences (line 3-5). In order to add a token type to a timeline (the *addToken* function on lines 4, 13, 15), we start from the set of token types in the timeline's current last state and consider all the token types reachable from any of these last token types (i.e., following the timeline's transition graph forward by one step). This set of reachable token types are added to the tokens list, and the corresponding state encoding object is added to the states list. This process is repeated (further states are added) until the token type being added appears among the reachable tokens in the state.

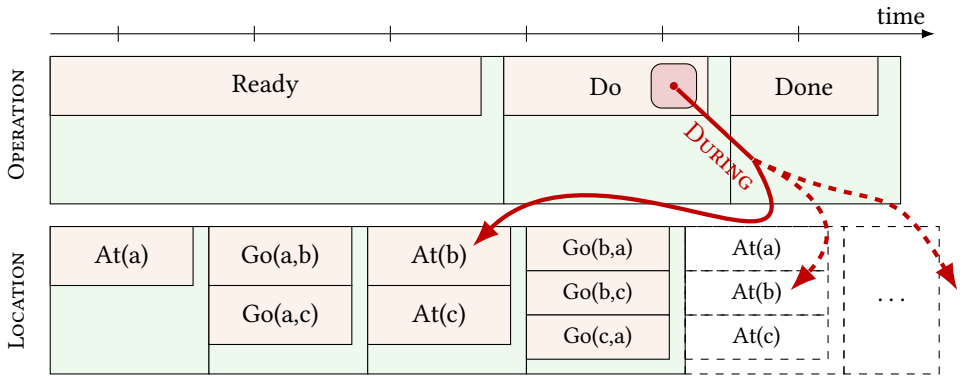

Fig. 4. Overview of the parallel transition system encoding. Green boxes represent states. The *Do* token contains a red box indicating the condition that the robot location must be At(b) during the inspection operation. The bundle of red arrows indicates alternative targets for fulfilling the synchronization (dashed tokens may be added during abstraction refinement).

Algorithm 1 SMT Timelines planner with abstraction refinement

Input: A timelines planning problem $P = (\mathbb{T}, \mathbb{C}, F, G)$

Let $\mathcal{O} = []$.
Let $\mathcal{S}$ = new SMTSolver().
for all $\pi \in F \cup G$ **do**
  $\mathcal{O}$ = addToken($\mathcal{S}, \mathcal{O}, \pi$)
end for
loop
  while any $o \in \mathcal{O}$ has state *initialized* **do**
    $\mathcal{O}$ = encode($\mathcal{S}, \mathcal{O}, o$)
  end while
  if $\mathcal{S}$.solve($[\gamma^+_{\pi_1}, \ldots, \lambda_{\eta_1,c_1}, \ldots]$) = UNSAT **then**
    for all $x \in \mathcal{S}$.get_core() **do**
      if x is a goal extension γ^+_π **then**
        $\mathcal{O}$ = addToken($\mathcal{S}, \mathcal{O}, \pi$)
      else if x is a synch. extension $\lambda^+_{\eta,c}$ **then**
        $\mathcal{O}$ = addToken($\mathcal{S}, \mathcal{O}$, typeOf(η))
      end if
    end for
    if $\mathcal{S}$.get_core() = $\varnothing$ **then**
      return UNSAT
    end if
  else
    return (SAT, $\mathcal{S}$.get_model())
  end if
end loop

Each of the encoding objects exists in one of two states: initialized or encoded. Whenever a new encoding object is created, it starts in the initialized state. Before calling the SMT solver on the

current formula, any objects in the initialized state are encoded into the SMT formula (the *encode* function line 8, based on equations (1)-(9)), which might in turn create new encoding objects, so we continue converting objects in a loop until all objects are in the encoded state (line 7-9).

