Journal of Artificial Intelligence Research 53 (2015) 541-632 Submitted 01/15; published 07/15

ITSAT: An Efficient SAT-Based Temporal Planner

Masood Feyzbakhsh Rankooh FEYZBAKHSH@CE.SHARIF.EDU
Gholamreza Ghassem-Sani SANI@SHARIF.EDU
Computer Engineering Department,
Sharif University of Technology,
Azadi ave., Tehran, Iran

Abstract

Planning as satisfiability is known as an efficient approach to deal with many types of planning problems. However, this approach has not been competitive with the state-space based methods in temporal planning. This paper describes ITSAT as an efficient SAT-based (satisfiability based) temporal planner capable of temporally expressive planning. The novelty of ITSAT lies in the way it handles temporal constraints of given problems without getting involved in the difficulties of introducing continuous variables into the corresponding satisfiability problems. We also show how, as in SAT-based classical planning, carefully devised preprocessing and encoding schemata can considerably improve the efficiency of SAT-based temporal planning. We present two preprocessing methods for mutex relation extraction and action compression. We also show that the separation of causal and temporal reasoning enables us to employ compact encodings that are based on the concept of parallel execution semantics. Although such encodings have been shown to be quite effective in classical planning, ITSAT is the first temporal planner utilizing this type of encoding. Our empirical results show that not only does ITSAT outperform the state-of-the-art temporally expressive planners, it is also competitive with the fast temporal planners that cannot handle required concurrency.

1. Introduction

Temporal planning is an extension of classical planning where actions are durative rather than being instantaneous. The introduction of durative actions adds a new dimension to solving planning problems, namely reasoning about time. Temporal reasoning is per se very different from causal reasoning, because time is a real-valued quantity, whereas the causal aspects of planning are normally represented by propositions.

The current standard language for defining temporal planning problems is PDDL2.1 (Fox & Long, 2003). Although PDDL+ (Fox & Long, 2002) has been introduced to the planning community as a more expressive language for defining temporal and numerical planning problems, throughout this paper, we will focus on PDDL2.1, because the planning problems that we have tackled do not need the expressive power of PDDL+. In PDDL2.1, actions can have separate preconditions and effects upon starting and ending. Each temporal action can also have some invariants, which must be preserved during the execution of that action. An important subset of problems defined by PDDL2.1 are problems for which every valid plan includes the concurrent execution of two or more actions. This subset is called the problems with required concurrency. It has been shown that the concurrent execution of two actions may be necessary for solving some temporal problems (Halsey, Long, & Fox, 2004; Cushing, Kambhampati, Mausam, & Weld, 2007). For instance, in some temporal planning problems, actions may require a proposition that is only available during the execution of another action. In such cases, these two actions must be executed concurrently. A more specific example is given in Section 2, where we describe the Driverlogshift domain. A common approach in many planners that are not temporally expressive is to eliminate such cases by compressing all temporal actions to create non-durative classical actions.

In this paper, we describe ITSAT, a temporally expressive SAT-based (i.e., satisfiability based) planner. ITSAT uses an approach that takes advantage of parallel execution semantics without rendering it incomplete for the problems with required concurrency. In this approach, the durations of all actions of a given problem are first abstracted out. This is done by breaking each temporal action into two starting and ending instantaneous events. Then the obtained temporally abstract problem is encoded into a SAT formula using novel ∀-step and ∃-step semantics for causally valid plans. We show how these semantics can be used to encode a given temporal planning problem into a SAT formula. Classical ∀-step and ∃-step encoding methods have been introduced before (Rintanen, Heljanko, & Niemelä, 2006). In addition to extending these methods to the temporal planning context, we also introduce a new encoding method based on ∃-step semantics for causally valid plans. We show that our new encoding often results in a significant reduction of the number of required steps.

After generating a causally valid plan, ITSAT performs a scheduling phase. During this phase, ITSAT tries to satisfy those temporal constraints that are imposed by considering the durations of actions. This is done by solving a Simple Temporal Problem (STP) (Dechter, Meiri, & Pearl, 1991). However, for problems with required concurrency, the posed STP may be inconsistent. In such cases, the cause of inconsistency, which manifests itself as a negative cycle in the corresponding Simple Temporal Network (STN), is detected. ITSAT then generates a number of clauses that if added to the SAT formula, collectively prevent the reoccurrence of the particular negative cycle that has occurred and other similar cycles. This process is repeated until a temporally valid plan is found.

Similar to other SAT-based planners, ITSAT takes advantage of a preprocessing phase to extract some information about the structure of problems. Such information is used throughout the encoding phase to produce a formula whose satisfiability can be checked more efficiently by the SAT solver. In Section 3, we describe the preprocessing phase of ITSAT, which includes the reasoning about mutual exclusion and the so-called safe action compression (Coles, Coles, Fox, & Long, 2009). Two propositions are regarded as mutually exclusive if they can never be jointly true in the same state of a valid temporal plan. Here, we show that one can detect the mutually exclusive propositions of temporal problems by using planning graph analysis (Blum & Furst, 1997). It is known that employing mutual exclusion reasoning can significantly improve the performance of SAT-based planners (Gerevini & Schubert, 1998).

As we mentioned earlier, ITSAT breaks down each temporal action into two starting and ending instantaneous events. Although in some cases such breaking down might be necessary for producing concurrent plans, there are situations where this is not necessary for finding valid plans. In Section 3, we show that by using the mutual exclusion information, one can identify those temporal actions that can safely be compressed into a classical action without falsifying the validity of temporal plans. This analysis results in a smaller number of distinct events and therefore a simpler planning problem.

We empirically show that by taking advantage of its preprocessing, encoding, and scheduling phases, not only does ITSAT significantly outperform state-of-the-art temporally expressive planners, it is also competitive with the best temporally simple planners which are incapable of solving problems with required concurrency property. The components of ITSAT are shown in Figure 1. In this figure, the processing components of ITSAT system are shown by rectangular blocks, while links represents the data produced and received by these components.

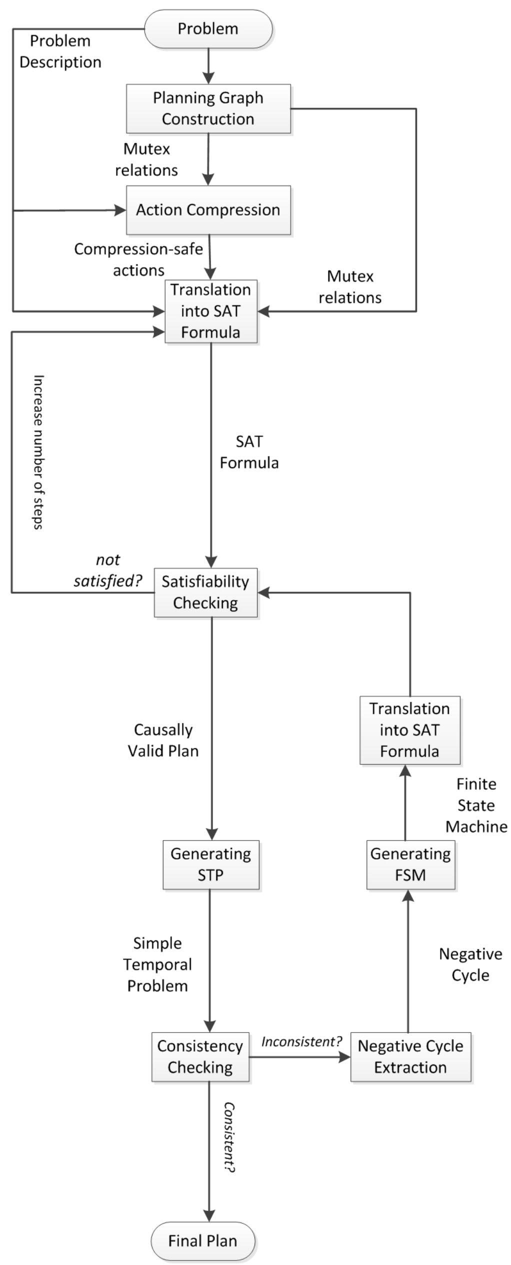

Figure 1. The block diagram of ITSAT

1.1 Motivation

As we mentioned earlier, temporal planners need to reason about time, which is a continuous quantity. Nevertheless, causal structures of problems in temporal planning are still very similar to those in classical planning. The existence of abundant temporal planning domains that also have classical versions can be regarded as an evidence to this claim. This suggests that temporal planners can benefit from using approaches that have been previously shown to be effective in dealing with classical problems.

The usage of Boolean satisfiability checking is a well-known paradigm in tackling classical planning problems (Kautz & Selman, 1992). In this approach, a given planning problem is translated into a formula in propositional logic. Each variable of the SAT formula typically represents the occurrence of the corresponding action or proposition in a certain place of potential plan. The causal constraints of the planning problem are represented by a number of ground clauses. The output plan is assumed to have a finite number of steps. Each step may include one or more actions. The original SAT-based planner allowed only one action per step (Kautz & Selman, 1992). However, the previously introduced SAT-based planners allow multiple occurrence of actions in each step. The produced formula is given as the input to an off-the-shelf SAT solver, which tries to find a model for it. If such a model is found, a plan is extracted from it. Otherwise, the number of steps in the output plan is increased by one and the corresponding SAT formula is given again to the SAT solver. This process is repeated until a valid plan is extracted or some predefined termination condition is reached. In order to obtain an efficient SAT-based planner, one important issue that should be considered is how to encode the given planning problem into a SAT formula.

SAT-based planning was originally used to find optimal plans, i.e., plans with minimum number of actions (Kautz & Selman, 1992). To guarantee the optimality of the output plan, the formula must include certain clauses to ban each step from containing more than one action. However, in the so-called satisficing planning, in which optimality is not the main objective, forcing single-action steps is not necessary. An alternative approach is to consider actions that can be executed in parallel in each step of the output plan (Ernst, Millstein, & Weld, 1997). Exploiting such parallelism can result in a smaller number of steps in the SAT formula. Another important benefit of producing compact formulae is lower memory requirements. Several encoding methods have been introduced to take advantage of action parallelism. These encoding methods are based on the so-called $\forall$-step and $\exists$-step semantics of valid plans (Rintanen et al., 2006).

The $\forall$-step and $\exists$-step semantics are different in the extent of action parallelism that they allow to occur in each step. The $\forall$-step semantics allows a set of actions to be executed in parallel, only if those actions can be executed in every possible ordering without affecting the validity of the plan. The $\exists$-step semantics, on the other hand, imposes a weaker restriction: for each step of a plan, there must exist at least one possible ordering in which the actions can be executed without falsifying the validity of the plan. It should be clear that the $\exists$-step semantics potentially allows more parallelism than is permitted by the $\forall$-step semantics. In fact, by taking advantage of $\exists$-step semantics, the most efficient SAT-based classical planner, i.e., M_p (Rintanen, 2012), is competitive with the state-of-the-art state-space planners. In this paper, we show that the separation of causal and temporal reasoning phases of temporal planning enables us to employ these compact encodings for efficient temporal planning.

1.2 Related Work

Previous research in the field of temporal planning has benefited enormously from employing well-developed classical planning strategies. For instance, many successful temporal planners have utilized the ideas of partial order planning, e.g. VHPOP (Younes & Simmons, 2003) and CPT (Vidal & Geffner, 2006). Planning graph analysis has also been adopted by temporal planners such as TGP (Smith & Weld, 1999) and TPSYS (Garrido, Fox, & Long, 2002). Some other temporal planners have embedded temporal reasoning into heuristic state space search. TFD (Eyerich, Mattmüller, & Röger, 2009), LPG-td (Gerevini, Saetti, & Serina, 2006), and POPF (Coles, Coles, Fox, & Long, 2010) are the successful instances of this latter approach.

The usage of Boolean satisfiability checking is one of the well-known paradigms in tackling classical planning problems (Kautz & Selman, 1992). In order to obtain an efficient SAT-based planner, one important issue that should be considered is how to encode the given planning problem into a SAT formula. In fact, devising efficient encoding methods has been an important research trend in the field of SAT-based planning. Examples of efficient encodings are: the split action representation (Kautz & Selman, 1996; Ernst et al., 1997; Robinson, Gretton, Pham, & Sattar, 2009), the SAS+ based encoding (Huang, Chen, & Zhang, 2012), and the compact mutual exclusion representation (Rintanen, 2006). Based on parallel semantics of plans, another effective encoding method has been introduced (Rintanen et al., 2006). This latter encoding method is of particular interest in this paper.

Satisfiability checking has also been employed in the field of temporal planning. However, SAT-based temporal planners do encounter a major challenge: representing temporal aspects of problems. Since time is a continuous quantity, it cannot be treated in the exact same way in which discrete causality is handled. To tackle this problem, STEP (Huang, Chen, & Zhang, 2009), SCP2 (Lu, Huang, Chen, Xu, Zhang, & Chen, 2013), and T-SATPLAN (Mali & Liu, 2006) use a discrete representation of time. These planners assign explicit discrete time labels to each step of the encoding. Generally speaking, in this approach, each step i is exactly one time unit ahead of step $i + 1$. As a result, if an action with duration d starts in step i, it is forced to end in step $i + d$. One immediate outcome of such an approach is the introduction of an enormous number of steps into the encoding, many of which will not contribute to the output plan. This drawback of the explicit time representation causes STEP, SCP2, and T-SATPLAN to be inefficient in terms of both speed and memory usage. To obtain a better performance, SCP2 uses $\exists$-step semantic to allow causal relations between actions in each time point (Lu et al., 2013).

TM-LPSAT (Shin & Davis, 2005), which has been designed to solve planning problems defined in PDDL+ (Fox & Long, 2002), is another SAT-based planner capable of handling temporal planning problems. Similar to STEP and T-SATPLAN, TM-LPSAT attaches time labels to each step. However, in TM-LPSAT, these labels are not predefined discrete numbers. Instead, each label is a numeric variable whose value will be determined after the problem is solved by an SMT solver (Armando & Giunchiglia, 1993). This approach can result in encodings which are more compact than those produced by STEP and T-SATPLAN.

A major disadvantage of assigning a time label to each step of the formula is that the parallelism mentioned above cannot be exploited effectively. That is because when two events are to happen in a certain step of the plan, their time labels must be the same, and thus, they must be simultaneous when the final plan is executed. This compulsory simultaneity is a restriction that reduces the number of events that can happen in each step of the final plan, which in turn, increases the number of steps needed for solving input problems. This implies that all the efficiency gain one could obtain from using parallel execution semantics will be sacrificed to achieve an easy way to deal with temporal constraints. However, in the majority of current temporal planning problems, satisfying temporal constrains is not the hardest task in finding a valid plan. It was shown that for the problems without required concurrency, one can omit the temporal constraints altogether, find a causally valid plan, and then, by considering temporal constraints only in a postproccessing step, schedule the actions of that plan to find a temporally valid plan (Cushing et al., 2007). This approach has actually been used by many previous temporal planners including YAHSP3-mt (Vidal, 2014), the winner of the temporal satisficing track of IPC 2014. Despite being efficient in solving many temporal problems, such planners are incomplete, as they are incapable of solving problems with required concurrency.

In addition to classical planning problems, SAT-based methods have also been used to deal with other categories of planning problems. Examples are planning under uncertainty (Castellini, Giunchiglia, & Tacchella, 2003), cost-optimal planning (Robinson, Gretton, Pham, & Sattar, 2010) and numerical planning (Hoffmann, Gomes, Selman, & Kautz, 2007).

2. Preliminaries

The standard language used for defining temporal planning problems is PDDL2.1 (Fox & Long, 2003). Figure 2 presents an example of the PDDL2.1 representation of a temporal planning domain. This domain, which is a simplified version of Driverlogshift (Halsey, 2004), will be referred to several times throughout this paper. As Figure 2 shows, in PDDL2.1, each action can have separate conditions and effects upon starting and ending. The starting and ending conditions (or effects) of an action are specified by the `at start` and `at end` tokens, respectively. Each action may also have some conditions that need to be preserved during execution. These conditions are specified using the `over all` token. Moreover, the duration of each action is defined by an `(= ?duration x)` statement, where `x` is a rational number or a function specifying the actual duration of that action.

Driverlogshift is the temporal version of the Driverlog domain from IPC3. As in its classical counterpart, in Driverlogshift, the objective is to transfer several objects from their original places to their destinations. Each object can be loaded into and unloaded from a certain truck by using the `LOAD` and `UNLOAD` operators, respectively. A truck can move between locations using the `MOVE` operator. The main difference between Driverlogshift and Driverlog is that the trucks here must be rested during the intervals between their working shifts. Each working shift is defined by the `WORK` operator, which produces the `(working truck)` proposition upon starting, and deletes that proposition upon ending. `LOAD`, `UNLOAD`, and `MOVE` have `(working truck)` as their invariant. Once `(working truck)` is deleted by the ending of `WORK`, it may be reproduced by the `REST` operator, which defines the resting shift of a certain truck.

```
(define (domain driverlogshift)
    (:requirements :typing :durative-actions)
    (:types
        location locatable - object
        truck obj - locatable)
    (:predicates
        (at ?obj - locatable ?loc - location)
        (in ?obj1 - obj ?obj - truck)
        (link ?x ?y - location)
        (working ?t - truck)
        (need-rest t - truck)
        (rested t - truck))

    (:durative-action WORK
        :parameters
            (?truck - truck)
        :duration (= ?duration 100)
        :condition (and
            (at start (rested ?truck)))
        :effect (and (at start (working ?truck))
            (at end (not (working ?truck)))
            (at start (not (rested ?truck)))
            (at end (need-rest ?truck))))

    (:durative-action REST
        :parameters
            (?truck - truck)
        :duration (= ?duration 20)
        :condition (and
            (at start (need_rest ?truck)))
        :effect (and
            (at start (not (need_rest ?truck)))
            (at end (rested ?truck))))

    (:durative-action LOAD
        :parameters
            (?obj - obj
            ?truck - truck
            ?loc - location)
        :duration (= ?duration 10)
        :condition (and
            (over all (at ?truck ?loc))
            (over all (working ?truck))
            (at start (at ?obj ?loc)))
        :effect (and
            (at start (not (at ?obj ?loc)))
            (at end (in ?obj ?truck))))

    (:durative-action UNLOAD
        :parameters
            (?obj - obj
            ?truck - truck
            ?loc - location)
        :duration (= ?duration 10)
        :condition (and
            (over all (at ?truck ?loc))
            (over all (working ?truck))
            (at start (in ?obj ?truck)))
        :effect (and
            (at start (not (in ?obj ?truck)))
            (at end (at ?obj ?loc))))

    (:durative-action MOVE
        :parameters
            (?truck - truck
            ?loc-from - location
            ?loc-to - location)
        :duration (= ?duration 50)
        :condition (and
            (at start (at ?truck ?loc-from))
            (at start (link ?loc-from ?loc-to))
            (over all (working ?truck)))
        :effect (and
            (at start (not (at ?truck ?loc-from)))
            (at end (at ?truck ?loc-to)))))
```

Figure 2. PDDL2.1 description of Driverlogshift domain

Note that the version of Driverlogshift described in Figure 2 is slightly different from its original version (Halsey, 2004), in which there are drivers that can walk to, board, and disembark trucks. Furthermore, the `REST` and `WORK` actions are performed by the drivers rather than trucks. However, in order to make the examples simpler, we have merged drivers and trucks into a single entity of just trucks.

A simple example of problems in Driverlogshift is shown in Figure 3. In this problem, there are three locations (`s0`, `s1`, and `s2`), one truck (`truck1`), and one object (`package1`). In the initial state, `truck1` and `package1` are at `s0`. The objective of the problem is to transfer `package1` to `s2`.

2.1 Formalism of PDDL2.1

Now we present our formalism of the specifications of PDDL2.1. Our formalism is devised in a way that simplifies the description of the preprocessing, encoding, and scheduling phases of ITSAT. We should mention that our formalism has some limitations compared to the full specifications of PDDL2.1. These limitations will be discussed in details in Section 2.2.

```
(define (problem DLOG)
     (:domain driverlogshift)
     (:objects
          truck1    - truck
          package1  - obj
          s0 s1 s2  - location)
     (:init
  (rested truck1)

          (at truck1 s0)

          (at package1 s0)

          (link s0 s1)
          (link s1 s0)
          (link s2 s1)
          (link s1 s2)
          )
     (:goal (and
          (at package1 s2))))
```

Figure 3. PDDL2.1 description of a problem in Driverlogshift domain

Definition 1 (events). An event, e, is a triple $(pre(e), add(e), del(e))$, where $pre(e)$, $add(e)$, and $del(e)$ are three sets of atomic propositions (facts) representing preconditions, positive effects, and negative effects of e, respectively.

Definition 2 (temporal actions) A temporal action, a, is a quadruple ($start(a)$, $end(a)$, $inv(a)$, $dur(a)$), where $start(a)$ and $end(a)$ are two events denoting the starting and ending events of a, $inv(a)$ is a set of atomic propositions representing the invariants of a, and $dur(a)$ is a positive rational number specifying the duration of a.