When all objects are encoded, the SMT solver is called using the list of current assumptions which are the last disjunction extension variables for the goal state disjunctions and the synchronization condition disjunctions (line 10). If the formula is determined to be *unsatisfiable*, then either the whole planning problem is unsatisfiable, or the current encoding failed to include one or more tokens that were required to construct a solution plan. To analyse this, we look at the *unsatisfiable core* reported by the solver (line 10-20), which is a subset of the current assumptions. If the core is empty (line 18-20), then the planning problem is unsatisfiable. If the core contains one or more goal expansion variables (line 12-13) or condition expansion variables (line 13-14), we add more tokens of the required types to the problem, and go back to the encoding loop.

If the formula is instead determined to be satisfiable (line 21-23), the problem has been solved, and the solution can be extracted from the variable assignment returned by the SMT solver. The variable assignment determines which tokens are part of the solution, the dependencies between them, and the start and end time of each token.

4 EXTENSIONS

One of the advantages of using an SMT translation for planning is that the problem definition can be extended to describe a richer model, for example including numerical (linear or non-linear) planning or explicit resources. However, care must be taken in the encoding so that missing tokens can be identified in the unsatisfiable core, and that the encoding can be correctly incrementally extended when new tokens are added.

In this section, we describe an extension for explicitly handling resource constraints as part of the timeline planning problem. This extension was used in all of the ICP4 benchmark translations described in the performance evaluation below.

To extend the definition of timeline planning to explicitly describe reusable resource constraints, we associate a capacity to each token type, and a consumption amount to each condition.

Definition 3. *A timeline planning problem with reusable resource constraints $\mathcal{P}_R$ is a tuple $P_R = (P, r_{\rightarrow}, r_{\leftarrow})$, where*

- *$P = (\mathbb{T}, \mathbb{C}, F, G)$ is a timeline planning problem as defined in Definition 2.*
- *$r_{\rightarrow} : \mathcal{T} \rightarrow \mathbb{R}$ assigns a capacity to each token type π.*
- *$r_{\leftarrow} : \mathcal{T} \rightarrow (C_\pi \rightarrow \mathbb{R})$ assigns a consumption amount to each condition $c \in C_\pi$ for a token type π.*

Let Φ_η be the set containing a tuple $(\tilde{\eta}, q)$ of tokens and consumption amount for each possible token that has a consumption of a resource from the token type π of the token η. I.e., for conditions $c \in C_{\tilde{\pi}}$ for tokens $\tilde{\eta}$ of type $\tilde{\pi}$, and for all tokens η of the condition's token type $\pi \in \Pi_c$, if $r_{\leftarrow}(\tilde{\pi}, c) = q > 0$, then $(\tilde{\eta}, q) \in \Phi_\eta$.

For reusable resources, we add a task-indexed resource constraint using a pseudo-Boolean encoding (see [7] for details):

$$\forall(\eta_j, \_) \in \Phi_\eta : \sum_{(\eta_k, q_k) \in \Phi_\eta} q_k a(\eta_j, \eta_k) \leq r_{\rightarrow}(\pi)$$

$$\text{where } a(\eta, \tilde{\eta}) = v_\eta \wedge v_{\tilde{\eta}} \wedge (\rho_s^{\zeta(\tilde{\eta})} \leq \rho_e^{\zeta(\eta)}) \wedge (\rho_e^{\zeta(\tilde{\eta})} \leq \rho_s^{\zeta(\eta)}).$$

If the set Φ_η is extended by abstraction refinement algorithm, we encode this constraint again. We also use a special case encoding for reusable resources with capacity 1, using pairwise non-overlap

constraints instead of a pseudo-Boolean constraint:

$$\forall i : \forall j > i : \neg v_{\eta_i} \vee \neg v_{\eta_j} \vee (\rho_e^{\zeta(\eta_i)} \leq \rho_s^{\zeta(\eta_j)}) \vee (\rho_e^{\zeta(\eta_j)} \leq \rho_s^{\zeta(\eta_i)})$$

This encoding has the advantage that it can be incrementally extended when the set Φ_η is extended.

5 PERFORMANCE EVALUATION

We have used two sets of benchmark planning domains and performed a preliminary performance evaluation of our planner. Since timeline planners use different input file formats (and indeed, there is not yet any standard file format for timelines planning problems), we have manually translated the problems from the original input format to our planner's custom input format.