Example 1. Figure 4 shows a temporal action $a = \texttt{LOAD}(\texttt{package1}, \texttt{truck1}, \texttt{s0})$, which is an instance of the `LOAD` operator defined in Figure 2. In Figure 4, a is depicted by a rectangular box. Conditions and effects of a are written above and below the box, respectively. The `at start` conditions and effects of a are placed at the left hand side of the box, and the `at end` conditions and effects are placed at the right hand side of the box. The `over all` conditions of a are placed at the middle of the box. Here, $start(a)$ and $end(a)$ are two events, where $pre(start(a)) = \{(\texttt{at package1 s0})\}$, $add(start(a)) = \emptyset$, $del(start(a)) = \{(\texttt{at package1 s0})\}$, $pre(end(a)) = \emptyset$, $add(end(a)) = \{(\texttt{at package1 truck1})\}$, and also $del(end(a)) = \emptyset$. Moreover, we have $inv(a) = \{(\texttt{at truck1 s0}), (\texttt{working truck1})\}$, and $dur(a) = 10$.

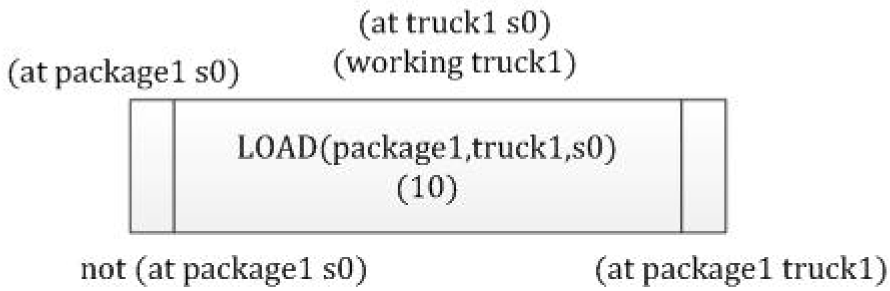

Figure 4. A temporal action

Definition 3 (temporal states). A temporal state, s, is a pair $(state(s), agenda(s))$, where $state(s)$ is a classical planning state represented by a set of atomic propositions, and $agenda(s)$ contains a finite set of open actions (i.e., actions started prior to s and not yet ended).

Definition 4 (applicability). The starting event e of an action a is applicable in state s, if the following conditions hold:

(1) $state(s)$ contains all the preconditions of e and all the invariants of a (except for those invariants of a that are added by e): $pre(e) \bigcup (inv(a) - add(e)) \subseteq state(s)$

(2) a is not already open in s: $a \notin agenda(s)$

(3) e does not delete the invariants of any open action of s:
$\bigcup_{a' \in agenda(s)} inv(a') \bigcap del(e) = \emptyset$

The ending event e of an action a is applicable in state s, if the following conditions hold:

(1) $state(s)$ contains all the preconditions of e: $pre(e) \subseteq state(s)$

(2) a is open in s: $a \in agenda(s)$

(3) e does not delete the invariants of any open action of s (other than a):
$\bigcup_{a' \in agenda(s)-\{a\}} inv(a') \bigcap del(e) = \emptyset$

Definition 5 (successors). If the starting event e of action a is applicable in state s, it will change s to the unique state s' satisfying the following conditions:

- The set of open actions of s' is equal to the set of open actions of s with a: $agenda(s') = agenda(s) \bigcup \{a\}$
- All positive and negative effects of e are respectively added to and deleted from s': $state(s') = (state(s) - del(e)) \bigcup add(e)$

If the ending event e of action a is applicable in state s, it will change s to the unique state s' satisfying the following conditions:

- The set of open actions of s' is equal to the set of open actions of s without a: $agenda(s') = agenda(s) - \{a\}$
- All positive and negative effects of e are respectively added and deleted in s': $state(s') = (state(s) - del(e)) \bigcup add(e)$

From now on, we may use $succ(s, e)$ to represent the successor state s' obtained by applying e to s.

Definition 4 and Definition 5 can be easily extended to also cover any sequence of events: $succ(s, \langle e_1, ..., e_n\rangle) = succ(succ(s, \langle e_1, ..., e_{n-1}\rangle), e_n)$, and $succ(s, \langle\rangle) = s$. A sequence of events $\langle e_1, ..., e_n\rangle$ is applicable in a temporal state s, if $succ(s, \langle e_1, ..., e_n\rangle)$ is defined.

Example 2. Let s be a temporal state such that

$$state(s) = \{(\texttt{at package1 s0}),\ (\texttt{at truck1 s0}), (\texttt{working truck1}), (\texttt{link s0 s1})\}$$

and

$$agenda(s) = \emptyset.$$

Let $a = \texttt{LOAD}(\texttt{package1}, \texttt{truck1}, \texttt{s0})$ and $a' = \texttt{MOVE}(\texttt{truck1}, \texttt{s0}, \texttt{s1})$ be two temporal actions, which are respectively instances of the LOAD and MOVE operators presented in Figure 2. the event $start(a)$ is applicable in s and it changes s to s' such that

$$state(s') = \{(\texttt{at truck1 s0}), (\texttt{working truck1}), (\texttt{link s0 s1})\}$$

and

$$agenda(s') = \{\texttt{LOAD}(\texttt{package1}, \texttt{truck1}, \texttt{s0})\}.$$

Now, $start(a)$ is not applicable in s' because a is already open in s'. Nor is$start(a')$ applicable in s', as it deletes (at truck1 s0) that is an invariant of a, which is still open in s'. However, $end(a)$ is applicable in s' and changes it to s'' such that

$$state(s'') = \{(\texttt{at package1 truck1}),\ (\texttt{at truck1 s0}), (\texttt{working truck1}), (\texttt{link s0 s1})\}$$

and

$$agenda(s'') = \emptyset.$$

Definition 6 (temporal problems). A temporal problem, $\mathcal{P}$, is a triple (I, G, A), where I, representing the initial state, is a temporal state such that $agenda(I) = \emptyset$. G is a set of atomic propositions denoting the goal conditions, and A is the finite set of all possible temporal actions of $\mathcal{P}$.

Definition 7 (causally valid plans). Let $\mathcal{P} = (I, G, A)$ be a temporal problem and $\pi = \langle e_1, ..., e_n\rangle$ be a sequence of events where for each i, e_i is a starting or an ending event of an action in A. π is a causally valid plan for $\mathcal{P}$, if it is applicable in I, $G \subseteq state(succ(s, \pi))$, and $agenda(succ(s, \pi)) = \emptyset$.

Definition 8 (pairing events). Let $\pi = \langle e_1, ..., e_n \rangle$ be a causally valid plan for problem $\mathcal{P} = (I, G, A)$. Assume that e_i and e_j are respectively the starting and ending events of a certain action $a \in A$ and $i < j$. If for all k such that $i < k < j$, e_k is neither the starting nor the ending event of a, we say that e_i (e_j) is the pairing event of e_j (e_i) in π. In other words, e_i and e_j are pairing events in π if they are related to the same occurrence of a in π.

Definition 9 (valid temporal plans and makespan). Let $\pi = \langle e_1, ..., e_n \rangle$ be a causally valid plan for $\mathcal{P} = (I, G, A)$, and $\tau_\pi : \{1, ..., n\} \to \mathbb{Q}$ be a scheduling function for π, where $\mathbb{Q}$ is the set of rational numbers. (π, τ_π) is a valid temporal plan for $\mathcal{P}$ if τ_π has the following properties:

- For all i, $\tau_\pi(i) < \tau_\pi(i+1)$.
- For each $a \in A$, if $start(a) = e_i$, and e_j is the pairing event of e_i, then $\tau_\pi(j) = \tau_\pi(i) + dur(a)$.

The maximum value assigned by τ_π to the events of π is called the *makespan* of π.

Example 3. Consider the problem $\mathcal{P} = (I, G, A)$ depicted in Figure 3, where $state(I)$ and G contain the propositions listed after the labels `:init` and `:goal`, respectively, and A is the set of all possible instantiations of operators presented in Figure 2 with the objects listed after label `:objects` of Figure 3. Let

$$\begin{aligned}\pi = \langle & start(\texttt{WORK}(\texttt{truck1})),\\ & start(\texttt{LOAD}(\texttt{truck1}, \texttt{package1}, \texttt{s0})),\\ & end\ (\texttt{LOAD}(\texttt{truck1}, \texttt{package1}, \texttt{s0})),\\ & start(\texttt{MOVE}(\texttt{truck1}, \texttt{s0}, \texttt{s1})),\\ & end\ (\texttt{MOVE}(\texttt{truck1}, \texttt{s0}, \texttt{s1})),\\ & start(\texttt{MOVE}(\texttt{truck1}, \texttt{s1}, \texttt{s2})),\\ & end\ (\texttt{MOVE}(\texttt{truck1}, \texttt{s1}, \texttt{s2})),\\ & start(\texttt{UNLOAD}(\texttt{truck1}, \texttt{package1}, \texttt{s2})),\\ & end\ (\texttt{UNLOAD}(\texttt{truck1}, \texttt{package1}, \texttt{s2})),\\ & end\ (\texttt{WORK}(\texttt{truck1}))\rangle\end{aligned}$$

A schematic representation of π is depicted in Figure 5. A straightforward checking shows that π is a causally valid plan for $\mathcal{P}$. However, π is not a valid temporal plan because the duration of `WORK(truck1)` is 100, π requires the serial execution of two `MOVE` actions, one `LOAD` action, and one `UNLOAD` action, with the total duration of 120, while `WORK(truck1)` is still open. In other words, one single working shift of `truck1` is not sufficient to transfer `package1` from `s0` to `s2`. Therefore, no scheduling function τ_π with the properties of Definition 9 exists for π. A valid temporal plan for $\mathcal{P}$ is depicted in Figure 6. In this plan, two working shifts of `truck1` are used.

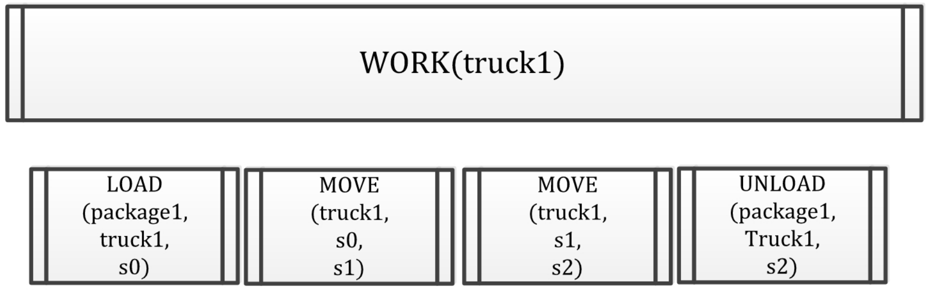

Figure 5. A causally valid plan

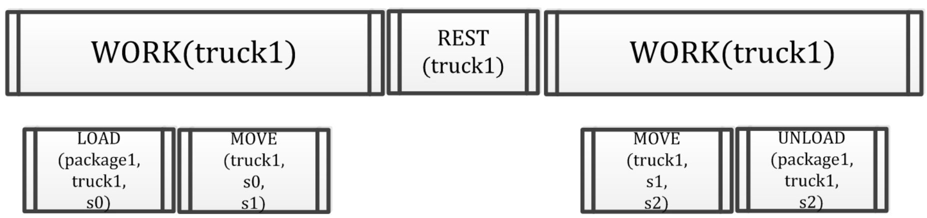

Figure 6. A valid plan

2.2 Limitations

We end this section by describing the differences between our formalism of valid temporal plans and that of PDDL2.1. The main limitations of our formalism are listed below:

- According to Definition 4, the starting event of an action a is applicable in the state s only if a is not already open in s. This means that, similar to many previous temporal planners, we do not permit two versions of the same action to overlap. Consequently, the current implementation of ITSAT does not allow self-overlapping actions. However, the specification of PDDL2.1 allows plans to have such actions, which have been shown to be necessary for solving certain temporal problems (Fox & Long, 2007). Our experimental results indicate that this restriction does not render ITSAT incapable of solving current benchmark problems. Nevertheless, it has been shown that, in theory, having self-overlapping actions may cause the complexity of temporal planning to become EXPSPACE-hard rather than PSPACE-hard (Rintanen, 2007).

- Our formalism does not allow two or more events to be simultaneously applied to any state. As an example of cases where such a simultaneity is required, consider two temporal actions a and b, such that the starting event of a adds an invariant of b, and the starting event of b adds an invariant of a. In this case, it might be necessary to simultaneously apply the starting event of both actions to a given state. It is not clear from the specification of PDDL2.1 whether such a simultaneity is permitted or not. On the other hand, it has been shown that almost none of current benchmark problems require such a simultaneity for being solvable (Rankooh & Ghassem-Sani, 2013).

- PDDL2.1 allows the usage of numerical variables. This is not supported by ITSAT. PDDL2.1 also allows duration dependent effects and state dependent durations for actions in numerical planning problems. These features are not supported by ITSAT either; because ITSAT does not currently handle numerical fluents.

- According to our formalism, the duration of a temporal action is defined by an `(= ?duration x)` assignment, where `x` is a rational number or a function specifying the actual duration of that action. PDDL2.1, on the other hand, also allows using inequalities such as `(≥ ?duration x)` and `(≤ ?duration x)` to define a range for the duration of any temporal action. Nevertheless, the current benchmark problems do not include such inequalities. Although the current implementation of ITSAT does not support these inequalities, it is quite easy to include this feature, as these kinds of constraints on the duration of actions are handled by Simple Temporal Problems (Dechter et al., 1991).

3. Preprocessing Phase

Preprocessing is an important phase in many planners. The main objective of this phase is to extract certain information from the problem. This information can later be used to enhance search performance. One important issue that should be addressed when devising any preprocessing method is the correctness of extracted information. In other words, the constraints inferred during the preprocessing phase must be correct in the sense that, it does not cause the planner to become incapable of finding valid plans. Moreover, for a preprocessing method to be effective, it is required to be performed in at most polynomial time. In this section, we explain two different preprocessing methods used by ITSAT: mutual exclusion analysis and action compression. We also formally prove that these methods are both correct and can be performed in polynomial time.

3.1 Mutual Exclusion Analysis

Mutual exclusion analysis is a preprocessing method to find pairs of propositions that cannot be mutually true in any state of a valid plan. SAT-based planners typically add an explicit clause to their SAT formula for each pair of mutually exclusive propositions. Such clauses prevent mutually exclusive pairs of propositions from being true *true* at the same time. Although such information can be obtained through the search phase itself, by acquiring it beforehand, one can prune the search tree of the SAT solver and thereby improve performance.

Polynomial time mutual exclusion analysis for classical planning problems was originally performed by constructing planning graphs, a data structure which was introduced in GRAPHPLAN (Blum & Furst, 1997). It has been shown that the mutual exclusion information obtained from planning graphs can be quite effective in improving the performance of SAT-based planners (Gerevini & Schubert, 1998). Other methods have also been introduced to compute n-way mutexes (instead of the pairwise mutexes computed by the planning graphs). The h^n heuristic (Haslum & Geffner, 2000), which analyzes the reachability of any set of n propositions from the initial state, is an example of such methods. It has been shown that a generalization of the h^n heuristic can be efficiently computed by using a syntactic regression operation (Rintanen & Gretton, 2013).

The method used by ITSAT for finding mutual exclusion relations is based on the planning graph analysis. A classical planning graph is a layered structure. The first layer includes all the propositions that are present in the initial state of the problem. In each layer of planning graph, mutual exclusion (*mutex*) relations between pairs of proposition are computed. Two propositions are non-mutex in the first layer if and only if they are both present in the initial state. An action is applicable in a layer if all its preconditions are non-mutex in that layer. Two different actions are mutex in layer i, if at least one of the following conditions holds: 1) they have interference with each other (i.e., one action deletes any effect of the other action), 2) they have conflict with each other (i.e., one action deletes any precondition of the other action), or 3) their preconditions are mutex in layer i. Layer $i+1$ includes all the effects of the actions applicable in layer i. Two propositions that are mutex in layer i become non-mutex in layer $i+1$ if they are produced by non-mutex actions of layer i. To transfer propositions from one layer to the next layer, there exists a special $noop_p$ action for each proposition p that both requires and adds p. The construction of planning graph may continue until no change take places between two consecutive layers. In that case, we say the graph has leveled off.

Planning graphs have previously been employed to tackle temporal planning problems (Smith & Weld, 1999). In fact, the first completely domain-independent temporal planner called TGP, was an extension of GRAPHPLAN (Blum & Furst, 1997). TGP requires all preconditions of each temporal action to be preserved throughout the time that the action is open, and also does not allow actions to have effects upon starting. As a result, TGP is not compatible with the requirements of PDDL2.1. TPSYS (Garrido et al., 2002), an extension of TGP, is another planning graph based temporal planner that can produce plans in domains with required concurrency. Similar to GRAPHPLAN, in addition to the construction of a planning graph, both TPSYS and TGP perform a backward search for a valid temporal plan.

LPGP (Long & Fox, 2003) is another planning graph based temporal planner. In LPGP, the mutex relations between proposition and actions are computed by considering only the causal constraints of the problem; whereas the temporal constraints are taken into account later while a plan is being extracted by solving a Linear Programming (LP) problem. Omitting the temporal constraints of the problem is done by converting the given temporal problem into a classical problem. As a result, the graph construction of LPGP is very similar to that of GRAPHPLAN.

As mentioned earlier, in ITSAT, the temporal constraints of the problem are considered only after a causally valid plan is produced. Therefore, those constraints are not needed to be dealt with in the planning graph construction phase. This makes the graph structure of LPGP suitable for ITSAT. Here, we explain the graph construction phase of LPGP. The correctness of mutual exclusion information obtained by this method is essential for the correctness of our action compression and SAT encoding methods. However, the description of LPGP was not accompanied by any formal proof of correctness. Therefore, here, we formally prove the correctness and tractability of this preprocessing method.

Definition 10 (causal abstraction of temporal problems). Let $\mathcal{P} = (I, G, A)$ be a temporal planning problem and A^c be a set of classical actions such that for each $a \in A$ there are exactly three classical actions a^s, a^i, and a^e in A^c, with the following properties:

- $pre(a^s) = pre(start(a)) \cup (inv(a) - add(a))$
- $add(a^s) = add(start(a)) \cup \{open_a\}$, where $open_a$ is a new proposition specifying that a is started but not yet finished
- $del(a^s) = del(start(a)) - add(start(a))$
- $pre(a^i) = inv(a) \cup \{open_a\}$
- $add(a^i) = inv(a) \cup \{open_a\}$
- $del(a^i) = \emptyset$
- $pre(a^e) = pre(end(a)) \cup \{open_a\}$
- $add(a^e) = add(end(a))$
- $del(a^e) = (del(end(a)) - add(end(a))) \cup \{open_a\}$

The causal abstraction of $\mathcal{P}$ is the classical problem $\mathcal{P}^c = (state(I), G, A^c)$.

In fact, by Definition 10, to produce a causal abstraction of a given temporal planning problem, we split any temporal action a into three classical actions a^s, a^i, and a^e. Actions a^s and a^e correspond respectively to the starting and ending events of a. In addition to their normal effects and preconditions, a^s adds a special proposition named $open_a$, which is required and deleted by a^e. The action a^i is called the *invariant checking action* of a, and requires all invariants of a plus $open_a$ as its preconditions, and produces $open_a$ as its effect.

For a given temporal planning problem $\mathcal{P} = (I, G, A)$, ITSAT produces $\mathcal{P}^c = (state(I), G, A^c)$ *i.e.*, the causal abstraction of $\mathcal{P}$. ITSAT then constructs a classical planning graph for $\mathcal{P}^c$.

The planning graph in ITSAT is very similar to that of GRAPHPLAN. There is only one difference between the planning graphs of these two planners. In GRAPHPLAN, as mentioned earlier, all propositions are propagated through layers by the so-called *noop* actions. However, in ITSAT, there is an exception to this usage of *noop* actions: the new proposition of form $open_a$ introduced by our causal abstraction of action a. This particular proposition is propagated by a^i, the invariant checking action of a. Therefore, a^i can be seen as a new kind of *noop* action used to cover the invariants during the reasoning about mutex relations.

Theorem 1. Let $\mathcal{P} = (I, G, A)$ be a temporal planning problem and $\mathcal{P}^c = (state(I), G, A^c)$ be the causal abstraction of $\mathcal{P}$. Let $\pi = \langle e_1, ..., e_n \rangle$ be any finite sequence of events that is applicable in I, and $s_n = succ(I, \pi)$. Then the following conditions must hold:

- If two propositions p and q are both members of $state(s_n)$, then p and q are non-mutex in the layer n of the planning graph of $\mathcal{P}^c$.
- If proposition p is a member of $state(s_n)$, and action a is a member of $agenda(s_n)$, then p and $open_a$ are non-mutex in layer n of the planning graph of $\mathcal{P}^c$.