We have created two sets of benchmark planning domains for performing empirical evaluation of the performance of our planner. We have used three of the oRatio planner's benchmark problems.We have hand-encoded these problems to our own planner, and also into SMT directly (shown as *manual SMT* in the table), as a test for how close the timelines encoding comes to a manual SMT encoding. The planning domains are:

- **Cooking**: a pure scheduling problem with a resource constraint. The problem involves cooking a dish with several ingredients and preparation steps. Additionally, to cook the individual ingredients and the dish, requires exclusive access to one of the available cooking plates.
- **Ceramic**: a set of ceramic items need to be baked, treated, and assembled into composite items, which again are baked. The baking requires space in an oven, and the oven can only bake for a specific duration before it needs to be shut down and re-started.
- **Rover**: a mobile robot traveling to locations, taking pictures, and uploading pictures in certain time windows. It can upload only one picture at a time, which is a modelled as a reusable antenna resource.

The results in Table 1 indicate that our planner outperforms oRatio on the three benchmark problems (`cc`, `tms`, and `goac`).

Table 2 compares our planner to FAPE ([11]) on our translations of the temporal versions of `airports`, `satellite` and `pipesworld` from the fourth International Planning Competition (IPC4). Also here, our planner performs competitively with the state of the art.

- **Airport**: planning the movement of airplanes in an airport, requiring both exclusive and non-exclusive access to different road segment. The airplanes operate in different modes (push-back, move, parked, airborne).
  In our timelines formulation, each airplane is represented by a single timeline whose values are a 3-tuple consisting of mode (push-back, move, parked, airborne), road segment, and direction (traveling north/south/etc.).
- **Satellites**: planning to take measurements by calibrating instruments, turn into the correct position, take the measurement, and transfer the data. In our formulation, separate timelines represent the target that each satellite is pointing to, the state of each instrument (off, starting, calibrating, measuring, etc.), and the state of each measurement goal (measuring, sending, finished).
- **Pipesworld**: planning the flow of oil products through a pipeline network discretized into *batches* of equal size, and in order to remove a batch from one end of a pipe, another batch must be inserted in the other end.
  In our timelines formulation, there is one timeline for each batch, representing its location (e.g., a section of a pipe). Pipe parts are used exclusively as reusable resources. Removing a batch from a pipe segment requires simultaneous movement into that segment.

Table 1. Performance comparison between oRatio and our planner. The cells show the running time (in secs). T/O indicates > 60 secs. Note that problem models are translated by hand into our solver's format.

Instance	oRatio	our planner
cc, 1 plate, 5 dishes	0.109	0.026
cc, 1 plate, 50 dishes	10.682	0.294
cc, 1 plate, 100 dishes	T/O	1.880
cc, 2 plates, 50 dishes	11.185	0.298
cc, 2 plates, 100 dishes	T/O	1.860
tms, 1 oven, 2 items	0.073	0.014
tms, 2 ovens, 4 items	0.849	0.021
tms, 2 ovens, 6 items	5.921	0.037
tms, 4 ovens, 6 items	15.234	0.029
tms, 5 ovens, 10 items	T/O	0.158
goac, 5 locs., 3 time windows	0.252	0.056
goac, 5 locs., 5 time windows	0.300	0.027
goac, 7 locs., 3 time windows	0.469	0.049
goac, 7 locs., 5 time windows	0.973	0.045
goac, 9 locs., 3 time windows	1.589	0.142
goac, 9 locs., 5 time windows	1.420	0.154

Table 2. Performance comparison between FAPE and our planner. Cells show running time (in secs). T/O indicates > 300 secs. Note that problem models are translated by hand into our solver's format

Instance	FAPE	our planner
Airports-2, 1 vehicle, 17 locations	10.4	0.4
Airports-4, 1 vehicle, 40 locations	73.8	0.2
Airports-6, 2 vehicles, 40 locations	T/O	1.5
Airports-8, 3 vehicles, 40 locations	T/O	18.7
Airports-10, 1 vehicle, 44 locations	142.9	0.2
Satellite-2, 2 tools, 8 directions	4.3	24.5
Satellite-4, 3 tools, 10 directions	6.2	0.9
Satellite-6, 5 tools, 11 directions	6.7	1.0
Satellite-8, 10 tools, 15 directions	41.0	19.4
Satellite-10, 11 tools, 17 directions	T/O	17.1
Pipesworld-2, 2 pipes, 6 deliveries	32.2	0.7
Pipesworld-4, 2 pipes, 8 deliveries	9.4	0.4
Pipesworld-6, 2 pipes, 10 deliveies	T/O	0.8
Pipesworld-8, 2 pipes, 12 deliveries	T/O	6.1
Pipesworld-10, 2 pipes, 14 deliveries	T/O	3.8

Note, however, that the manual translation of the problems means that the performance evaluation does not directly compare the algorithms on exactly the same input, and the input language and modeling choices will have an effect on the running times. Even so, these results are promising for the practical use of our solver, and for a more direct comparison in the future using a standard input format.

6 CONCLUSIONS

We have presented a novel algorithm for encoding timeline planning problems into SMT. The approach performs well on many planning domains, compared to SMT-based approaches from the recent literature, possibly because of the competitive off-the-shelf SMT solver used instead of a custom solver. For example, the Z3 SMT solver used in our planner, uses the VSIDS branching heuristic, which is not directly applicable to flaw-directed planning algorithms.

Our approach extends earlier work on SMT-encoding of partial-order planning ([6]) by exploiting the transition relation of a timeline to generate a state space and then using partial-order causal link constraints for relations across different timelines. Our algorithm also makes use of the unsatisfiable core of the SMT solver to extend the set of tokens in the plan in a conflict-directed way.

Another approach is to encode the planning problem to linear-temporal logic and use a general LTL solver (see [10]), though it is reported that this does not scale to real-world problem sizes.

In the future, we would like to investigate better heuristics for the number of states to generate on each timeline. Extensions to controllability and flexibility could also fit into the SMT encoding paradigm, as demonstrated in [9]. Finally, future support for a standard input language for timelines, e.g. GHOST ([5]), would be good for sharing and comparing of problem instances between planners. We are also working on integrating our planner in an execution framework for autonomous inspection using mobile robots.

REFERENCES

[1] James F Allen. 1983. Maintaining knowledge about temporal intervals. *Commun. ACM* 26, 11 (1983), 832–843.

[2] Javier Barreiro, Matthew Boyce, Minh Do, Jeremy Frank, Michael Iatauro, Tatiana Kichkaylo, Paul Morris, James Ong, Emilio Remolina, Tristan Smith, et al. 2012. EUROPA: A platform for AI planning, scheduling, constraint programming, and optimization. *4th International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS)* (2012).

[3] Clark W. Barrett and Cesare Tinelli. 2018. Satisfiability Modulo Theories. In *Handbook of Model Checking*, Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem (Eds.). Springer, 305–343. https://doi.org/10.1007/978-3-319-10575-8_11

[4] Riccardo De Benedictis and Amedeo Cesta. 2020. Lifted Heuristics for Timeline-Based Planning. In *ECAI 2020 - 24th European Conference on Artif. Intel. (FAIA, Vol. 325)*, Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang (Eds.). IOS Press, 2330–2337. https://doi.org/10.3233/FAIA200362

[5] Giulio Bernardi, Amedeo Cesta, Andrea Orlandini, Alessandro Umbrico, and Marta Cialdea Mayer. 2020. A Language for Timeline-based Planning. In *2nd Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis (CEUR Workshop, Vol. 2785)*, Riccardo De Benedictis, Luca Geretti, and Andrea Micheli (Eds.). CEUR-WS.org, 53–58. http://ceur-ws.org/Vol-2785/paper9.pdf