Proof. See Appendix A. □

After a planning graph has been leveled off, all the subsequent extensions of the graph has no effect on the new layers. Therefore, if two propositions are mutex in the last layer of a leveled-off graph, they will remain mutex in all subsequently produced layers. In this case, Theorem 1 implies that such pairs of propositions can never appear at the same temporal state during the execution of a valid temporal plan. The only matter that remains is to show that the mutual exclusion analysis of ITSAT can be performed in polynomial time. Let $\mathcal{P}$ be a temporal planning problem, and $\mathcal{P}^c$ be the causal abstraction of $\mathcal{P}$. It can be deduced from Definition 10, that the size of $\mathcal{P}^c$ is greater than that of $\mathcal{P}$ only by a constant factor. The process of constructing the planning graph of $\mathcal{P}^c$ can be obtained by modifying the construction process of planning graphs in GRAPHPLAN planner, in such a way that for any temporal action a, $noop_{open_a}$ is never used. GRAPHPLAN constructs its planning graphs in polynomial time (Blum & Furst, 1997). Therefore, the overall time needed for the mutual exclusion analysis of ITSAT is also polynomial in the size of any given temporal planning problem.

3.2 Action Compression

Temporal actions can have a variety of temporal relations with one another. A popular model for representing temporal relations between actions was initially introduced by James Allen (1984). The model included 13 possible temporal relations between any two actions. Some of Allen's temporal relations require the starting and/or ending events of actions to be executed simultaneously. As it was mentioned in Section 2.2, none of the temporal plans produced by ITSAT can necessitate such a simultaneity. As a result, the set of temporal relations between any two temporal actions will be confined to a proper subset of all Allen's temporal relations. These possible temporal relations are depicted in Figure 7. As it is shown in Figure 7, in 4 out of 6 types of these relations, the actions are concurrent, i.e., there exists a time in which the two actions are both being executed. Such a concurrency is unnecessary for solving some temporal planning problems. If we know that two actions are not required to be concurrently executed, in order to find a valid plan, checking only the two temporal relations depicted in Figure 7-(c) is sufficient in the searching phase of any planner. However, if all valid plans include concurrent executions of two or more actions, restricting the temporal relations of actions to just the two relations depicted in Figure 7-(c) will render the planner incomplete.

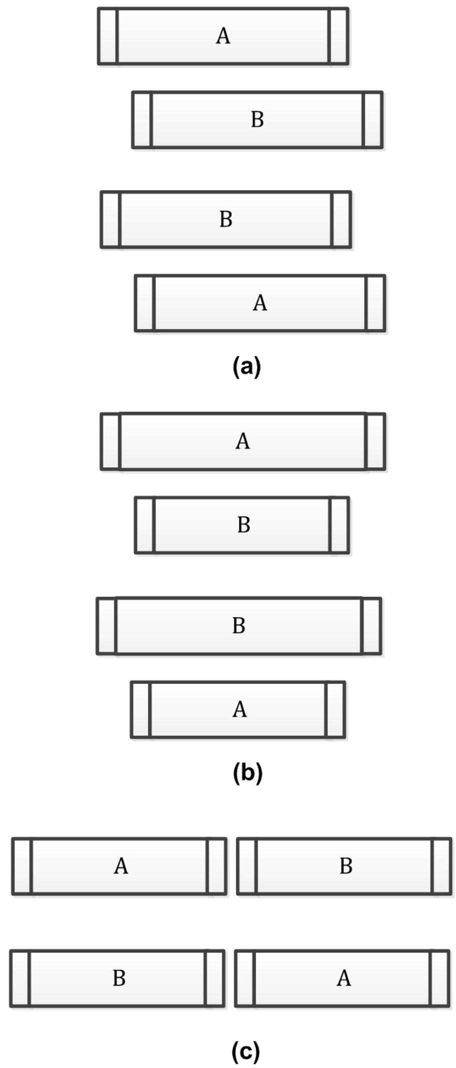

Figure 7. Temporal relations between two PDDL2.1 actions

Definition 11 (compression-safe sets of actions and compressed plans). Let $\mathcal{P} = (I, G, A)$ be a temporal planning problem for which there exists at least one valid temporal plan, and A' be a subset of A. We say A' is compression-safe for $\mathcal{P}$, if there exists a causally valid plan for $\mathcal{P}$ that is compressed with respect to A'. A causally valid plan $\pi = \langle e_1, ..., e_n \rangle$ is compressed with respect to A' if it has the following property:

- For each k, if e_k is the starting event of action $a \in A'$, then e_{k+1} is the ending event of a.

According to Definition 11, the starting and ending events of all members of A' are assumed to be executed consecutively in at least one causally valid plan. Therefore, while that plan is being executed, no other event is causally needed to happen between the starting and ending of any member of A'. This suggests that members of A' can be regarded as a single event in the environment, rather than having two separate starting and ending events.

In other words, for each member of A', we can compress the starting and ending events into a single event without rendering the problem unsolvable. As an example, consider the `DRIVERLOGSHIFT` temporal planning problem presented in Example 3. The plan π presented in Example 3 shows that the set of all `LOAD`, `MOVE`, and `UNLOAD` actions is a compression-safe set of actions for that problem. A straightforward analysis of this example shows that neither of `WORK` actions presented in Example 3 can be a member of any compression-safe subset of actions.

Note that, according to Definition 11, only a causally valid plan can be regarded as a compressed sequence of events. Although the concept of compression can be extended to cover even those sequences of events that do not lead to any goal state, for the sake of simplicity, we have focused our attention to only those sequences that are causally valid plans, and defined compression-safe actions only for solvable temporal planning problems. As we explain later in Section 4, the information obtained by our compression-safety analysis is incorporated into the encoding of the problem by adding some extra SAT formulae, which makes the problem at hand tighter. In other words, this information is only used to prune the search space of the SAT solver. As a result, our handling of compression-safety can never cause the planner to produce any (invalid) plan for an unsolvable planning problem.

Safe action compression has been employed before in the field of temporal planning (Coles et al., 2009). It has been shown that in the temporal problems that do not possess the property of *required concurrency*, all temporal actions can be safely compressed into classical actions (Cushing et al., 2007). A temporal problem is said to have *required concurrency*, if its every valid temporal plan includes at least one action whose execution overlaps with the execution of some other action. In the problems without required concurrency, all temporal actions can be compressed into classical actions. In this case, the problem is transformed into a classical planning problem. This phenomenon is completely consistent with the semantics of Definition 11, as it can be easily shown that in the problems without *required concurrency*, the set of all actions is indeed a compression-safe set of actions. However, as it is the case in Example 3, even when the problem does have the *required concurrency* property, there may still exist a non-empty compression-safe set of actions.

CRIKEY3 and its successor, POPF, are two state-space based temporal planners that detect the compression-safe actions as a preprocessing task (Coles et al., 2009). However, the concept of compression-safety in those planners is different from what we presented in Definition 11. CRIKEY3 does not assume that the ending event of a compression-safe action must be executed immediately after its corresponding starting event. Instead, once the starting event of a compression-safe action is applied to a state, using a simple inference method, CRIKEY3 can determine when to apply the corresponding ending event. This method can reduce the branching factor of the search space in state-space based temporal planning. Here, we show that by using the idea of detecting compression-safe actions, one can significantly reduce the search space of the satisfiability checking based temporal planning. As it is later explained in Section 4.4, for each compression-safe action a, we add a clause to the SAT formula to guarantee that the starting event of a is present in a step if and only if the ending event of a is present in the same step. These clauses can be used to prune the search tree when the SAT solver is checking the satisfiability of the produced formula.

CRIKEY3 considers action a to be compression-safe if the following two conditions hold:

- $pre(end(a)) \subseteq inv(a)$
- $del(end(a)) = \emptyset$

Figure 8-(a) shows a temporal plan that is executed to reach proposition q. In this example the ending event of action b does not have any precondition or delete effect. Therefore, CRIKEY3 considers b to be compression safe. However, if our goal is to produce q, the singleton $A' = \{b\}$ is not a compression-safe set by Definition 11. In fact, the method used by CRIKEY3 has been specifically devised for the state-space based temporal planners, and cannot be easily employed by the SAT-based planners such as ITSAT. In contrast, as it is later shown, our method can be easily used by both state-space based temporal planners and SAT-based planners.

There are also cases where the method used by CRIKEY3 cannot detect actions that are compression-safe according to Definition 11. Consider the plan depicted in Figure 8-(b). Suppose that proposition p is the only member of the initial state, and the goal is to produce proposition g. In this plan, actions a and b must be executed consecutively to produce g. That is because p and q, which are respectively the overall conditions of a and b are mutually exclusive, and can never be true together. However, neither a nor b has the second property required by CRIKEY3 to be regarded as a compression-safe action. In this section we show how the mutex information can be used for detecting compression-safe actions.

Definition 12 (swappable events). Let a and a' be two different temporal actions, e be the starting or ending event of a, and e' be the starting or ending event of a'. We say e and e' are swappable if all the following conditions hold:

- e and e' do not have interference with each other: $add(e) \cap del(e') = \emptyset$ and $add(e') \cap del(e) = \emptyset$.
- e and e' do not have conflict with each other: $del(e) \cap (pre(e') \cup inv(a')) = \emptyset$ and $del(e') \cap (pre(e) \cup inv(a)) = \emptyset$.
- e and e' are not supporting each other: $add(e) \cap (pre(e') \cup (inv(a') - add(e'))) = \emptyset$ and $add(e') \cap (pre(e) \cup (inv(a) - add(e))) = \emptyset$.

According to Definition 12, two events are swappable if there is no causal relation between them. This means in any causally valid plan $\pi = \langle e_1, ..., e, e', ..., e_n \rangle$, we can swap e and e' to reach another causally valid plan $\pi = \langle e_1, ..., e', e, ..., e_n \rangle$. We can use such swapping to reorder the events of a given causally valid plan without falsifying it.

Consider a causally valid plan $\pi = \langle e_1, ..., e_n \rangle$. Let e_i and e_j be the starting and ending event of the same action. If all other events of this plan are swappable with e_j, then, by repeatedly swapping, one can reorder π to produce another causally valid plan $\pi' = \langle e_1, ..., e_i, e_j, e_{i+1}, ..., e_{j-1}, e_{j+1}, ..., e_n \rangle$, in which e_i and e_j are two consecutive events. Therefore, here $\{a\}$ is a compression-safe set. In this case, we say that a is compressed towards its start. Similarly, if every event of the plan other than e_i and e_j is swappable with e_i, then, by repeatedly swapping, one can reorder π to produce the causally valid plan $\pi'' = \langle e_1, ..., e_{i-1}, e_{i+1}, ..., e_{j-1}, e_i, e_j, ..., e_n \rangle$. Once again, we can conclude that $\{a\}$ is a compression-safe set. In this latter case, we say that a is compressed towards its end.

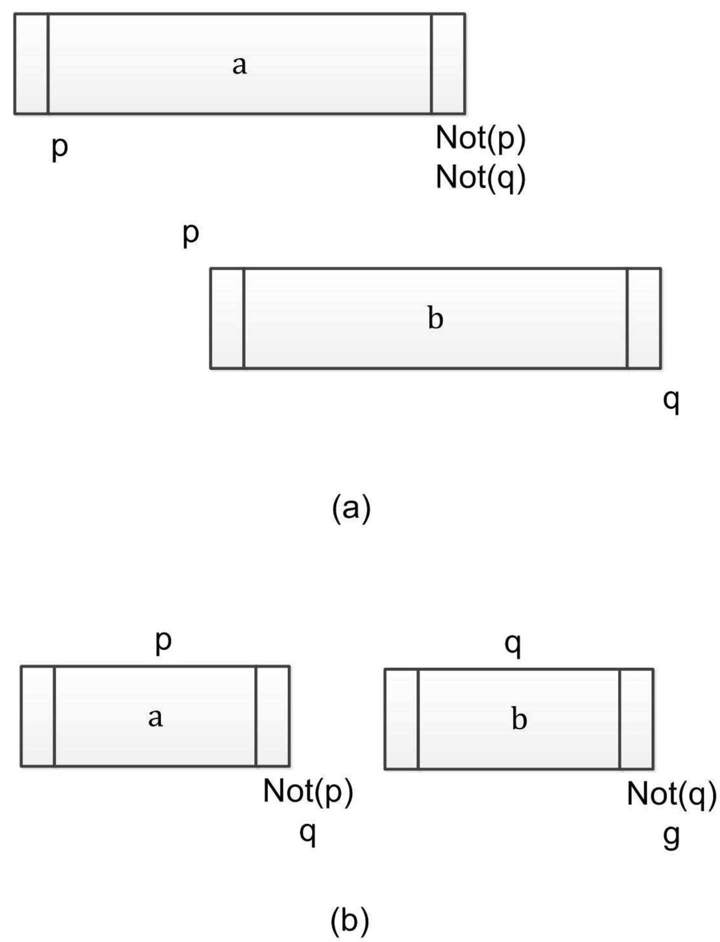

Figure 8. Temporal actions that are not regarded as compressible by ITSAT (a) and CRIKEY3 (b)

To find out whether it is safe to compress a given action a, there is no need to check if *all* events are swappable with the starting and/or ending events of a. In fact, by considering the mutex relations obtained from the planning graph of the problem, we already know that some events can never be executed while a is open. This information can be effectively used to find out if a given set of actions is compression-safe.

Definition 13 (compressible actions). Let $\mathcal{P} = (I, G, A)$ be a temporal planning problem, and $a \in A$ be a particular temporal action. We say that a is compressible towards its start, if for every event e such that e is the starting or ending event of $a' \in A - \{a\}$, at least one of the following conditions holds:

- A precondition or add effect of e is mutex with $open_a$ in the last layer of the leveled-off planning graph of the causal abstraction of $\mathcal{P}$.
- e is swappable with $end(a)$.

Similarly, we say that a is compressible towards its end, if for every event e such that e is the starting or ending event of $a' \in A - \{a\}$, at least one of the following conditions holds:

- A precondition or add effect of e is mutex with $open_a$ in the last layer of the leveled-off planning graph of the causal abstraction of $\mathcal{P}$.
- e is swappable with $start(a)$.

Theorem 2. Let $\mathcal{P} = (I, G, A)$ be a solvable temporal planning problem. Let A' be the set of every member of A that is either compressible towards its start or compressible towards its end. A' is compression-safe for $\mathcal{P}$.

Proof. See Appendix A. □

We now give an example for further clarification of this matter.

Example 4. Let $\mathcal{P} = (I, G, A)$ be a temporal planning problem, where A is the set of three temporal actions a, b, and c. Consider the hypothetical causally valid plan depicted in Figure (9-a), where the execution of action a includes the execution of action b that in turn includes the execution of action c. Assume that a is compressible towards its start, and b is compressible towards its end. We show how this plan can be converted to another causally valid plan in which a, b, and c are being executed sequentially. Figures (9-b) and (9-c) show the results of doing two consecutive swaps by which b is compressed towards its end. The starting event of b has been swapped with the starting event of c to transform the plan of Figure (9-a) into the plan of Figure (9-b). Since b is compressible towards its end, this swapping cannot result in an invalid plan. Similarly, the starting event of b has been swapped with the ending event of c to transform the plan of Figure (9-b) into the plan of Figure (9-c). Figures (9-d) to (9-g) show the results of doing four consecutive swaps by which a is compressed towards its start. As a result of doing these swaps, the fully sequential plan shown in figure (9-g) is produced. This implies that even if a planner does not allow the execution of any event while a or b is open, it will still be capable of producing the temporally valid plan of Figure (9-g).

For any given problem $\mathcal{P} = (I, G, A)$, ITSAT computes the compression-safe set A' of Theorem 2. To check the first condition of Definition 13, ITSAT needs to construct a planning graph for the causal abstraction of $\mathcal{P}$ which, as we showed in the previous subsection, can be done in polynomial time. For the second condition of Definition 13, it suffices to check every possible pair of events to see if they are swappable. Since this can be done for each pair in constant time, the total time will be $O(|A|^2)$. We conclude that finding A' can be performed in polynomial time.

The method described here for finding compression-safe actions can be used by state-space temporal planners, too. State-space temporal planners can be divided into two categories. The first category includes the planners that are based on the so-called decision epoch planning method (Cushing et al., 2007). Examples of decision epoch planners are TP4 (Haslum, 2006), SAPA (Do & Kambhampati, 2003), and TFD (Eyerich et al., 2009).

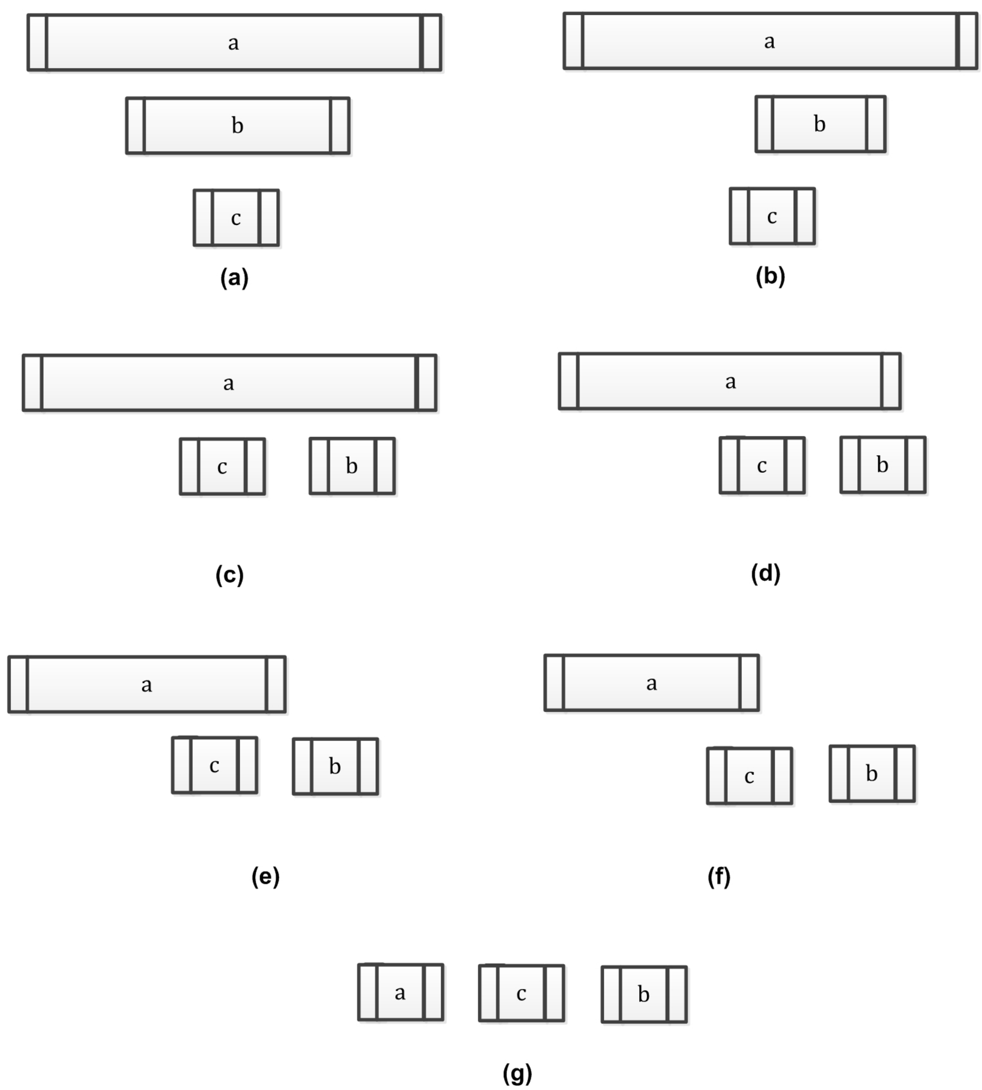

Figure 9. Action compression

In this method, the start of each action is restricted to be immediately after the start or end of another action. Each state has an explicit time-stamp. When an action is applied to a state, the starting time of the action will be set to time-stamp of that state. As a result, once the starting event of an action is added to the plan, the time of its corresponding ending event will be exactly known. When searching for a valid plan, in each state, the planner has to make a decision between either advancing to the time of the ending event of an open action, or to open a new action. However, if we know that an action is compression-safe, the planner can advance the time to the ending of that action and thereby prune the search space. Plans produced in this way might have larger makespans in comparison to those produced without pruning the search space. Nevertheless, the produced plans can be rescheduled to find plans with improved makespans by the method we explain later in Section 6.

An alternative approach for the state space search is the so-called temporally lifted progression planning, which has been proved to be complete for PDDL2.1 (Fox & Long, 2003). CRIKEY3 and POPF are examples of the planners that are using this approach. Each state in the temporally lifted progression planning represents a permutation of a number of events. At each state, the consistency of temporal constraints imposed by the sequence of events in that state is checked by solving a Simple Temporal Problem (STP). Similar to the decision epoch planning, in each state, there may exist two possible choices: to add the ending event of an open action, or to open a new action. However, for compression-safe actions, the ending event of actions can be applied immediately after the starting event, which in turn reduces the future choices of the planner. We will show in Section 6 that by taking advantage of compression-safe actions in this manner, the planner can still visit all the STPs of all causally valid permutations of events.

Table 1 shows the comparison between the average percentage of actions regarded as compression-safe by our new method and the method used in CRIKEY3 and POPF, in various temporal planning domains. We will explain more information regarding our benchmark domains and problems later in Section 6. At it can be seen in Table 1, our compression method can detect significantly more compressible actions in a number of benchmark domain.

4. Encoding Phase

In this section, we explain how the abstract causal problem associated with a given temporal problem is encoded into a SAT formula. As in classical planning, there exist more than one way to translate a particular planning problem into its corresponding SAT formula. Previous investigations in the field of classical planning show that the choice of the encoding method can have a major impact on the efficiency of a SAT-based planner. As mentioned earlier, the most successful SAT-based classical planners have used special encoding methods that are based on the so-called $\forall$-step and $\exists$-step semantics of valid plans (Rintanen et al., 2006).

In this section, we define the temporal versions of the classical $\forall$-step and $\exists$-step plans. We also show how exactly these semantics can be used to translate a given temporal planning problem into a SAT formula. We introduce a $\forall$-step encoding and two different types of $\exists$-step encodings in temporal planning. The $\forall$-step and the first $\exists$-step encoding methods are temporal versions of the classical $\forall$-step and $\exists$-step encodings. Similar to their classical versions, in these new encodings, a few restrictive simplifying assumptions are assumed to hold. Our second type of the $\exists$-step encoding, however, is obtained by relaxing one of these assumptions. As we later show, this new $\exists$-step encoding often requires fewer steps than the other one. Besides, as our experimental results show, among these new encoding methods, the second $\exists$-step encoding results in the best performance of ITSAT in terms of both speed

domain	CRIKEY3	ITSAT
zenotravel	12	100
rovers	85	100
depots	100	100
airport	0	95
pegsol	100	100
crewplanning	100	100
openstacks	100	100
elevators	100	100
sokoban	100	100
parcprinter	100	100
driverlog	100	98
floortile	100	100
mapanalyser	13	96
matchcellar	96	96
parking	100	100
rtam	88	95
satellite	100	98
storage	100	99
turnandopen	95	99
tms	73	75
driverlogshift	98	98
matchlift	95	95

Table 1: Average Percentage of Compressed Actions

and memory usage of the planner. The necessary proofs of soundness and completeness of our encoding methods are also given in this section.

4.1 Parallel Semantics for Causally Valid Plans

As mentioned earlier, the classical $\forall$-step semantics permits the parallel execution of more than one action in each step, only if the validity of the plan does not depend on the execution order of those actions. This can simply be guaranteed by adding a particular clause for each pair of mutually exclusive actions to ensure that such actions will not be included in the same step. However, this strategy does not work in temporal planning. In temporal planning, because of the temporal constraints imposed on the starting and ending events of the actions, the validity of a particular ordering of events in a certain step, also depends on the ordering of events in the other steps. Nevertheless, in ITSAT this problem has been tackled by separating the causal and temporal reasoning phases. In general, if we focus only on finding causally valid plans, and postpone the scheduling phase, the mentioned problem, about checking the feasibility of imposing different orderings of events in each step, will no longer exist. We next introduce our semantics for causally valid $\forall$-step and $\exists$-step temporal plans.

Definition 14 (temporal $\forall$-steps and $\exists$-steps). Let $E = \{e_1, ..., e_n\}$ be a set of events, and s_1 and s_2 be two temporal states. S is a temporal $\forall$-step from s_1 to s_2 only if for all one-to-one ordering functions $O : \{1, ..., n\} \rightarrow \{1, ..., n\}$ (i.e., all permutations of events), we have: $s_2 = succ(s_1, \langle e_{O(1)}, ..., e_{O(n)} \rangle)$. S is a temporal $\exists$-step from s_1 to s_2 only if there exist at least a one-to-one ordering function $O : \{1, ..., n\} \rightarrow \{1, ..., n\}$ (i.e., at least one permutation of events), such that: $s_2 = succ(s_1, \langle e_{O(1)}, ..., e_{O(n)} \rangle)$.

Definition 15 (causally valid $\forall$-step and $\exists$-step plans). Let $\mathcal{P} = (I, G, A)$ be a temporal planning problem. Suppose $s_0, ..., s_n$ is a sequence of temporal states such that $s_0 = I$, $G \subseteq state(s_n)$, and $agenda(s_n) = \phi$. If for each $1 \leq i \leq n$, $Step_i$ is a $\forall$-step ($\exists$-step) from s_{i-1} to s_i, then we call the sequence $\pi = \langle Step_1, ..., Step_n \rangle$, a causally valid $\forall$-step ($\exists$-step) plan for $\mathcal{P}$. We say that $\langle s_0, ..., s_n \rangle$ is the state transition sequence of π.

Classical $\forall$-step and $\exists$-step encodings (Rintanen et al., 2006) are based on the $\forall$-step and $\exists$-step semantics for classical valid plans, respectively. However, in the $\exists$-step encoding, for the sake of improving the efficiency of the planner, the following restrictive rules have been also enforced on the semantics.

- Rule 1: Instead of accepting all possible orderings among the actions of each step, only a fixed arbitrary ordering is allowed. As a result, by this rule, the execution of a step necessitates the execution of its actions according to this fixed ordering.

- Rule 2: Preconditions of all actions of each step must be members of the state immediately before that step. Similarly, the effects of all actions of each step must be consistent with the state reached immediately after that step.

In this section, we present one $\forall$-step and two $\exists$-step encodings for planning in causal abstractions of temporal planning. Our encodings are based on the $\forall$-step and $\exists$-step semantics for causally valid plans (Definition 15). By considering events, instead of actions, both of the above rules can be applied to temporal planning, too. While in our first $\exists$-step encoding, we respect both rules, in our second $\exists$-step encoding, the second restrictive rule is relaxed.

In fact, the second rule imposes some serious restrictions on the applicability of actions in each step. For instance, it prevents a proposition from being both produced and used in the same step of a plan. Neither does it allow the deletion and production of any particular proposition in the same step. By relaxing these restrictions, the encoding can be more compact, i.e., the relaxation permits more events to occur in each step. In classical planning, a less relaxed form of Rule 2 has been introduced so that the effects of actions in each step can be used by other actions in that step (Wehrle & Rintanen, 2007). Here, however, we totally relax Rule 2 and allow each proposition to be required, added, and deleted many times in the same step.

Before explaining our SAT encodings, we first define SAT variables and auxilary clauses commonly used in our three encoding methods. Let $\pi = \langle Step_1, ..., Step_n \rangle$ be a causally valid $\forall$-step (or $\exists$-step) plan for a given temporal planning problem $\mathcal{P} = (I, G, A)$, and $\langle s_0, ..., s_n \rangle$ be the state transition sequence of π. In order to encode $\mathcal{P}$ into a SAT formula whose model can be translated back into π, we use the following SAT variables:

- For each proposition p, and each t such that $0 \leq t \leq n$, we define a SAT variable p^t. Assigning $true$ ($false$) to p^t implies that p is (is not) a member of $state(s_t)$.
- For each action $a \in A$, and each t such that $0 \leq t \leq n$, we define a SAT variable a^t. Assigning $true$ ($false$) to a^t implies that a is (is not) a member of $agenda(s_t)$.
- For each event e such that e is the starting or ending event of an action in A, and each t such that $1 \leq t \leq n$, we define a SAT variable e^t. Assigning $true$ ($false$) to e^t implies that e is (is not) a member of $Step_t$.

If a SAT formula is satisfiable there exists a model for it. This model is a binary function that assigns a value of $true$ or $false$ to each variable of the formula in such a way that the formula is satisfied. For each of our encoding methods, if the produced formula has a model M, one can easily translate M to a corresponding causally valid ∀-step (or ∃-step) plan, using the description given above about the variables of the formula. We denote the resulting plan by $plan(M)$. For showing the correctness of a particular encoding method, two issues must be addressed. First, we must show that if there exists a causally valid plan for the temporal problem $\mathcal{P}$, then the encoding of $\mathcal{P}$ has a model. We call this the completeness of our encoding method. Second, we must show that if the encoding of $\mathcal{P}$ has a model M, then $plan(M)$ is a causally valid plan for $\mathcal{P}$. This is called the soundness of our encoding method.

Note that here, we prove the finite-horizon completeness and not the ∃-completeness for our encodings. In other words, we prove that if there exists a ∀-step (or ∃-step) plan π with l steps for a given problem, then the problem can be translated by our ∀-step (or ∃-step) encoding into a satisfiable SAT formula with at most l steps, so that the model of the formula can be translated back into π. On the other hand, proof of ∃-completeness would need the value of l to be determined. Our proofs of the finite-horizon completeness could have implied the ∃-completeness if at least an upper bound on the value of l had been determined. Recent research in the field of classical planning has shown that in some classical planning domains, tight upper bounds on the length of optimal plans can be determined (Rintanen & Gretton, 2013). However, determining such upper bounds in temporal planning is beyond the scope of the current work. To find a causally valid plan, ITSAT starts from an encoding with only one step, and sequentially produces and tries to satisfy formulae with increasing number of steps, until a satisfiable formula will be encountered or a predefined time limit will be reached.

In classical SAT-based planning, in order to produce linear-size encodings for ∀-step and ∃-step semantics of valid plans, special sets of clauses, named *chains*, have been used (Rintanen et al., 2006). Since we have also used these chains in ITSAT, the formal definition of their temporal version is given here. Let $e_1, ..., e_n$ be an arbitrary fixed ordering of all events, E and R be two sets of events, k be a natural number, and m be special symbol that assigns a unique name to the chain at hand. We define $chain(e_1, ..., e_n; E; R; k; m)$ by the conjunction of formulae (C-1) to (C-3) stated below.

(C-1) $\bigwedge\{e_i^k \rightarrow b_{j,m}^k | i < j, e_i \in E, e_j \in R, \{e_{i+1}, ..., e_{j-1}\} \cap R = \emptyset\}$

(C-2) $\bigwedge\{b_{i,m}^k \rightarrow b_{j,m}^k | i < j, \{e_i, e_j\} \subset R, \{e_{i+1}, ..., e_{j-1}\} \cap R = \emptyset\}$

(C-3) $\bigwedge\{b^k_{i,m} \rightarrow \neg e^k_i | e_i \in R\}$

The above formulae in fact encodes a message passing strategy. The symbol m specifies the name of the message and is used to distinguish the SAT variables of a certain chain from those of other chains. The number k specifies the step whose variables are affected by the message to be produced. The message may be produced by any member of E. The receivers of the message are the members of R. If $b^k_{i,m}$ is true, it means that message is received by the i-th event of the k-th step of the formula. If e_i is a member of E, and e^k_i is true, then a message will be produced and sent to e_j, which is the first member of R located after e_i in the fixed ordering. This is represented in $chain(e_1, ..., e_n; E; R; k; m)$ by the clauses of the form $e^k_i \rightarrow b^k_{j,m}$ in formula (C-1). Once the message is produced, it will be transmitted forward according to the fixed ordering by the clauses of the form $b^k_{i,m} \rightarrow b^k_{j,m}$ in formula (C-2). If an event e_i receives the message, its corresponding SAT variable will be$false$ by the clauses of the form $b^k_{i,m} \rightarrow \neg e^k_i$ in formula (C-3). In fact, the members of R that receive the message will certainly be excluded from the final plan.

We now present some examples to show how these chains are practically used to guarantee particular characteristics that the output plan will have.

Example 5. Assume that e_1, e_2, e_3, e_4 are four events. Suppose that proposition x is required by e_1 and e_4, deleted by e_2, and added by e_3. Also assume that four propositions $p_1, ..., p_4$ are respectively added by $e_1, ..., e_4$. Consider the following two cases:

- Case 1: we want to prevent proposition x from being both required and deleted in the same step, say k, of the final plan. For this purpose, we can add the conjunction of $chain(e_1, ..., e_4; E; R; k; m^x_1)$ and $chain(e_4, ..., e_1; E; R; k; m^x_2)$ to the formula, where E is the set of all events that delete x (i.e., $E = \{e_2\}$), and R is the set of all events that require x (i.e., $R = \{e_1, e_4\}$). Note that m^x_1 and m^x_2 are two symbols that enable us to distinguish between the SAT variables used in these two different chains. In this case, adding $chain(e_1, ..., e_4; E; R; k; m^x_1)$ will add the following formulae to the encoding of the problem:

  - $e^k_2 \rightarrow b^k_{4,m^x_1}$
  - $b^k_{1,m^x_1} \rightarrow b^k_{4,m^x_1}$
  - $b^k_{1,m^x_1} \rightarrow \neg e^k_1$
  - $b^k_{4,m^x_1} \rightarrow \neg e^k_4$

  Assume that there exists a model M for the produced SAT encoding such that $M(e^k_2) = true$. In this case, since M satisfies $e^k_2 \rightarrow b^k_{4,m^x_1}$, we have $M(b^k_{4,m^x_1}) = true$. Consequently, since M satisfies $b^k_{4,m^x_1} \rightarrow \neg e^k_4$, we have $M(e^k_4) = false$. In other words, if e_2 is a member of step k, then e_4 cannot be a member of the same step. Similarly, adding $chain(e_4, ..., e_1; E; R; k; m^x_2)$ will add the following formulae to the encoding of the problem:

  - $e^k_2 \rightarrow b^k_{1,m^x_2}$

- $b^k_{4,m_2^x} \rightarrow b^k_{1,m_2^x}$
- $b^k_{4,m_2^x} \rightarrow \neg e^k_4$
- $b^k_{1,m_2^x} \rightarrow \neg e^k_1$

An argument similar to the one given for $chain(e_1, ..., e_4; E; R; k; m_1^x)$ shows that after adding $chain(e_4, ..., e_1; E; R; k; x^2)$, if e_2 is a member of step k, then e_1 cannot be a member of the same step. As a result, by adding both mentioned chains to the SAT formula, if the execution of step k produces p_2, it then cannot produce p_1 or p_4. This is actually how the occurrence of conflicting actions in each step of the final plan is avoided by the linear-size classical $\forall$-step encoding (Rintanen et al., 2006).

- Case 2: we allow proposition x to be both required and deleted in a particular step k only if the deleting event does not precede the requiring event in the fixed ordering $\langle e_1, e_2, e_3, s_4\rangle$. For this purpose, we only need to add $chain(e_1, ..., e_n; E; R; k; m^x)$ to the formula, where E and R are the same as E and R in case 1. In this case, if the execution of step k produces p_2, it can also produce p_1, but not p_4. This strategy, too, was initially introduced for the linear-size classical $\exists$-step encoding (Rintanen et al., 2006).

Note that if one admits the second restrictive rule mentioned above, which is the case in classical $\forall$-step and $\exists$-step encodings, no proposition can be added by an event in any step while being deleted by another event in the same step. As a result, if the execution of step k produces p_2, it cannot produce p_3 in any of the above cases.

4.2 Temporal Versions of Classical $\forall$-step and $\exists$-step Encodings

We first present the temporal versions of the classical $\forall$-step and $\exists$-step encodings. Similar to their classical forms, in the temporal versions of these encodings, we assume an arbitrary but fixed ordering $e_1, ..., e_n$ for all events of the given temporal problem $\mathcal{P} = (I, G, A)$. We also assume that the output plan will have a fixed number of steps, denoted by l. Let $\pi = \langle Step_1, ..., Step_l\rangle$ be an output plan for $\mathcal{P}$, and $\langle s_0, ..., s_l\rangle$ be the state transition sequence of π. For each event e, let $action(e)$ be the member of A whose starting or ending event is equal to e. Let P be the set of all propositions of $\mathcal{P}$. For each proposition $p \in P$, let $E^-_p = \{e|p \in del(e)\}$, $E^+_p = \{e|p \in add(e)\}$ and $R_p = \{e|p \in pre(e)\} \cup \{e|p \in inv(action(e)) - add(e)\}$. Moreover, assume that there are two dummy events e_0 and e_{n+1}, which do not have any precondition, add effect, or delete effect.

4.2.1 The $\forall$-step Encoding

Given the temporal problem $\mathcal{P} = (I, G, A)$, we produce the SAT-formula $\phi^{\forall}_l$, which is based on the $\forall$-step semantics of causally valid plans, for $\mathcal{P}$ by the conjunction of all formulae described below.

($\forall$-1) $\bigwedge\{p^0|p \in state(I)\} \cup \{\neg p^0|p \notin state(I)\}$

($\forall$-2) $\bigwedge\{p^l|p \in G\}$

(∀-3) $\bigwedge\{\neg a^0 | a \in A\}$

(∀-4) $\bigwedge\{\neg a^l | a \in A\}$

(∀-5) $\bigwedge\{e^k \rightarrow p^{k-1} | 0 < k \leq l, p \in P, e \in R_p\}$

(∀-6) $\bigwedge\{e^k \rightarrow p^k | 0 < k \leq l, p \in P, e \in E_p^+\}$

(∀-7) $\bigwedge\{e^k \rightarrow \neg p^k | 0 < k \leq l, p \in P, e \in E_p^-\}$

(∀-8) $\bigwedge\{\neg p^{k-1} \wedge p^k \rightarrow \bigvee_{e \in E_p^+} e^k | 0 < k \leq l, p \in P\}$

(∀-9) $\bigwedge\{p^{k-1} \wedge \neg p^k \rightarrow \bigvee_{e \in E_p^-} e^k | 0 < k \leq l, p \in P\}$

(∀-10) $\bigwedge\{chain(e_1, ..., e_{n+1}; E_p^-; R_p \cup \{e_{n+1}\}; k; m_1^p) | 0 < k \leq l, p \in P\} \cup$
$\{(b^k_{n+1,m_1^p} \rightarrow \neg a^k) | 0 < k \leq l, p \in inv(a)\}$

(∀-11) $\bigwedge\{chain(e_n, ..., e_0; E_p^-; R_p \cup \{e_0\}; k; m_2^p) | 0 < k \leq l, p \in P\} \cup$
$\{(b^k_{0,m_2^p} \rightarrow \neg a^{k-1}) | 0 < k \leq l, p \in inv(a)\}$

(∀-12) $\bigwedge\{e^k \leftrightarrow \neg a^{k-1} \wedge a^k | 0 < k \leq l, a \in A, e = start(a)\}$

(∀-13) $\bigwedge\{e^k \leftrightarrow a^{k-1} \wedge \neg a^k | 0 < k \leq l, a \in A, e = end(a)\}$

Formula (∀-1) indicates that any member of $state(s_0)$ is *true* iff it is present in the initial state. Similarly, formula (∀-2) states that all members of the goal state must be true in $state(s_l)$. Formulae (∀-3) and (∀-4) imply that $agenda(s_0)$ and $agenda(s_l)$ are both empty. Formulae (∀-5) to (∀-7) show that when an event is applied to step k, its preconditions must be present in $state(s_{k-1})$, and its effects must be consistent with $state(s_k)$. Formulae (∀-8) and (∀-9) are responsible for encoding the so-called *explanatory frame axioms*: formula (∀-8) implies that if p is present after but not before step k, then there must exist at least one event in step k that has p in its add effects. Similarly, formula (∀-9) implies that if p is present before but not after step k, then there must exist at least one event in step k that deletes p. Formulae (∀-10) and (∀-11) are added to guarantee that the events of each step can be executed in any possible ordering. Formula (∀-10) implies that if p is deleted by an event e_i in step k, then p cannot be required by any other event e_j of step k such that $j > i$. It also implies that if p is deleted by any event in step k, and action a has p as an invariant, then a cannot be a member of $agenda(s_k)$. Note that in $chain(e_1, ..., e_{n+1}; E_p; R_p; k; m_1^p)$ used in formula (∀-10), the value of b_{n+1,m_1^p} indicates whether or not p is deleted by any event in step k. The reason why we are using the dummy event e_{n+1} is to have such an indicator. Analogously, formula (∀-11) implies that if p is deleted by an event e_i in step k, then p cannot be needed by any other event e_j of step k such that $j < i$. Formula (∀-11) also implies that if p is deleted by any event in step k, and action a has p as an invariant, then a cannot be a member of $agenda(s_{k-1})$. Formulae (∀-12) and (∀-13) are responsible for applying appropriate changes in the agendas of the states that are located before and after each step of the final plan. Formula (∀-12) implies that if the starting event of an action a is a member of the step k of the output plan, then a must be a member of $agenda(s_k)$ but not $agenda(s_{k-1})$. Similarly, formula (∀-13) implies that if the ending event of action a is a member of step k of the plan, then a must be a member of $agenda(s_{k-1})$ but not $agenda(s_k)$.

Theorem 3 (completeness of temporal $\forall$-step encoding). Let $\mathcal{P} = (I, G, A)$ be a solvable temporal planning problem, $\{e_1, ..., e_n\}$ be the set of all events of $\mathcal{P}$, and $\pi = \langle Step_1, ..., Step_l \rangle$ be a causally valid $\forall$-step plan for $\mathcal{P}$. There exists a model M for $\phi_l^{\forall}$ such that $\pi = plan(M)$.

Proof. See Appendix A. □

Theorem 4 (soundness of $\forall$-step encoding). Let $\mathcal{P} = (I, G, A)$ be a temporal planning problem, $\{e_1, ..., e_n\}$ be the set of all events of $\mathcal{P}$, and $\phi_l^{\forall}$ be the $\forall$-step encoding for $\mathcal{P}$. If $\phi_l^{\forall}$ has a model M, then $plan(M)$ is a causally valid $\forall$-step plan for $\mathcal{P}$.