[6] Arthur Bit-Monnot. 2018. A Constraint-Based Encoding for Domain-Independent Temporal Planning. In *Principles and Practice of Constraint Programming - 24th International Conference, CP 2018 (Lecture Notes in Computer Science, Vol. 11008)*, John N. Hooker (Ed.). Springer, 30–46. https://doi.org/10.1007/978-3-319-98334-9_3

[7] Miquel Bofill, Jordi Coll, Josep Suy, and Mateu Villaret. 2020. SMT encodings for Resource-Constrained Project Scheduling Problems. *Comput. Ind. Eng.* 149 (2020), 106777. https://doi.org/10.1016/j.cie.2020.106777

[8] Amedeo Cesta, Simone Fratini, and Federico Pecora. 2008. Unifying planning and scheduling as timelines in a component-based perspective. *Archives of Control Science* 18, 2 (2008), 231–271.

[9] Alessandro Cimatti, Andrea Micheli, and Marco Roveri. 2013. Timelines with Temporal Uncertainty. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, Marie desJardins and Michael L. Littman (Eds.). AAAI Press. http://www.aaai.org/ocs/index.php/AAAI/AAAI13/paper/view/6319

[10] Alessandro Cimatti, Andrea Micheli, and Marco Roveri. 2017. Validating Domains and Plans for Temporal Planning via Encoding into Infinite-State Linear Temporal Logic. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, Satinder Singh and Shaul Markovitch (Eds.). AAAI Press, 3547–3554. http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14311

[11] Filip Dvorak, Roman Barták, Arthur Bit-Monnot, Félix Ingrand, and Malik Ghallab. 2014. Planning and Acting with Temporal and Hierarchical Decomposition Models. In *IEEE International Conference on Tools with Artificial Intell., ICTAI 2014*. IEEE Computer Society, 115–121. https://doi.org/10.1109/ICTAI.2014.27

[12] Jeremy Frank. 2013. What is a timeline. In *Proc. of the 4th Workshop on Knowledge Engineering for Planning and Scheduling*. 31–38.

[13] Jeremy Frank and Ari K. Jónsson. 2003. Constraint-Based Attribute and Interval Planning. *Constraints An Int. J.* 8, 4 (2003), 339–364. https://doi.org/10.1023/A:1025842019552

[14] M. Ghallab, A. Howe, C. Knoblock, D. Mcdermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. 1998. PDDL—The Planning Domain Definition Language. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.212

[15] Malik Ghallab and Hervé Laruelle. 1994. Representation and Control in IxTeT, a Temporal Planner. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, Kristian J. Hammond (Ed.). AAAI, 61–67. http://www.aaai.org/Library/AIPS/1994/aips94-011.php

[16] Henry A. Kautz and Bart Selman. 1992. Planning as Satisfiability. In *10th European Conference on Artificial Intelligence, ECAI 92*, Bernd Neumann (Ed.). John Wiley and Sons, 359–363.

[17] Marta Cialdea Mayer, Andrea Orlandini, and Alessandro Umbrico. 2016. Planning and execution with flexible timelines: a formal account. *Acta Informatica* 53, 6 (2016), 649–680.

[18] Jussi Rintanen. 2012. Planning as satisfiability: Heuristics. *Artif. Intell.* 193 (2012), 45–86. https://doi.org/10.1016/j.artint.2012.08.001

[19] Alessandro Umbrico, Amedeo Cesta, Marta Cialdea Mayer, and Andrea Orlandini. 2018. Integrating resource management and timeline-based planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI.

[20] Alessandro Valentini, Andrea Micheli, and Alessandro Cimatti. 2020. Temporal Planning with Intermediate Conditions and Effects. In *AAAI 2020*. AAAI Press, 9975–9982. https://ojs.aaai.org/index.php/AAAI/article/view/6553

[21] Gérard Verfaillie, Cédric Pralet, and Michel Lemaître. 2010. How to model planning and scheduling problems using constraint networks on timelines. *Knowl. Eng. Rev.* 25, 3 (2010), 319–336. https://doi.org/10.1017/S0269888910000172