Proof. See Appendix A. □

4.2.2 The $\exists$-step Encoding

In this part, we present the SAT-formula $\phi_l^{\exists}$, which is based on the $\exists$-step semantics of causally valid plans. By considering the two restrictive rules stated above, our $\exists$-step encoding is very similar to the $\forall$-step encoding described previously in this section. However, there are two major differences between these two kinds of encoding. First, our $\exists$-step encoding allows each proposition to be both required and deleted in each step, provided that the deleting event does not precede the requiring event in the fixed ordering $\langle e_1, ..., s_n \rangle$. This is in contrast with our $\forall$-step encoding, where a proposition could not be both deleted and required in the same step of the final plan. Second, in our $\exists$-step encoding, each step may also contain both the starting and ending event of the same action. Given the temporal problem $\mathcal{P} = (I, G, A)$, we produce the SAT-formula $\phi_l^{\exists}$, which is based on the $\exists$-step semantics of causally valid plans, for $\mathcal{P}$ by the conjunction of all formulae described below.

($\exists$-1) $\bigwedge\{p^0 | p \in state(I)\} \cup \{\neg p^0 | p \notin state(I)\}$

($\exists$-2) $\bigwedge\{p^l | p \in G\}$

($\exists$-3) $\bigwedge\{\neg a^0 | a \in A\}$

($\exists$-4) $\bigwedge\{\neg a^l | a \in A\}$

($\exists$-5) $\bigwedge\{e^k \rightarrow p^{k-1} | 0 < k \leq l, p \in P, e \in R_p\}$

($\exists$-6) $\bigwedge\{e^k \rightarrow p^k | 0 < k \leq l, p \in P, e \in E_p^+\}$

($\exists$-7) $\bigwedge\{e^k \rightarrow \neg p^k | 0 < k \leq l, p \in P, e \in E_p^-\}$

($\exists$-8) $\bigwedge\{\neg p^{k-1} \wedge p^k \rightarrow \bigvee_{e \in E_p^+} e^k | 0 < k \leq l, p \in P\}$

($\exists$-9) $\bigwedge\{p^{k-1} \wedge \neg p^k \rightarrow \bigvee_{e \in E_p^-} e^k | 0 < k \leq l, p \in P\}$

($\exists$-10) $\bigwedge\{chain(e_1, ..., e_{n+1}; E_p^-; R_p \cup \{e_{n+1}\}; k; m_1^p) | 0 < k \leq l, p \in P\} \cup \{(b_{n+1, m_1^p}^k \rightarrow \neg a^k) | 0 < k \leq l, p \in inv(a)\}$

(∃-11) $\bigwedge\{e_i^k \rightarrow \neg a^{k-1} | 0 < k \leq l, a \in A, e_i = start(a), e_j = end(a), i < j\}$

(∃-12) $\bigwedge\{e_i^k \rightarrow a^k \vee e_j^k | 0 < k \leq l, a \in A, e_i = start(a), e_j = end(a), i < j\}$

(∃-13) $\bigwedge\{e_j^k \rightarrow \neg a^k | 0 < k \leq l, a \in A, e_i = start(a), e_j = end(a), i < j\}$

(∃-14) $\bigwedge\{e_j^k \rightarrow a^{k-1} \vee e_i^k | 0 < k \leq l, a \in A, e_i = start(a), e_j = end(a), i < j\}$

(∃-15) $\bigwedge\{e_i^k \rightarrow \neg a^{k-1} \vee e_j^k | 0 < k \leq l, a \in A, e_i = start(a), e_j = end(a), j < i\}$

(∃-16) $\bigwedge\{e_i^k \rightarrow a^k | 0 < k \leq l, a \in A, e_i = start(a), e_j = end(a), j < i\}$

(∃-17) $\bigwedge\{e_j^k \rightarrow \neg a^k \vee e_i^k | 0 < k \leq l, a \in A, e_i = start(a), e_j = end(a), j < i\}$

(∃-18) $\bigwedge\{e_j^k \rightarrow a^{k-1} | 0 < k \leq l, a \in A, e_i = start(a), e_j = end(a), j < i\}$

(∃-19) $\bigwedge\{\neg a^{k-1} \wedge a^k \rightarrow e_i^k | 0 < k \leq l, a \in A, e_i = start(a)\}$

(∃-20) $\bigwedge\{a^{k-1} \wedge \neg a^k \rightarrow e_j^k | 0 < k \leq l, a \in A, e_j = end(a)\}$

Note that formulae (∃-1) to (∃-9) are exactly the same as formulae (∀-1) to (∀-9). Similar to our ∀-step encoding, these formulae are responsible for the validity of the initial state, goal state, conditions and effects of events, and also for the explanatory frame axioms explained before. Moreover, notice that while formula (∀-10) is also present in our ∃-step encoding as formula (∃-10), formula (∀-11) is not present in $\phi_l^\exists$. This results in the first major difference stated above between our ∃-step encoding and ∀-step encoding. Formulae (∃-11)to (∃-20) enforce appropriate changes to $agenda(s_{k-1})$ and $agenda(s_k)$, which are the agendas of the states immediately before and after step k of the final plan. According to their definitions, formulae (∃-11) to (∃-14) are added for each action a with the property that $start(a)$ is located before $end(a)$ in the fixed ordering $\langle e_1, ..., e_n \rangle$. Formula (∃-11) ensures that a can be started in step k, only if it is not open in s_{k-1}. Formula (∃-12) guarantees that if a is started but not ended in step k, it must be open in s_k. Formula (∃-13) ensures that if a is ended in step k, it will not be open in s_k. Formula (∃-14) implies that if a is ended but not started in step k, then it must be open in s_{k-1}. Analogously, formulae (∃-15) to (∃-18) guarantee similar properties for each action a with the property that $start(a)$ is located after $end(a)$ in the fixed ordering $\langle e_1, ..., e_n \rangle$. Formula (∃-19) ensures that if a is a member of $agenda(s_k)$ but not $agenda(s_{k-1})$, it must be started in step k. Similarly, formula (∃-20) ensures that if a is a member of $agenda(s_{k-1})$ but not $agenda(s_k)$, it must be ended in step k.

Since our ∃-step encoding conforms to the two restrictive rules stated earlier in this section, there may exist a ∃-step causally valid plan with l steps for a given problem while $\phi_l^\exists$ would be unsatisfiable for the same problem. This is also the case in the linear size ∃-step encoding of the classical planning problems (Rintanen et al., 2006). However, since we showed by Theorem 3 that our ∀-step encoding is complete, the completeness of our ∃-step encoding can be proved by showing that the satisfiability of $\phi_l^\forall$ entails the satisfiability of $\phi_l^\exists$.

Theorem 5 (completeness of ∃-step encoding). Let $\mathcal{P} = (I, G, A)$ be a solvable temporal planning problem, $\{e_1, ..., e_n\}$ be the set of all events of $\mathcal{P}$, and $\pi = \langle Step_1, ..., Step_l \rangle$ be a causally valid ∀-step plan for $\mathcal{P}$. There exists a model M for $\phi_l^\exists$ such that $\pi = plan(M)$.

Proof. See Appendix A. □

Theorem 5 also shows that in our ∃-step encoding, the number of required steps to solve a temporal planning problem is less than (or equal to) what is required by our ∀-step encoding. In other words, our ∃-step encoding is more compact than its ∀-step counterpart.

Theorem 6 (soundness of ∃-step encoding). Let $\mathcal{P} = (I, G, A)$ be a temporal planning problem, $\{e_1, ..., e_n\}$ be the set of all events of $\mathcal{P}$, and $\phi_l^{\exists}$ be the ∃-step encoding for $\mathcal{P}$. If $\phi_l^{\exists}$ has a model M, then $plan(M)$ is a causally valid ∃-step plan $\mathcal{P}$.

Proof. See Appendix A. □

4.3 The Relaxed ∃-step Encoding

As we mentioned in Section 4.2.2, our ∃-step encoding allows each proposition to be both required and deleted by any two events of the same step, only if the deleting event does not precede the requiring event in the fixed ordering $\langle e_1, ..., e_n \rangle$. Besides, since formulae (∃-5) and (∃-6) are present in both of our ∀-step and ∃-step encodings, no proposition can be both added and deleted in the same step of these encodings. These restrictions, which are also present in the classical ∀-step and ∃-step encodings (Rintanen et al., 2006), are lifted in our new relaxed version of ∃-step encoding. As a result, each proposition can be required, added, and deleted in any step as many times as it is needed. This property has not been previously examined in classical ∃-step encoding, and consequently, the chaining mechanism explained in Section 4.1 is not compatible with it. Here, we introduce a generalized version of the chains and explain the conceptual difference with those used in classical encodings. We also present new kinds of chains to be used specially in temporal planning for preserving the invariants of temporal actions while the plan is being produced. Note that, similar to our non-relaxed ∃-step encoding, here we assume that the events of each step are executed according to a fixed ordering $\langle e_1, ..., e_n \rangle$.

Let k be a natural number and $e_1, ..., e_n$ be a fixed ordering of all events. For some reasons to be discussed later, we assume that if e_i is the starting event of an action, then e_{i+1} is the ending event of that action. In other words, we assume that the ending event of each action is located immediately after its starting event in the fixed ordering. Note that here, we do not demand the end of an action to immediately follow its start in the final plan. We only put this constraint on the fixed ordering. This cannot compromise the completeness of ITSAT: the SAT solver can still choose the start of an action a from step k, choose whatever actions are needed from steps k to $k + m$ for an arbitrary m, and then choose the end of a from step $k+m$. Moreover, suppose that there are two dummy events e_0 and e_{n+1}, which do not have any precondition, add-effect, or delete-effect. Let P be the set of all propositions of $\mathcal{P}$. For each proposition $p \in P$, let $E_p^- = \{e|p \in del(e)\}$, $E_p^+ = \{e|p \in add(e)\}$, $O_p = \{e|p \in inv(action(e))\} \cup \{e_0, e_{n+1}\}$, and $R_p = \{e|p \in pre(e)\} \cup \{e|p \in inv(action(e)) - add(e)\}$. We define $chain^*(e_0, ..., e_{n+1}; E_p^+; E_p^-; R_p; k; m_p^*)$ by the conjunction of formulae (C*-1) to (C*-8) stated below. Note that m_p^* is a symbol used for distinguishing the SAT varibales used in the formula $chain^*(e_0, ..., e_{n+1}; E_p^+; E_p^-; R_p; k; m_p^*)$ from other variables used in other formulae.

(C*-1) $\bigwedge\{e_i^k \rightarrow b_{j,m_p^*}^k | i < j, e_i \in E_p^+, e_j \in R_p \cup E_p^-, \{e_{i+1}, ..., e_{j-1}\} \cap (R_p \cup E_p^-) = \emptyset\}$

(C*-2) $\bigwedge\{e_i^k \rightarrow \neg b_{j,m_p^*}^k | i < j, e_i \in E_p^-, e_j \in R_p \cup E_p^+, \{e_{i+1}, ..., e_{j-1}\} \cap (R_p \cup E_p^+) = \emptyset\}$

(C*-3) $\bigwedge\{b_{i,m_p^*}^k \leftrightarrow b_{j,m_p^*}^k | i < j, e_i \in R_p - (E_p^+ \cup E_p^-), e_j \in R \cup E_p^+ \cup E_p^-, \{e_{i+1}, ..., e_{j-1}\} \cap (R_p \cup E_p^+ \cup E_p^-) = \emptyset\}$

(C*-4) $\bigwedge\{(b_{i,m_p^*}^k \wedge \neg e_i^k) \rightarrow b_{j,m_p^*}^k | i < j, \{e_i, e_j\} \subset R_p \cup E_p^+ \cup E_p^-, \{e_{i+1}, ..., e_{j-1}\} \cap (R_p \cup E_p^+ \cup E_p^-) = \emptyset\}$

(C*-5) $\bigwedge\{(\neg b_{i,m_p^*}^k \wedge \neg e_i^k) \rightarrow \neg b_{j,m_p^*}^k | i < j, \{e_i, e_j\} \subset R_p \cup E_p^+ \cup E_p^-, \{e_{i+1}, ..., e_{j-1}\} \cap (R_p \cup E_p^+ \cup E_p^-) = \emptyset\}$

(C*-6) $\bigwedge\{\neg b_{i,m_p^*}^k \rightarrow \neg e_i^k | e_i \in R_p\}$

(C*-7) $b_{0,m_p^*}^k \leftrightarrow p^{k-1}$

(C*-8) $b_{n+1,m_p^*}^k \leftrightarrow p^k$

In fact, $chain^*(e_0, ..., e_{n+1}; E_p^+; E_p^-; R_p; k; m_p^*)$ encodes a message passing method that is different from that of $chain(e_1, ..., e_n; E; R; k; m)$. In $chain^*(e_0, ..., e_{n+1}; E_p^+; E_p^-; R_p; k; m_p^*)$, the conveyed message is in fact the value of proposition p, and is therefore either $true$ or $false$. Similar to the message passing strategy of $chain(e_1, ..., e_n; E; R; k; m)$, the received message is transferred only in the forward direction of the fixed ordering $e_1, ..., e_n$. Each event in E_p^+, E_p^-, or R_p receives a message from its previous event in the fixed ordering. Every event may or may not change the value of the received message. In either cases, the message is then passed to the next event. The events in E_p^+ can only change the value of the received message to $true$, as these events have p in their add-effects. Similarly, the events in E_p^- can only change the value of the received message to $false$. The formulae (C*-1) and (C*-2) impose such changes on the value of a received message. If an event is not a member of E_p^+ or E_p^-, it neither adds nor deletes p, and thus, it will pass the received message without altering its value. This is enforced by (C*-3). (C*-4) and (C*-5) ensure that received messages are passed without being changed by those events that are not to be chosen for $Step_k$ of the output plan. According to (C*-6), if an event in R_p receives a message with the value of $false$, the event cannot be chosen as a member of $Step_k$. That is because members of R_p require p, which necessitates their received messages to have a value of $true$. (C*-7) implies that the initial value of the message produced in $Step_k$ is equal to the value of p in the state immediately before the execution of $Step_k$. Similarly, (C*-8) implies that the value of p in the state immediately after the execution of $Step_k$ will be equal to the final value of the message in $Step_k$.

Example 6. Consider the same events given in Example 5. Let E^+ be the set of events that add x (i.e., $E^+ = \{a_3\}$), E^- be the set of events that delete x (i.e., $E^- = \{a_2\}$), and R be the set of events that require x (i.e., $R = \{a_1, a_4\}$). Moreover, suppose that there are two dummy events e_0 and e_5, which do not have any precondition, add-effect, or delete-effect. Assume that we have added $chain^*(e_0, ..., e_5; E^+; E^-; R \cup \{e_0, e_5\}; k; m_x^*)$ to the SAT formula. According to formulae (C*-1) to (C*-8), this chain is the conjunction of the following formulae:

- $e_3^k \to b_{4,m_p^*}^k$
- $e_2^k \to \neg b_{3,m_p^*}^k$
- $b_{0,m_p^*}^k \leftrightarrow b_{1,m_p^*}^k$
- $b_{1,m_p^*}^k \leftrightarrow b_{2,m_p^*}^k$
- $b_{4,m_p^*}^k \leftrightarrow b_{5,m_p^*}^k$
- $b_{0,m_p^*}^k \wedge \neg e_0^k \to b_{1,m_p^*}^k$
- $b_{1,m_p^*}^k \wedge \neg e_1^k \to b_{2,m_p^*}^k$
- $b_{2,m_p^*}^k \wedge \neg e_2^k \to b_{3,m_p^*}^k$
- $b_{3,m_p^*}^k \wedge \neg e_3^k \to b_{4,m_p^*}^k$
- $b_{4,m_p^*}^k \wedge \neg e_4^k \to b_{5,m_p^*}^k$
- $\neg b_{0,m_p^*}^k \wedge \neg e_0^k \to \neg b_{1,m_p^*}^k$
- $\neg b_{1,m_p^*}^k \wedge \neg e_1^k \to \neg b_{2,m_p^*}^k$
- $\neg b_{2,m_p^*}^k \wedge \neg e_2^k \to \neg b_{3,m_p^*}^k$
- $\neg b_{3,m_p^*}^k \wedge \neg e_3^k \to \neg b_{4,m_p^*}^k$
- $\neg b_{4,m_p^*}^k \wedge \neg e_4^k \to \neg b_{5,m_p^*}^k$
- $\neg b_{0,m_p^*}^k \to \neg e_0^k$
- $\neg b_{1,m_p^*}^k \to \neg e_1^k$
- $\neg b_{4,m_p^*}^k \to \neg e_4^k$
- $\neg b_{5,m_p^*}^k \to \neg e_5^k$
- $b_{0,m_p^*}^k \leftrightarrow x^{k-1}$
- $b_{5,m_p^*}^k \leftrightarrow x^k$

A straightforward examination shows that we can have a model M for the chain mentioned above such that $M(e_0^k) = M(e_1^k) = M(e_2^k) = M(e_3^k) = M(e_4^k) = M(e_5^k) = true$, $M(b_{0,m_p^*}^k) = M(b_{1,m_p^*}^k) = M(b_{2,m_p^*}^k) = M(b_{4,m_p^*}^k) = M(b_{5,m_p^*}^k) = true$, and $M(b_{3,m_p^*}^k) = false$. In other words, if x is deleted by e_2 in $Step_k$ of the final plan, it can later be produced again by e_3, and as a result, e_4 can appear in $Step_k$, too. Here, $M(b_{3,m_p^*}^k) = false$ represents the fact that x has been deleted after the execution of e_2. In this example, all four propositions p_1, p_2, p_3, and p_4 can be produced by a single step of the final plan. Note that in neither of the cases of Example 5, producing all these propositions by only one step was possible. This is an example of how our new $\exists$-step encoding, which employs the generalized message passing strategy, can permit more parallelism than is allowed by the temporal versions of classical $\forall$-step and $\exists$-step encodings.

In addition to $chain^*(e_0, ..., e_{n+1}; E_p^+; E_p^-; R_p; k; m_p^*)$, which is responsible for tracking the value of p inside $Step_k$, we also need some extra formulae to prevent p from being deleted whenever p is an invariant of an open temporal action. Therefore, we introduce two new chain formulae: $chain^{of}(e_1, ..., e_{n+1}; E_p^-; O_p; k; m_p^{of})$ and $chain^{ob}(e_0, ..., e_n; E_p^-; O_p; k; m_p^{ob})$. Formula $chain^{of}(e_1, ..., e_{n+1}; E_p^-; O_p; k; m_p^{of})$ is produced by the conjunction of formulae (C^{of}-1) to (C^{of}-4). Similar to the chains explained before, m_p^{of} is a symbol used to distinguish the SAT varibales of this chain from those of other formulae.

(C^{of}-1) $\bigwedge\{e_i^k \rightarrow b_{j,m_p^{of}}^k | i < j, e_i \in E_p^-, e_j \in O_p, \{e_{i+1}, ..., e_{j-1}\} \cap O_p = \emptyset\}$

(C^{of}-2) $\bigwedge\{b_{i,m_p^{of}}^k \rightarrow b_{j,m_p^{of}}^k | i < j, \{e_i, e_j\} \subset O_p, \{e_{i+1}, ..., e_{j-1}\} \cap O_p = \emptyset\}$

(C^{of}-3) $\bigwedge\{(b_{j,m_p^{of}}^k \wedge e_j^k) \rightarrow e_i^k | e_i \in O_p, e_i = start(a), e_j = end(a)\}$

(C^{of}-4) $\bigwedge\{(b_{n+1,m_p^{of}}^k \wedge a^k) \rightarrow e_i^k | a \in A, e_i = start(a), e_i \in O_p\}$

Similar to $chain(e_1, ..., e_n; E_p^-; R_p; k; m_p)$, $chain^{of}(e_1, ..., e_n; E_p^-; O_p; k; m_p^{of})$ represents a message that is produced and sent in the forward direction of the fixed ordering, whenever a proposition p is deleted by an event. (C^{of}-1) and (C^{of}-2) are responsible for the production and propagation of the mentioned message, respectively. By (C^{of}-3), if the ending event of an action a with p as its invariant receives this message in step k, then step k must also include the starting event of a. In these cases, (C^{of}-3) prevents a from being open when p is deleted. That is because we assume that in the fixed ordering, the ending event of each action is located immediately after its starting event. (C^{of}-4) guarantees that if p is deleted somewhere in step k, and an action a with p as its invariant is open after step k, then step k must also include the starting event of a (otherwise, a is open everywhere in step k, and thus, p, which is an invariant of a, is deleted while a is open).

In $chain^{of}(e_1, ..., e_{n+1}; E_p^-; O_p; k; m_p^{of})$, the message that indicates p is deleted is only sent forward. Thus, it cannot help preserving the invariants of those members of O_p that are started prior to the deletion of p. To tackle this problem, we add another chain, namely $chain^{ob}(e_0, ..., e_n; E_p^-; O_p; k; m_p^{ob})$, to the formula. This chain is quite analogous to $chain^{of}(e_1, ..., e_{n+1}; E_p^-; O_p; k; m_p^{of})$, that whenever p is deleted by an event, the chain sends the message backward according to the fixed ordering. $chain^{ob}(e_0, ..., e_n; E_p^-; O_p; k; m_p^{ob})$ is produced by the conjunction of formulae (C^{ob}-1) to (C^{ob}-4).

(C^{ob}-1) $\bigwedge\{e_i^k \rightarrow b_{j,m_p^{ob}}^k | j < i, e_i \in E_p^-, e_j \in O_p, \{e_{j+1}, ..., e_{i-1}\} \cap O_p = \emptyset\}$

(C^{ob}-2) $\bigwedge\{b_{i,m_p^{ob}}^k \rightarrow b_{j,m_p^{ob}}^k | j < i, \{e_i, e_j\} \subset O_p, \{e_{j+1}, ..., e_{i-1}\} \cap O_p = \emptyset\}$

(C^{ob}-3) $\bigwedge\{(b_{i,m_p^{ob}}^k \wedge e_i^k) \rightarrow e_j^k | e_i \in O_p, e_i = start(a), e_j = end(a)\}$

(C^{ob}-4) $\bigwedge\{(b_{0,m_p^{ob}}^k \wedge a^{k-1}) \rightarrow e_j^k | a \in A, e_j = end(a), e_j \in O_p\}$

We now present the SAT-formula $\phi_l^{\exists^*}$, which represents our relaxed $\exists$-step encoding and is based on the $\exists$-step semantics of causally valid plans. $\phi_l^{\exists^*}$ is produced by the conjunction of all formulae described below.

($\exists^*$-1) $\bigwedge\{p^0 | p \in state(I)\} \cup \{\neg p^0 | p \notin state(I)\}$

($\exists^*$-2) $\bigwedge\{p^l | p \in G\}$

($\exists^*$-3) $\bigwedge\{\neg a^0 | a \in A\}$

($\exists^*$-4) $\bigwedge\{\neg a^l | a \in A\}$

($\exists^*$-5) $\bigwedge\{chain^*(e_0, e_1, ..., e_{n+1}; E_p^+; E_p^-; R_p \cup \{e_0, e_{n+1}\}; k; m_p^*) | 0 < k \leq l, p \in P\}$

($\exists^*$-6) $\bigwedge\{chain^{of}(e_1, ..., e_{n+1}; E_p^-; O_p; k; m_p^{of}) | 0 < k \leq l, p \in P\}$

($\exists^*$-7) $\bigwedge\{chain^{ob}(e_0, ..., e_n; E_p^-; O_p; k; m_p^{ob}) | 0 < k \leq l, p \in P\}$

($\exists^*$-8) $\bigwedge\{e_i^k \rightarrow \neg a^{k-1} | 0 < k \leq l, a \in A, e_i = start(a)\}$

($\exists^*$-9) $\bigwedge\{e_i^k \rightarrow a^k \vee e_j^k | 0 < k \leq l, a \in A, e_i = start(a), e_j = end(a)\}$

($\exists^*$-10) $\bigwedge\{e_j^k \rightarrow \neg a^k | 0 < k \leq l, a \in A, e_j = end(a)\}$

($\exists^*$-11) $\bigwedge\{e_j^k \rightarrow a^{k-1} \vee e_i^k | 0 < k \leq l, a \in A, e_i = start(a), e_j = end(a)\}$

($\exists^*$-12) $\bigwedge\{\neg a^{k-1} \wedge a^k \rightarrow e_i^k | 0 < k \leq l, a \in A, e_i = start(a)\}$

($\exists^*$-13) $\bigwedge\{a^{k-1} \wedge \neg a^k \rightarrow e_j^k | 0 < k \leq l, a \in A, e_j = end(a)\}$

($\exists^*$-1) ensures that any member of $state(s_0)$ is true if and only if that member is present in the initial state. Similarly, ($\exists^*$-2) guarantees that all members of the goal state are true in $state(s_l)$. ($\exists^*$-3) and ($\exists^*$-4) imply that $agenda(s_0)$ and $agenda(s_l)$ are both empty. ($\exists^*$-5), as explained before, is responsible for imposing appropriate changes in the value of the SAT variables, whenever proposition p is added or deleted by an event in a certain step of the output plan. ($\exists^*$-6) and ($\exists^*$-7) prevent the invariants of an action a from being deleted while a is open. ($\exists^*$-8) to ($\exists^*$-13) are responsible for enforcing the appropriate changes to $agenda(s_{k-1})$ and $agenda(s_k)$, which are agendas of the states immediately before and after step k of the output plan. ($\exists^*$-8) ensures that a can be started in step k only if it is not open in s_{k-1}. ($\exists^*$-9) indicates that if a is started but not ended in step k, then it has to be open in s_k. ($\exists^*$-10) ensures that if a is ended in step k, it will not be open in s_k. ($\exists^*$-11) implies that if a is ended but not started in step k, then it has to be open in s_{k-1}. ($\exists^*$-12) ensures that if a is a member of $agenda(s_k)$ but not a member of $agenda(s_{k-1})$, then a is started in step k. Similarly, ($\exists^*$-13) ensures that if a is a member of $agenda(s_{k-1})$ but not $agenda(s_k)$, then a is ended in step k.

By Theorem 5, we know that if a temporal planning problem $\mathcal{P}$ is satisfiable, then there exists a positive number l, such that the non-relaxed $\exists$-step encoding of $\mathcal{P}$ with l steps (i.e., $\phi_l^{\exists}$) is satisfiable. Accordingly, the completeness of our relaxed $\exists$-step encoding can be proved by showing that if $\phi_l^{\exists}$ is satisfiable then $\phi_l^{\exists^*}$ will also be satisfiable.

Theorem 7 (completeness of the relaxed $\exists$-step encoding). Let $\mathcal{P} = (I, G, A)$ be a temporal planning problem and formulae $\phi_l^{\exists}$ and $\phi_l^{\exists^*}$ be two $\exists$-step encodings of $\mathcal{P}$ explained above. If M is a model for $\phi_l^{\exists}$, then $\phi_l^{\exists^*}$ has a model M' such that $plan(M') = plan(M)$.

Proof. See Appendix A. □

Theorem 7 also shows that, in our $\exists^*$-step encoding, the number of required steps to solve a temporal planning problem is less than (or equal to) what is required by our $\exists$-step encoding, provided that the same fixed ordering has been used in these two encodings. In other words, our $\exists^*$-step encoding is more compact than our $\exists$-step encoding.

Theorem 8 (soundness of the relaxed $\exists$-step encoding). Let $\mathcal{P} = (I, G, A)$ be a temporal planning problem, $\{e_1, ..., e_n\}$ be the set of all events of $\mathcal{P}$, and $\phi_l^{\exists^*}$ be the relaxed $\exists$-step encoding for $\mathcal{P}$. If $\phi_l^{\exists^*}$ has a model M, then $plan(M)$ is a causally valid $\exists$-step plan $\mathcal{P}$.

Proof. See Appendix A. □

4.4 Mutual Exclusion Relations and Action Compression

As we mentioned earlier in Section 3, the performance of any SAT-based temporal planner can be improved by mutual exclusion analysis and action compression. In this section, we show how the information obtained by these tasks has been utilized in ITSAT. Let $\mathcal{P} = (I, G, A)$ be a temporal planning problem, $\mathcal{MUT}$ be the set of all mutually exclusive pairs of propositions of $\mathcal{P}$, and $\mathcal{COM}$ be the set of all compression-safe actions of $\mathcal{P}$ (see Section 3). Let ϕ_l be the encoding of $\mathcal{P}$, which can be any of $\phi_l^{\forall}$, $\phi_l^{\exists}$, or $\phi_l^{\exists^*}$. For taking advantage of mutual exclusion relations, we add an extra formula ϕ_l^{mut} to ϕ_l, where $\phi_l^{mut} = \bigwedge\{\neg p^k \vee \neg q^k | (p, q) \in \mathcal{MUT}, 1 \leq k \leq l\}$. As Theorem 1 shows, two mutually exclusive propositions p and q can never be both true in any state that is achieved by the execution of a valid temporal plan starting from I. As a result, adding ϕ_l^{mut} to the encoding cannot render the planner incapable of finding valid plans.

Let $a \in A$ be a compression-safe action. As it was showed in Section 3.2, it is safe to assume that in any causally valid plan, the ending event of a occurs immediately after its starting event. One way to impose such constraint is to add some extra clauses to the encoding to guarantee that both starting and ending events of a are always included in the same step. However, these two events may have conflicting effects, in which case $\phi_l^{\forall}$ and $\phi_l^{\exists}$ will not allow such events to be both present at any step. Therefore, the information regarding compression-safe actions is only added to our relexed $\exists$-step encoding, $\phi_l^{\exists^*}$. This is done by adding ϕ_l^{com} to $\phi_l^{\exists^*}$, where $\phi_l^{com} = \bigwedge\{e_i^k \leftrightarrow e_{i+1}^k | a \in \mathcal{COM}, e_i = start(a), 1 \leq k \leq l\}$. Note that in $\phi_l^{\exists^*}$, if e_i is the starting event of an action, e_{i+1} denotes its ending event.

5. Scheduling Phase

In this section, we describe how a causally valid plan is augmented by temporal information to produce a valid temporal plan. Let $\pi = \langle e_1, ..., e_n \rangle$ be a causally valid plan produced by our planner. The scheduling of π is done by defining the scheduling function τ_π of Definition 9, which assigns a rational number to each event of π as its execution time. Suppose that we have given different names to different occurrences of the same action in this plan, so all the events $e_1, ..., e_n$ are unique. We can assume that for each i, $\tau_\pi(i) < \tau_\pi(i+1)$, and thereby satisfy the first condition of Definition 9. However, this can lead to the plans with unnecessarily large makespans. Alternatively, for obtaining plans with improved quality, we impose a more relaxed set of constraints on the function τ.

Definition 16 (relaxed scheduling functions). Let π be a causally valid plan. The scheduling function τ_π is a relaxed scheduling function if it has the following properties:

(S-1) For all i and j such that e_i is located before e_j in π, and e_i is not swappable with e_j (cf. Definition 12), we require that $\tau_\pi(i) < \tau_\pi(j)$.

(S-2) For all i and j, if e_i is the starting event of a particular action a, and e_j is the pairing event of e_i in π (cf. Definition 8), we require that $\tau_\pi(j) = \tau_\pi(i) + dur(a)$.

Theorem 9. Let $\mathcal{P} = (I, G, A)$ be a temporal planning problem, $\pi = \langle e_1, ..., e_n \rangle$ be a causally valid plan for $\mathcal{P}$, and $\tau_\pi : \{1, ..., n\} \to \mathbb{Q}$ be a relaxed scheduling function for π. There exists a valid temporal plan for $\mathcal{P}$.

Proof. See Appendix A. □

Theorem 9 shows that whenever a relaxed scheduling function exists for a causally valid plan of $\mathcal{P}$, a valid temporal plan can be produced for $\mathcal{P}$. To prove that our scheduling method does not render ITSAT incomplete, we also need to show that if $\mathcal{P}$ is solvable, our planner will be able to produce a causally valid plan π and a scheduling function τ_π such that τ_π is a relaxed scheduling function for π. Let (π, τ_π) be a valid temporal plan for $\mathcal{P}$. Every causally valid plan can also be regarded as a causally valid $\forall$-step plan with singleton steps. Therefore, by Theorem 3, $\phi_l^{\forall}$ has a model M such that $\pi = plan(M)$. By Theorem 5, M also satisfies $\phi_l^{\exists}$. Moreover, by Theorem 7, $\phi_l^{\exists^*}$ has a model M' such that $\pi = plan(M') = plan(M)$. Therefore, if any of our encoding methods are used for translating $\mathcal{P}$ to a SAT formula, the resulting formula has a model that will be translated to π. On the other hand, according to Definition 9, τ_π satisfies all constraints of the form (S-1) and (S-2), and therefore, is a relaxed scheduling function of π. However, as mentioned in Section 4.2.4, we may add certain clauses to the encoding to ensure that the produced causally valid plan is always compressed (Definition 11). We now show that for each solvable temporal plan, there exists a compressed causally valid plan that can be scheduled to a valid temporal plan by a relaxed scheduling function.

Theorem 10. Let $\mathcal{P} = (I, G, A)$ be a solvable temporal planning problem, and $\mathcal{COM}$ be the set of every member of A that is either compressible towards its start or compressible towards its end (Definition 13). There exists a valid temporal plan (π, τ_π) for $\mathcal{P}$ such that π is a causally valid plan for $\mathcal{P}$, π is compressed with respect to $\mathcal{COM}$, and τ_π is a relaxed scheduling function for π.

Proof. See Appendix A. □

To check the existence of the function τ_π with the properties stated above, we solve an instance of a Simple Temporal Problem (STP) (Dechter et al., 1991). Each STP is associated with a weighted graph named a Simple Temporal Network (STN). We construct an STN in which node x_i corresponds to the event e_i in the causally valid plan π. Let ϵ be an arbitrary small rational number. For each constraint of the form $\tau_\pi(i) < \tau_\pi(j)$, we add an edge with the weight $-\epsilon$ from x_i to x_j. For each constraint of the form $\tau_\pi(j) = \tau_\pi(i) + dur(a)$, we add an edge with the weight -$dur(a)$ from x_i to x_j, and another edge with weight $dur(a)$ from x_j to x_i. We also add a reference node x_0 to the constructed STN. x_0 has an edge with the weight 0 to every other node. A solution of any STP can be found by computing the length of the shortest path form x_0 to all other nodes (Dechter et al., 1991). Suppose that such shortest paths exist and the length of the shortest from x_0 to x_i is shown by $distance(x_0, x_i)$. For each event e_i, we define $\tau_\pi(i)$ to be equal to $-distance(x_0, x_i)$. In this case, Theorem 9 guarantees that the resulting plan has all the specifications of a valid temporal plan.

To see the intuition behind the explained method of defining the function τ_π, suppose that in the constructed STN, there is an edge with the weight $-\epsilon$ from x_i to x_j. This means that $distance(x_0, x_j) \leq distance(x_0, x_i) - \epsilon$, which implies $-distance(x_0, x_i) \leq -distance(x_0, x_j) - \epsilon$. This, in turn, implies that $-distance(x_0, x_i) < -distance(x_0, x_j)$, and $\tau_\pi(i) < \tau_\pi(j)$. Similarly, it can be easily shown that if there exists an edge with the weight -$dur(a)$ from x_i to x_j, and another edge with weight $dur(a)$ from x_j to x_i, then we will have: $\tau_\pi(j) = \tau_\pi(i) + dur(a)$. Bellman-Ford algorithm (Cormen, Leiserson, Rivest, & Stein, 2009) can be used to find all single source shortest paths of any weighted graph in polynomial time. Besides, the number of the nodes of the produced STN is equal to the number of events of the causally valid plan. Therefore, we can conclude that, if the mentioned shortest paths exist, $\tau_\pi(i)$ can be computed in polynomial time.

However, there are situations in which such shortest paths do not exist. It happens when the corresponding STN has a negative cycle. In these situations, the STP is inconsistent and consequently, all the temporal constraints cannot be satisfied at the same time. An example of such a case is depicted in Figure 10.

In Figure 10, action a adds propositions p and g by its starting and ending events, respectively. a needs proposition q as a precondition for its ending event. Action b requires p upon starting and adds q upon ending. Durations of actions a and b, are 5 and 10, respectively. The goal of planning is reaching fact g. For this problem, $\pi = \langle a^s, b^s, b^e, a^e \rangle$

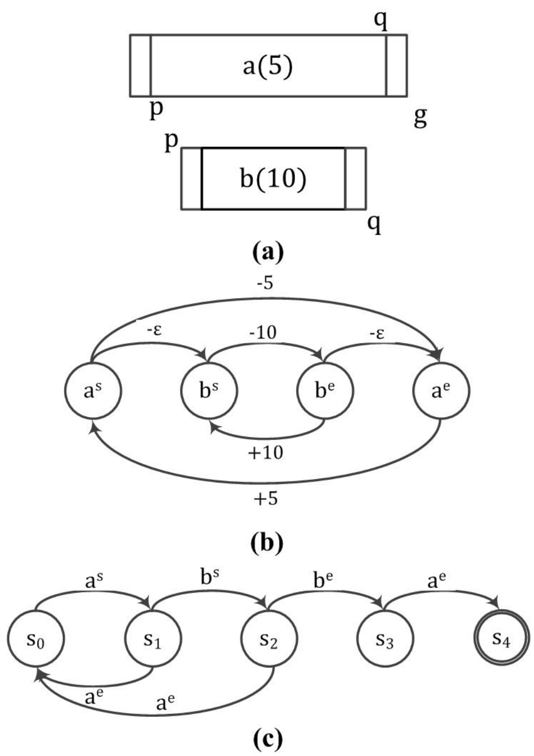

Figure 10. Negative Cycles

is a causally valid plan, where $a^s = start(a)$, $a^e = end(a)$, $b^s = start(b)$, $b^e = end(b)$. This plan is depicted in Figure 10-(a). In this plan, the execution of action b must be entirely inside that of action a (i.e., b is started after starting a and is ended before ending a). However, this is impossible considering the fact that the duration of a is less than that of b. The invalidity of this plan is caused by the fact that while producing this causally valid plan, all the durations are abstracted out. The STN constructed for the plan of Figure 10-(a) is depicted in Figure 10-(b). $a^s b^s b^e a^e a^s$ is a negative cycle with the total weight of $-5 - 2\epsilon$.

5.1 Negative Cycle Prevention

If the STN of a causally valid plan includes a negative cycle, the plan cannot be transformed into a valid temporal plan. In such cases, the SAT solver will be forced to find a different solution. This can be done by adding an extra clause such that at least one of the events of the current negative cycle will be prevented from occurring in its current step. However, after adding such a blocking clause, the planner can still produce new plans that are basically equivalent to the previous plan. For instance, consider the example given in Figure 10. Suppose a^s, b^s, b^e, and a^e are members of steps 1 to 4, respectively. Assume that the output plan is to have 5 steps. If we forbid the exact occurrence of the negative cycle, a new causally valid plan can still be produced by shifting a^e to layer 5 and maintaining all other events in their current steps. The new solution will have the same negative cycle and therefore cannot be transformed to a valid temporal plan. In fact, here the cause of the invalidity of the plan has not changed. We now show how by exploiting the simple structure of negative cycles, one can prevent the reoccurrence of such cycles more effectively.

From the discussion given above, it should be clear that the main reason that such negative cycles are encountered in the STN of a particular causally valid plan, is some specific order of events in that plan. In fact, if the events of any negative cycle reoccur with the same order in a new causally valid plan, the new plan will include the same negative cycle, too.

For any temporal planning problem $\mathcal{P}$, we can regard the set of all possible sequences of events as a language over the alphabet $\sum = \{e_1, ..., e_n\}$, where n is the number of all events of $\mathcal{P}$. The set of all sequences of events in which certain events have appeared in a particular order can also be regarded as another language over $\sum$. It is straightforward to show that this latter language is in fact a *regular* language and can be accepted by a Finite State Machine (FSM). Figure 10-(c) shows a Finite State Machine that detects the sequences of events in which a^s, a^e, b^s, and b^e appear according to the order $\langle a^s, b^s, b^e, a^e \rangle$. Note that, for the sake of clarity, the self-loop transitions of the Finite State Machine are not shown in Figure 10-(c).

Definition 17 (FSMs). A Finite State Machine ψ is a 5-tuple $(S_\psi, \sum, T_\psi, x_0, A_\psi)$, where S_ψ is a finite set of states, $\sum$ is a finite set of alphabet symbols, $T_\psi : S_\psi \times \sum \to S_\psi$ is a mapping defining all transitions of ψ, $x_0 \in S_\psi$ is the starting state, and $A_\psi \subseteq S_\psi$ is a set of accepting states.

We now show how by adding certain formulae to our SAT encodings, one can avoid the members of a given regular language from being produced as causally valid plans. Let $\mathcal{P}$ be a temporal problem, and $\sum = \{e_1, ..., e_n\}$ be the set of all events of $\mathcal{P}$. Let L be any regular langaue over $\sum$. Assume that L is accepted by the Finite State Machine $\psi = (S_\psi, \sum, T_\psi, A_\psi)$. For each $x_i \in S_\psi$, let $E_i^{out} = \{e \in \sum | T_\psi(x_i, e) = x_j, i \neq j\}$ and $E_i^{in} = \{e \in \sum | T_\psi(x_j, e) = x_i, i \neq j\}$. Assume that there are two dummy events e_0 and e_{n+1}, which do not have any precondition, add effect, or delete effect. We define the SAT variable $x^{k,i}$ for $1 \leq k \leq l$, $0 \leq i \leq n+1$, and $x \in S_\psi$. Assigning a value of *true* to $x^{k,i}$ means that ψ will be in state x, after operating on the sequence of all events of the steps 1 to $k-1$ and the events of step k with the indices less than i of the final plan. We construct the formula ϕ_l^ψ by the conjunction of formulae (ψ-1) to (ψ-6) stated below:

(ψ-1) $\bigwedge\{e_i^k \wedge x_s^{k,i} \to x_t^{k,j} | 1 \leq k \leq l, i < j, T_\psi(x_s, e_i) = x_t, e_j \in E_t^{out} \cup E_t^{in} \cup \{e_{n+1}\},$
$\{e_{i+1}, ..., e_{j-1}\} \cap (E_t^{out} \cup E_t^{in}) = \emptyset\}$

(ψ-2) $\bigwedge\{\neg e_i^k \wedge x_s^{k,i} \to x_s^{k,j} | 1 \leq k \leq l, i < j, e_i \in E_s^{out}, e_j \in E_s^{out} \cup E_s^{in} \cup \{e_{n+1}\},$
$\{e_{i+1}, ..., e_{j-1}\} \cap (E_s^{out} \cup E_s^{in}) = \emptyset\}$

(ψ-3) $\bigwedge\{x_s^{k,0} \to x_s^{k,i} | 1 \leq k \leq l, 1 \leq i \leq n, x_s \in S_\psi, e_i \in E_s^{out} \cup E_s^{in},$
$\{e_1, ..., e_{i-1}\} \cap (E_s^{out} \cup E_s^{in}) = \emptyset\}$

(ψ-4) $\bigwedge\{x_s^{k,n+1} \leftrightarrow x_s^{k+1,0} | 1 \leq k < l, x_s \in S_\psi\}$

(ψ-5) $\bigwedge\{x_0^{k,i} | 1 \leq k \leq l, 1 \leq i \leq n\}$

(ψ-6) $\bigwedge\{\neg x^{k,i} | x \in A_\psi, 1 \leq k \leq l, 1 \leq i \leq n+1\}$

Adding ϕ_l^ψ to the encoding of a problem makes the SAT solver to somehow simulate the behavior of ψ, while finding a model that represents a causally valid plan. Assume that observing e_i causes ψ to make a transition from x_s to x_t. Moreover, let e_j be the first event after e_i (according to the fixed ordering $e_1, ..., e_n$) that may cause a transition to or from x_t. Formula (ψ-1) guarantees that if e_i is a member of $Step_k$, and ψ is at state x_s at the time of observing e_i, then the state of ψ will be changed to x_t, and the next relevant event of x_t (i.e., e_j) will become aware of this transition. (ψ-2) implies that if e_i is not a member of $Step_k$, and ψ is at state x_s at the time of observing e_i, then ψ will remain at x_s, and the next relevant event of x_s will become aware of the current state of ψ. (ψ-3) causes the information regarding the state of ψ at the start of each step to be propagated to the first relevant event of that step. (ψ-4) propagates the information regarding the state of ψ at the end of each step to the next step. (ψ-5) ensures that ψ can be at its starting state at any place of the final plan. This means that the simulation of ψ can be started from anywhere in the plan that is being produced. This enables the SAT solver to detect not only the strings accepted by ψ, but also strings that have some subsequences accepted by ψ. Finally, (ψ-6) guarantees that ψ will never be at one of its accepting states.

Example 7. Let ψ be the Finite State Machine depicted in Figure 10-(c). This Finite State Machine detects the sequences of events in which a^s, a^e, b^s, and b^e appear according to the order $\langle a^s, b^s, b^e, a^e \rangle$. Assume that we have four events: $e_1 = a^s$, $e_2 = a^e$, $e_3 = b^s$, and $e_4 = b^e$. Also assume that there are two dummy events e_0 and e_5. For the sake of simplicity, suppose that the problem does not have any events other than e_0 to e_5, and the encoding has only two steps. Consider a boolean assignment M, such that $M(e_1^1) = M(e_3^1) = M(e_4^1) = M(e_2^2) = true$, and $M(e_2^1) = false$. In other words, M is choosing a^s, b^s, and b^e from the first step, and a^e from the second step. In fact, we have $plan(M) = \langle a^s, b^s, b^e, a^e \rangle$. In this example, we have $E_0^{in} = \{e_2\}$, because $e_2 = a^e$ is the only event that causes a transition to state s_0 of ψ. Similarly, we have: $E_0^{out} = \{e_1\}$, $E_1^{in} = \{e_1\}$, $E_1^{out} = \{e_2, e_3\}$, $E_2^{in} = \{e_3\}$, $E_2^{out} = \{e_2, e_4\}$, $E_3^{in} = \{e_4\}$, $E_3^{out} = \{e_2\}$, $E_4^{in} = \{e_2\}$, and $E_4^{out} = \emptyset$. We show that if we use the formulae (ψ-1) to (ψ-6) stated above to encode ψ, then M cannot be a model for the produced SAT formula. We show this by contradiction. Assume that M is a model for the produced SAT formula.

- s_0 is the starting state of ψ. Hence, according to (ψ-5), we have $M(s_0^{1,1}) = true$, which means that ψ is in state s_0, prior to checking whether e_1 is present in the first step of the final plan.

- According to (ψ-1), we have $e_1^1 \wedge s_0^{1,1} \rightarrow s_1^{1,2}$. Since we have $M(e_1^1) = true$ and $M(s_0^{1,1}) = true$, we must also have $M(s_1^{1,2}) = true$. In other words, ψ verifies that e_1 is present in the first step of the final plan, which causes the current state of ψ to be changed from s_0 to s_1. $M(s_1^{1,2}) = true$ implies that ψ is in state s_1, prior to checking whether e_2 is present in the first step of the final plan.

- According to (ψ-2), we have $\neg e_2^1 \wedge s_1^{1,2} \rightarrow s_1^{1,3}$. Since we have $M(e_2^1) = false$ and $M(s_1^{1,2}) = true$, we must also have $M(s_1^{1,3}) = true$. In other words, ψ verifies that e_2 is not present in the first step of the final plan, which causes the state of ψ to remain s_1. $M(s_1^{1,3}) = true$ implies that ψ is in state s_1, prior to checking whether e_3 is present in the first step of the final plan.

- According to (ψ-1), we have $e_3^1 \wedge s_1^{1,3} \rightarrow s_2^{1,4}$. Since we have $M(e_3^1) = true$ and $M(s_1^{1,3}) = true$, we must have $M(s_2^{1,4}) = true$. In other words, ψ verifies that e_3 is present in the first step of the final plan, which causes the state of ψ to be changed from s_1 to s_2. $M(s_2^{1,4}) = true$ implies that ψ is in state s_2, prior to checking whether e_4 is present in the first step of the final plan.

- According to (ψ-1), we have $e_4^1 \wedge s_2^{1,4} \rightarrow s_3^{1,5}$. Since we have $M(e_4^1) = true$ and $M(s_2^{1,4}) = true$, we must have $M(s_3^{1,5}) = true$. In other words, ψ finds out that e_4 is present in the first step of the final plan, which causes the state of ψ to be changed from s_2 to s_3. $M(s_3^{1,5}) = true$ implies that ψ is in state s_3, after visiting all events of the first step of the final plan.

- According to (ψ-4), we have $s_3^{1,5} \leftrightarrow s_3^{2,0}$. Since we have $M(s_3^{1,5}) = true$, we must also have $M(s_3^{2,0}) = true$, which implies that ψ is in state s_3, prior to visiting any event of the second step of the final plan.

- According to (ψ-3), we have $s_3^{2,0} \leftrightarrow s_3^{2,2}$. Since we have $M(s_3^{2,0}) = true$, we must also have $M(s_3^{2,2}) = true$, which implies that ψ is in state s_3, prior to checking whether e_2 is present in the second step of the final plan.

- According to (ψ-1), we have $e_2^2 \wedge s_3^{2,2} \rightarrow s_4^{2,5}$. Since we have $M(e_2^2) = true$ and $M(s_3^{2,2}) = true$, we must also have $M(s_4^{2,5}) = true$. In other words, ψ verifies that e_2 is present in the second step of the final plan, which causes the state of ψ to be changed from s_3 to s_4. $M(s_4^{2,5}) = true$ implies that ψ is in state s_4, after visiting all events of the first two steps of the final plan. On the other hand, s_4 is an accepting state of ψ. Hence, according to (ψ-6), we have $M(s_4^{2,5}) = false$. This is clearly a contradiction. Therefore, we can conclude that M cannot be a model for the produced SAT formula.

We now prove that adding ϕ_l^{ψ} to the encoding of the given problem, prevents the planner from producing those causally valid plans with a subsequence of events that is equivalent to any string accepted by ψ. This means that once a negative cycle has been translated into an FSM, the reoccurrence of the negative cycle can be avoided by translating that FSM into a SAT formula, and adding the formula to the encoding of the problem.

Theorem 11. Let $\mathcal{P} = (I, G, A)$ be a temporal planning problem, $\sum = \{e_1, ..., e_n\}$ be the set of all events of $\mathcal{P}$, ϕ_l be any of the three formulae $\phi_l^{\forall}$, $\phi_l^{\exists}$, $\phi_l^{\exists^*}$ (defined in Section 4), and π be a non-empty causally valid plan for $\mathcal{P}$ obtained by solving ϕ_l. Let $\psi = (S_\psi, \sum, T_\psi, x_0, A_\psi)$

be an FSM that accepts a subsequence $\pi' = \langle e'_1, ..., e'_m \rangle$ of π, and ϕ_l^{ψ} be the encoding of ψ presented by (ψ-1) to (ψ-6). There does not exist any model M for $\phi_l \wedge \phi_l^{\psi}$ such that $\pi = plan(M)$.

Proof. See Appendix A. $\square$

We also need to show that adding ϕ_l^{ψ} to the encoding will not render our planner incapable of producing those plans that do not contain any subsequence accepted by ψ.

Theorem 12. Let $\mathcal{P} = (I, G, A)$ be a temporal planning problem, $\sum = \{e_1, ..., e_n\}$ be the set of all events of $\mathcal{P}$, and ϕ_l be any of the three formulae $\phi_l^{\forall}$, $\phi_l^{\exists}$, $\phi_l^{\exists *}$ (defined in Section 4). Let M be a model that satisfies ϕ_l, and $\pi = \langle e'_1, ..., e'_m \rangle = plan(M)$. Let $\psi = (S_{\psi}, \sum, T_{\psi}, x_0, A_{\psi})$ be an FSM that does not accept any subsequence of π, and ϕ_l^{ψ} be the encoding of ψ composed of (ψ-1) to (ψ-6). There exists a model M' for $\phi_l \wedge \phi_l^{\psi}$ such that $\pi = plan(M')$.

Proof. See Appendix A. $\square$

We now explain how a sequence of events that introduce a negative cycle in the STN of a causally valid plan can be used to prevent similar negative cycles from reoccurring in future plans produced for the problem at hand. Let $\mathcal{P}$ be a temporal planning problem, $\sum$ be the set of all events of $\mathcal{P}$, and $\pi = e_1, ..., e_n$ be a causally valid plan for $\mathcal{P}$. Assume that the STN representing the scheduling function of π has a negative cycle N with nodes $x_{i_1}, ..., x_{i_m}$. Note that x_{i_k} is the node corresponding to event e_{i_k} of π. Without loss of generality, we assume that $i_1 < ... < i_m$, i.e., the events of the negative cycle are ordered by the same order that they have in π. Let O_{i_k} be the set of all temporal actions that are started but not finished before reaching e_{i_k} in the sequence $e_{i_1}, ..., e_{i_m}$, and $\Pi_{i_k} = \sum - \{e | action(e) \in O_{i_k}\} - \{e_{i_k}\}$. Consider the regular language L_N over the alphabet $\sum$ defined by $L_N = e_{i_1} \Pi_{i_2}^* e_{i_2} ... \Pi_{i_m}^* e_{i_m}$, where $\Pi_{i_k}^*$ denotes any string of the symbols in Π_{i_k}. In fact, in the strings of L_N, events other than those already present in the current negative cycle can be inserted in the sequence in such a way that the temporal constraints among $e_{i_1}, ..., e_{i_m}$ remain unchanged. To see why we exclude the events of open actions from Π_{i_k}, consider two hypothetical events e_{i_j} and $e_{i_{j'}}$ that are respectively the starting event and the ending event of an action a. Therefore, there is a temporal constraint on the scheduling function τ_π in the form of $\tau_\pi(i_{j'}) - \tau_\pi(i_j) = dur(a)$. Here, if we insert another copy of the ending event of a between these two events, then a is ended before the execution of $e_{i_{j'}}$ and, as a result, $e_{i_{j'}}$ will no longer be the pairing event of e_{i_j}, and the mentioned constraint will no longer exist.

Theorem 13. Let $N = x_{i_1}, ..., x_{i_m}$ be a negative cycle in the STN corresponding to a causally valid plan $\pi = e_1, ..., e_n$ of a temporal problem $\mathcal{P}$, where x_{i_k} is the node corresponding to event e_{i_k} of π. Let π' be another causally valid plan for $\mathcal{P}$. If a subsequence of π' is a member of L_N (defined above), the corresponding STN of π' will also have N as a negative cycle.

Proof. See Appendix A. $\square$

Consrtucting an FSM that accepts L_N is straightforward. Let ψ be that FSM. Theorem 13 shows that if ϕ^{ψ} is added to the encoding of the input problem, ITSAT will still be capable of finding a valid temporal plan, provided that such a plan exists.

6. Empirical Results

In this section, we show how our preprocessing, encoding, and scheduling methods contribute to the overall performance of ITSAT. Since the contribution of the preprocessing part can be investigated only when the encoding is fixed, we first analyze the performance of the three encodings explained in Section 4. We also compare the performance of ITSAT with several state-of-the-art temporal planners on all non-numerical temporal planning problems of the previous International Planning Competitions.

In Section 4, we theoretically showed that our novel relaxed ∃-step encoding is at least as compact as the temporal versions of the ∃-step and ∀-step encodings for fixed ordering (i.e., the number of steps needed by our relaxed ∃-step encoding to solve a given problem is less than or equal to that of the temporal versions of the ∃-step and ∀-step encodings). Here, we empirically show that our relaxed ∃-step often needs a significantly smaller number of steps, compared to the ∃-step and ∀-step encodings. We also show that the mentioned compactness causes our relaxed ∃-step to significantly outperform ∃-step and ∀-step encodings in the benchmark problems in terms of both memory and speed.

In Section 3, we explained two preprocessing methods, namely mutual exclusion analysis and action compression. In this section we show how each of these methods contribute to the overall performance of ITSAT on the benchmark problems. For this purpose, we compare four versions of ITSAT: 1) ITSAT without preprocessing, 2) ITSAT with mutual exclusion analysis, 3) ITSAT with action compression, and 4) ITSAT with both mutual exclusion analysis and action compression. Our experimental results show that each of these methods separately enhance the performance of ITSAT.

In Section 5, we discussed that by adding certain blocking clauses to the encoding of the problem, one can prevent negative cycles from reoccurring in the STNs of the produced causally valid plans. We also introduced a more elaborate method for preventing such negative cycles, by adding some extra clauses that are based on certain Finite State Machines. Here, we empirically show that our FSM-based method is crucial for the efficiency of ITSAT in the problems with required concurrency.

Finally, we compare the performance of ITSAT with that of the state-of-the-art temporal planners, namely OPTIC (Coles et al., 2010), LPG-td (Gerevini et al., 2006), and TFD (Eyerich et al., 2009). OPTIC and TFD have different degrees of temporal expressivity, whereas LPG-td is not temporally expressive at all (i.e., it is not capable of solving the problems with required concurrency). We show that ITSAT significantly outperforms OPTIC and TFD, while it is competitive with LPG-td in many domains.

6.1 Implementation Details

In order to parse the planning problems and domain, and also validating output plans produced by ITSAT, we have used VAL, which is a plan validation tool developed by the organizers of in IPC 2011. The schematic operators of a given domain are instantiated by the objects of the input problem to produce all possible valid ground temporal actions. ITSAT then performs a polynomial reachability analysis to recognize those actions and prepositions that are relevant to the given problem. For this purpose, all the goal conditions are initially assumed to be relevant propositions. Any action that produces a relevant proposition upon starting or ending is considered to be a relevant action. ITSAT then adds all preconditions of starting and ending events of all relevant actions to the current set of relevant propositions. The invariants of relevant actions will be added to this set, too. Updating the sets of relevant propositions and actions are repeated until no further changes will occur in these sets. We then update the set of relevant propositions by omitting some of relevant propositions that are present in the initial state of the given problem. The omitted propositions are those that are not deleted by any relevant action. These propositions will also be omitted from the at-start, at-end, and invariants of all relevant actions. Mutual exclusion analysis and action compression methods described in Section 3, are then performed on the sets of relevant actions and propositions.

As we mentioned in Section 4, in all our encoding methods we assume that there exists a predefined fixed ordering on all events of the given problem. In the current implementation of ITSAT, the ordering in which the events are produced while constructing ground actions, is taken as the presumed fixed ordering of events. The starting event of each action is placed immediately before its corresponding ending event in the mentioned ordering. More elaborate heuristic methods for producing such an ordering may result in a more compact encoding (Rintanen et al., 2006). Investigating such methods is beyond the scope of this paper and is left for future research.

In the current version of ITSAT, we use *Precosat* (Biere, 2009), which is a free off-the-shelf system, as our SAT solver. We have also examined two other SAT solvers, namely *Minisat* (Eén & Biere, 2005) and *Lingeling* (Biere, 2013) for satisfying the formulae. However, *precosat* had the best overall performance among these three SAT solvers; though *Lingeling* had a better performance in terms of memory usage.

Since *Precosat* accepts formulae only in the Conjunctive Normal Form (CNF), all formulae described throughout this paper had to be translated to their equivalent CNF formulae. This has been performed simply by using logical equivalence relations such as $(\phi_1 \rightarrow \phi_2 \leftrightarrow \neg\phi_1 \vee \phi_2)$ and $(\neg(\phi_1 \wedge \phi_2) \leftrightarrow \neg\phi_1 \vee \neg\phi_2)$.

For each problem, we start with a formula with just one step. We set a time limit of three minutes for *precosat* to find a model for the formula. In the case of the failure, we add three more steps to the formula and repeat this process until either a model is found or a predetermined maximum time of 30 minutes is reached. In the case of success in finding a model, a causally valid plan is extracted from the model. The plan is then given to a scheduling process to find a valid temporal plan. If the scheduling function fails, an appropriate FSM is generated and encoded in the problem formula (see Section 5) without increasing the number of steps. The new formula is again given to *Precosat* to find a new model. Although parallel solving of formulae with different number of steps was shown to be more effective than a naïve sequential approach (Rintanen et al., 2006; Streeter & Smith, 2007), our empirical results show that even our simple sequential method is sufficient to outperform current temporal planners in many planning domains. We leave the investigation regarding the effect of using such parallelism for our future research.

All the experiments explained in this section have been conducted on a 3.1GHz corei5 CPU with 4GB main memory. As our benchmark problems, we have used the problem sets of all previous IPCs. These problems are from different planning domains including *zenotravel*, *rovers*, and *depots* of IPC 2004, *airport* of IPC 2006, *pegsol*, *crewplanning*, *openstacks*, *elevators*, *sokoban*, and *parcprinter* of IPC 2011, and *driverlog*, *floortile*, *matchcellar*, *mapanalyser*, *parking*, *rtam*, *satellite*, *storage*, *turnandopen*, and *tms* of IPC 2014. Note that some of these domains have been used in different IPCs. For such domains, we have chosen the problem set of the most recent competition with those domains. That is why no problem set of IPC 2008 is present in our experiments.

Among the domains used in previous IPCs, only *matchcellar*, *turnandopen*, and *tms* include problems with the required concurrency. These are the problem sets in which only temporally expressive planners are capable of producing valid plans. In order to achieve a better assessment of ITSAT in problems with required concurrency, we have used two extra domains *driverlogshift* and *matchlift* (Halsey, 2004). We have also performed our experiments on time-window variants of *satellite* and *airport* domains. These domains, which have been used in IPC 2004 and do have required concurrency, are referred throughout this section by *satellite-tw* and *airport-tw*, respectively. The mentioned domains with the required concurrency are explained in more details in Section 6.4.

6.2 The Impact of Different Encoding Methods

To evaluate our $\forall$-step, $\exists$-step, and relaxed $\exists$-step encodings we have produced three different versions of ITSAT, namely, ITSAT-$\forall$, ITSAT-$\exists$, and ITSAT-$\exists^*$, respectively. In all these versions, the formula ϕ^{mut}, which encodes the mutex relations, is also added to the encoding. None of these versions take advantage of action compression. The negative cycle prevention method described in Section 5 has been used in all the three versions of ITSAT. Table 2 shows a comparison in each domain among these versions with regard to the number of solved problems.

As it can be seen in Table 2, ITSAT-$\exists^*$ has the best performance among the three versions. In fact, ITSAT-$\exists^*$ has been able to solve 65 problems more than ITSAT-$\exists$, and 103 problems more than ITSAT-$\forall$. Furthermore, almost all problems solved by ITSAT-$\forall$ or ITSAT-$\exists$ were also solved by ITSAT-$\exists^*$. This means that our relaxed $\exists$-step encoding is significantly more efficient than the temporal versions of the classical $\exists$-step and $\forall$-step encodings.

Table 3, shows a more detailed comparison among the mentioned encodings. The different columns of Table 3 represent the following items: the name of the domain, the problem number, the used encoding method, the number of steps in the encoding, the result of *Precosat* in terms of satisfiability or unsatisfiability of the formula, the number of clauses and variables divided by 1000, the amount of time taken by *Precosat* to determine the result, and the amount of memory needed for solving the formula. For each problem and each encoding method, the results are presented for two cases: unsatisfiable formula with the highest number of steps, and satisfiable formula with the lowest number of steps. Note that to produce these results, we have increased the number of steps by one when a formula was unsatisfiable. Symbol ∞ is used in the *time* column for those cases in which *Precosat* has failed to find a model for the formula in 1800 seconds. The results are presented only for those domains in which at least one of the problems has been solved by at least two of the planners. Accordingly, *openstacks*, *elevators*, *matchcellar*, and *rtam* have been omitted

domain	problems	solved		
		ITSAT-∀	ITSAT-∃	ITSAT-∃*
zenotravel	20	13	16	16
rovers	20	18	18	20
depots	22	13	17	19
satellite-tw	36	3	3	3
airport-tw	50	19	21	21
airport	50	20	21	38
pegsol	20	20	20	20
crewplanning	20	8	8	20
openstacks	20	0	0	9
elevators	20	0	0	0
sokoban	20	2	3	2
parcprinter	20	15	16	17
driverlog	20	0	2	3
floortile	20	10	16	20
mapanalyser	20	14	19	19
matchcellar	20	0	0	18
parking	20	10	10	10
rtam	20	0	0	0
satellite	20	0	3	3
storage	20	0	9	9
turnandopen	20	1	2	2
tms	20	18	18	18
driverlogshift	10	10	10	10
matchlift	14	14	14	14
total	542	208	246	311

Table 2: Overall Comparison of Different Encoding Methods

from Table 3. Moreover, in *satellite* and *storage*, the results are only presented for the $\exists^*$-step and $\exists$-step encodings.

In each domain, Table 3 presents only the results for the hardest problem (i.e., the problem with the greatest number of propositions). In our experiments, we have observed a pattern similar to that of the chosen problems for other problems of each domain. Note that the results presented in Table 3 are only for finding the first causally valid plan. Therefore, these results do not include the information regarding FSM encoding method described in Section 5. We will explain the impact of our FSM-based negative cycle prevention later in this section.

domain	prob	encoding	steps	result	$\frac{C}{1000}$	$\frac{V}{1000}$	time (s)	mem (MB)
zenotravel	13	$\forall$	10	F	474	136	25	131
		$\exists$	4	F	133	33	0.4	18
		$\exists^*$	3	F	69	23	0.38	10
		$\forall$	11	T	521	150	33	146
		$\exists$	5	T	167	42	0.54	20
		$\exists^*$	4	T	92	32	0.56	17
rovers	18	$\forall$	26	F	5467	560	∞	413
		$\exists$	25	F	4444	412	∞	244
		$\exists^*$	2	F	375	15	0.14	12
		$\forall$	27	T	5673	581	89	353
		$\exists$	26	T	4618	428	47	220
		$\exists^*$	3	T	511	24	0.36	12
depots	17	$\forall$	10	F	7950	1259	6.6	598
		$\exists$	6	F	3256	455	2.2	284
		$\exists^*$	3	F	1008	236	1.39	129
		$\forall$	11	T	8747	1387	8.9	1001
		$\exists$	7	T	3802	534	3	301
		$\exists^*$	4	T	1336	326	2.1	141
satellite-tw	3	$\forall$	12	F	33	9	1.2	3
		$\exists$	12	F	26	7	1.4	2
		$\exists^*$	5	F	7	3	0.1	1
		$\forall$	13	T	36	10	0.7	4
		$\exists$	13	T	28	8	0.7	4
		$\exists^*$	6	T	8	3	0.1	1
airport-tw	19	$\forall$	72	F	5415	1516	24	912
		$\exists$	33	F	1811	412	3.7	271
		$\exists^*$	7	F	243	44	0.43	22
		$\forall$	73	T	5573	1541	19	900
		$\exists$	34	T	1817	439	4.1	283
		$\exists^*$	8	T	270	49	0.5	28
airport	20	$\forall$	70	F	9261	1381	27	837
		$\exists$	31	F	3189	361	3.5	257
		$\exists^*$	5	F	430	37	0.16	31
		$\forall$	71	T	9392	1401	21	844
		$\exists$	32	T	3290	373	4.3	263
		$\exists^*$	6	T	506	44	0.62	36

Continued on next page

domain	prob	encoding	steps	result	$\frac{C}{1000}$	$\frac{V}{1000}$	time (s)	mem (MB)
pegsol	20	∀	12	F	34	9	∞	28
		∃	12	F	25	6	∞	26
		∃*	5	F	10	3	0.05	1
		∀	13	T	37	10	22.4	14
		∃	13	T	27	7	7.3	11
		∃*	6	T	12	4	0.3	3
crewplanning	8	∀	69	F	112	34	∞	40
		∃	69	F	102	29	∞	49
		∃*	4	F	3	1	0	1
		∀	70	T	114	35	30	44
		∃	70	T	104	30	15	46
		∃*	5	T	3	1	0	1
sokoban	4	∀	19	F	542	143	∞	163
		∃	13	F	274	64	12	78
		∃*	6	F	122	37	3.9	23
		∀	20	T	571	150	74	190
		∃	14	T	295	69	40	125
		∃*	7	T	142	43	1.5	20
parcprinter	20	∀	42	F	2325	382	∞	319
		∃	31	F	1436	198	30	119
		∃*	8	F	345	51	1.1	24
		∀	43	T	2380	391	93	289
		∃	32	T	1482	205	25	128
		∃*	9	T	385	58	0.7	26
driverlog	2	∃	7	F	2634	258	6.2	165
		∃*	5	F	2803	184	14	70
		∃	8	T	5248	294	23	280
		∃*	6	T	3291	221	13.9	131
floortile	10	∀	22	F	201	21	55	48
		∃	11	F	74	11	7	16
		∃*	6	F	31	6	0.14	5
		∀	23	T	211	22	20	49
		∃	12	T	80	12	0.62	13
		∃*	7	T	36	7	0.16	5
mapanalyser	15	∀	17	F	187009	1005	18	1196
		∃	9	F	101893	534	10	519
		∃*	2	F	29988	122	5.4	75

Continued on next page

domain	prob	encoding	steps	result	$\frac{C}{1000}$	$\frac{V}{1000}$	time (s)	mem (MB)
		∀	18	T	197433	1064	26	1177
		∃	10	T	112123	593	11	538
		∃*	3	T	40062	181	6.7	128
parking	11	∀	4	F	1769	150	2.5	177
		∃	2	F	605	75	0.5	70
		∃*	1	F	238	38	0.4	34
		∀	5	T	1868	187	4.2	297
		∃	3	T	911	112	2.5	87
		∃*	2	T	470	75	2	72
satellite	3	∃	11	F	870	221	∞	431
		∃*	5	F	294	91	∞	91
		∃	12	T	950	244	157	440
		∃*	6	T	352	110	4.9	95
storage	9	∃	11	F	1523	125	∞	198
		∃*	8	F	633	84	48	69
		∃	12	T	1662	136	5.9	149
		∃*	9	T	710	94	9.4	72
turnandopen	1	∀	42	F	2416	203	∞	454
		∃	22	F	944	106	33	112
		∃*	10	F	371	53	1.1	41
		∀	43	T	2474	207	176	465
		∃	23	T	987	111	37	106
		∃*	11	T	406	58	1.3	42
tms	18	∀	9	F	491	51	2.3	65
		∃	7	F	257	39	0.5	42
		∃*	3	F	91	18	0.1	14
		∀	10	T	546	56	1.4	67
		∃	8	T	284	43	0.5	42
		∃*	4	T	115	23	0.3	17
driverlogshift	11	∀	18	F	373	65	1.5	35
		∃	15	F	235	37	0	19
		∃*	9	F	54	17	0	7
		∀	19	T	411	69	0.6	39
		∃	16	T	260	39	0.3	21
		∃*	10	T	57	20	0.3	9

Table 3: Detailed Comparison of the Encoding Methods

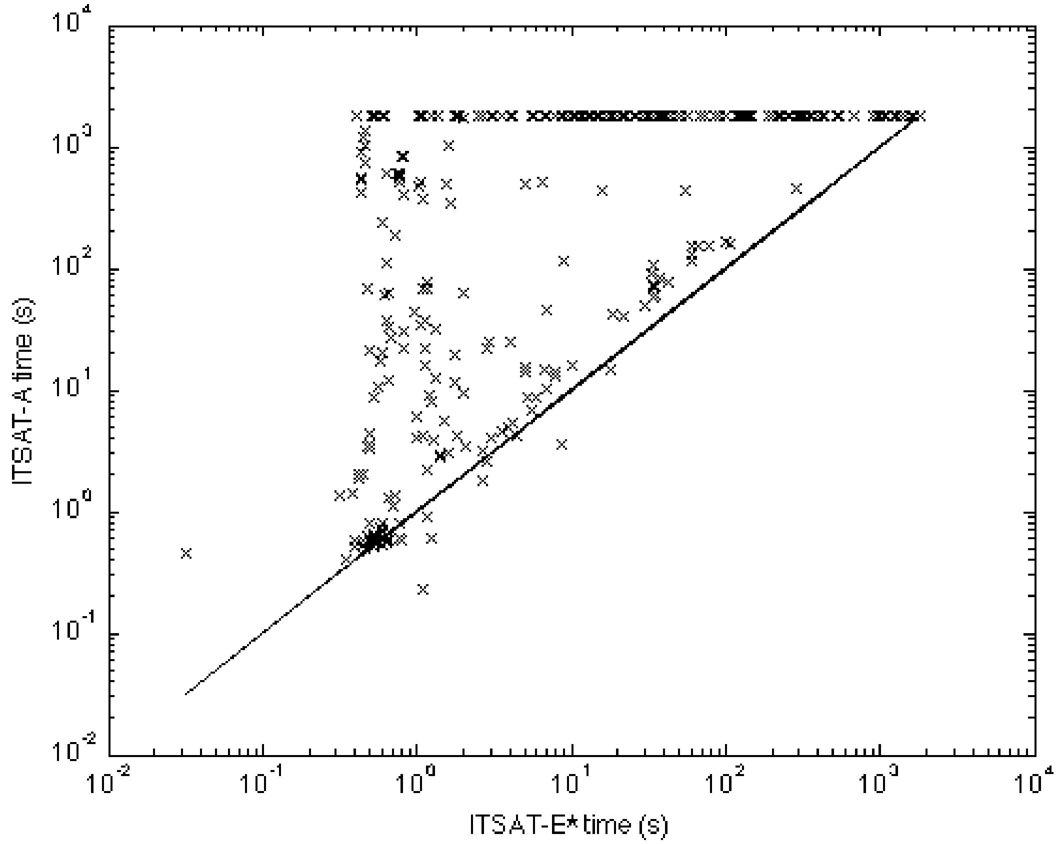

Figure 11. Speed Comparison Between ITSAT-$\exists^*$ and ITSAT-$\forall$

In Section 4, we theoretically showed that in order to solve a given planning problem, our relaxed $\exists$-step encoding requires fewer steps than the temporal versions of classical $\exists$-step and $\forall$-step encodings when the ordering is fixed. Table 3 shows that ITSAT-$\exists^*$ often needs a considerably smaller number of steps. This phenomenon is most prominent in *airport*, *crewplanning*, and *mapanalyser*. Moreover, in *openstacks* and *matchcellar*, neither ITSAT-$\exists$ nor ITSAT-$\forall$ was able to solve any problem due to the very large number of steps that had been required. This suggests a correlation between the performance of the planner, and the compactness of the encoding. Generally speaking, when a relatively high number of steps is needed for $\forall$-step encoding to solve a problem, we can deduce that there is a strong causal connection between the actions of the produced plan. On the other hand, our $\exists^*$-step encoding has been devised to take advantage of such causal connections. Therefore, the $\exists^*$-step encoding is expected to have more advantage over the $\forall$-step encoding in such domains. This phenomenon is most visible in *airport*, *crewplanning*, *openstacks*, and *matchcellar* domains, as the numbers of steps required by the $\forall$-step encoding in these domains are exceptionally high. Table 3 also shows that our relaxed $\exists$-step encoding results in a significant improvement of the planner in terms of the memory usage.

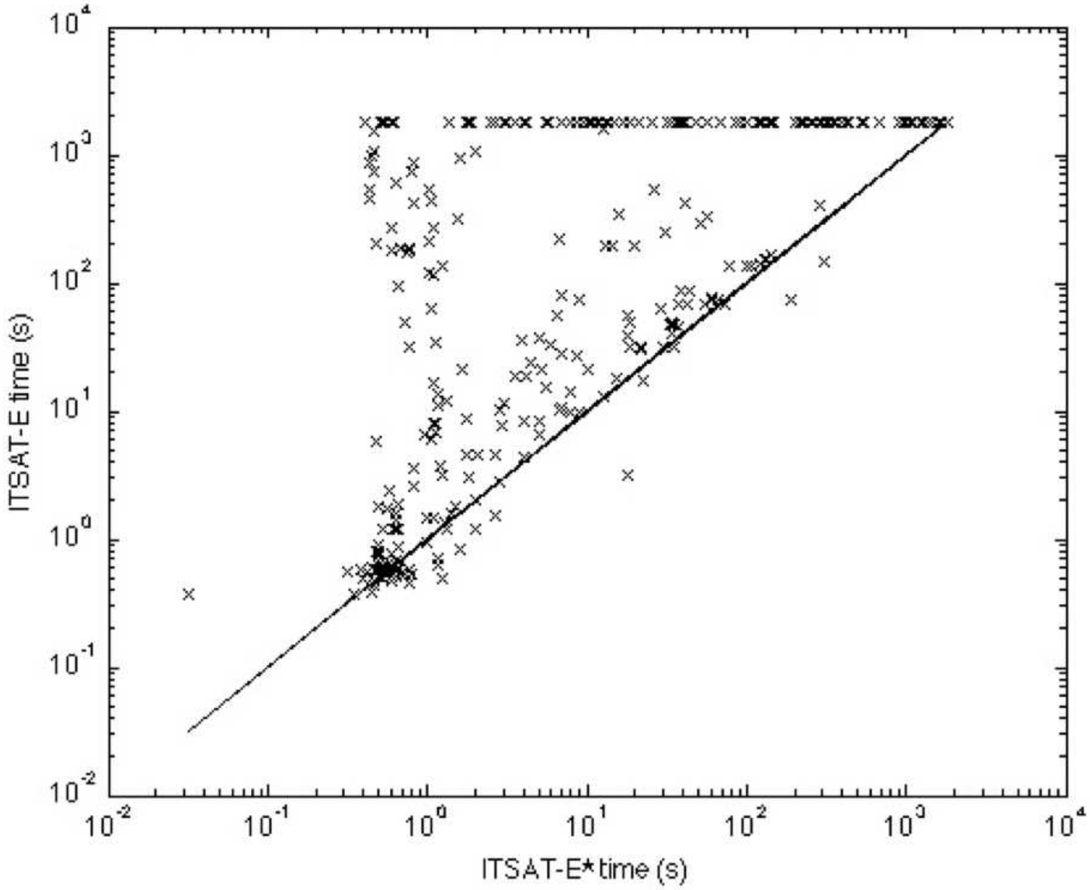

Figure 12. Speed Comparison Between ITSAT-$\exists^*$ and ITSAT-$\exists$

We have also compared the speed of ITSAT-$\exists^*$ with that of ITSAT-$\exists$ and ITSAT-$\forall$ in solving all the benchmark problems. The results are depicted in Figure 11 and Figure 12. As it can be seen, ITSAT-$\exists^*$ has outperformed ITSAT-$\exists$ and ITSAT-$\forall$ in almost all the problems.

6.3 The Impact of the Mutual Exclusion Analysis and Action Compression

In Section 3, we explained how mutual exclusion analysis and action compression are performed as preprocessing components of ITSAT. Here, we empirically show that these components are both quite effective in enhancing the performance of our planner. As we showed before, $\phi^{\exists^*}$ is the encoding that results the best performance of ITSAT. We have fixed this formula as the base of comparison, and produced three other formulae to investigate the impact of each preprocessing method. These three formulae are $\phi^{\exists^*} \wedge \phi^{mut}$ (the base encoding plus the mutual exclusion information), $\phi^{\exists^*} \wedge \phi^{com}$ (the base encoding plus the action compression information), and $\phi^{\exists^*} \wedge \phi^{mut} \wedge \phi^{com}$ (the base encoding plus both mutual exclusion and action compression information). Table 4 shows the number of problems solved by each of the mentioned versions of ITSAT.

domain	problems	$\phi^{\exists^*}$	$\phi^{\exists^*} \wedge \phi^{mut}$	$\phi^{\exists^*} \wedge \phi^{com}$	$\phi^{\exists^*} \wedge \phi^{mut} \wedge \phi^{com}$
zenotravel	20	13	16	14	18
rovers	20	20	20	20	20
depots	22	11	19	14	20
satellite-tw	36	3	3	3	3
airport-tw	50	21	21	21	21
airport	50	38	38	38	39
pegsol	20	20	20	20	20
crewplanning	20	20	20	20	20
openstacks	20	1	9	5	13
elevators	20	0	0	0	0
sokoban	20	1	2	3	8
parcprinter	20	17	17	19	20
driverlog	20	0	3	1	4
floortile	20	3	20	20	20
mapanalyser	20	14	19	19	20
matchcellar	20	0	18	20	20
parking	20	3	10	7	17
rtam	20	0	0	0	0
satellite	20	0	3	0	16
storage	20	0	9	0	20
turnandpen	20	2	2	3	9
tms	20	0	18	18	18
driverlogshift	10	10	10	10	10
matchlift	14	14	14	14	14
total	542	211	311	289	370

Table 4: The Impact of the Mutual Exclusion Analysis and Action Compression

As it can be seen in Table 4, these preprocessing methods result in a significant improvement in terms of overall coverage. In fact, the version of ITSAT that uses both methods solves 159 problems more than the base planner. Besides, the version that uses both methods even considerably outperforms any of the two versions that only use one preprocessing method. This suggest that both preprocessing components are necessary for producing the best performance of ITSAT.

To investigate the effectiveness of our action compression method in the domains that ITSAT compresses considerably more actions than CRIKEY, we have performed another experiment. We compressed only the actions that ITSAT considers compression safe but CRIKEY3 does not. In this new version of ITSAT, we also used the mutual exclusion information. This version of ITSAT solves six problems more than the version where only mutual exclusion information was used: of these six problems, four problems are from zenotravel, one is from airport, and one is from mapanalyser. The results does not change much in other domains. Note that the three mentioned domains are those in which ITSAT compresses considerably more actions than CRIKEY.

6.4 The Impact of the FSM-Based Negative Cycle Detection

As mentioned earlier, among the domains used to evaluate ITSAT, *matchcellar*, *turnandopen*, *tms*, *driverlogshift*, *matchlift*, and time-window versions of *airport* and *satellite* can have problems with required concurrency. In fact, in these domains, it may be impossible to schedule a causally valid plan produced by solving a SAT formula, into a valid temporal plan. Here, we briefly explain why the problems in each of these domains may require concurrency, and how this may introduce some negative cycles into the STN associated with a causally valid plan.

In *matchcellar* and *matchlift*, there exists an action for lighting a match. This action produces light for a certain amount of time. The objective is to mend some fuses. The actions for mending a fuse can only be executed if there is light. As a result, the actions of lighting a match and mending a fuse must be executed concurrent. However, in a causally valid plan, since the planner does not consider the durations of actions, it may assume that any match can remain lit until all the fuses are mended. As it was discussed in Section 5, this can introduce a negative cycle into the STN of a produced causally valid plan.

In *tms*, the objective is to produce a certain number of ceramic structures. These structures need several preparations that can be done only while a furnace is producing heat. It should be clear that this domain is very similar to *matchcellar* and *matchlift*, and requires concurrency in a similar way.

A simplified version of *driverlogshift* was introduced in Section 2. The only difference between that simplified version and the one used in this section to evaluate the planners, is that here, there are drivers that can walk to, board, and disembark trucks. Furthermore, the `REST` and `WORK` actions are performed by the drivers rather than trucks. In this domain, the working shifts of the drivers are analogous to the action of lighting a match in the *matchcellar* domain.

In *turnandopen*, there exists a robot that needs to move between a number of rooms. There are doors between each pair of adjacent rooms. All these doors, which are closed in the initial state, should be opened by the robot. The robot can open a door only while it is turning the doorknob. In this domain, the actions of turning the doorknob and opening the door must be executed concurrently. However, the duration of the action for turning the knob is 3, whereas that of opening the door is 2. This enables ITSAT to schedule every causally valid plan into a valid temporal plan. Therefore, preventing negative cycles is not necessary in this domain.

In time-window versions of *airport* and *satellite*, there is a specific time by which all goals must be obtained. Such a deadline is introduced into a problem by using a specific frame action with the duration equal to the time of that deadline. All other actions can be executed only when this frame action is being executed. In other words, all the actions must be concurrent with the is frame action. However, in a causally valid plan, since the planner does not consider the durations of actions, it may assume that the frame action can

domain	problem	restarts	$\frac{C}{1000}$ $\exists^*$	FSM	$\frac{V}{1000}$ $\exists^*$	FSM	memory (MB) $\exists^*$	FSM
satellite-tw	3	121	12	1450	3	318	1	382
airport-tw	21	1	552	5	65	2	41	1
matchcellar	20	34	44	408	22	203	9	77
tms	18	3	121	12	23	10	16	2
driverlogshift	10	43	68	930	26	280	13	107
matchlift	14	9	72	180	15	112	31	98

Table 5: The Collective Size of the SAT Encodings of FSMs

be arbitrarily long, and thereby neglect to meet the deadline for achieving the goals. This can introduce negative cycles into the STNs of produced causally valid plans.

As we explained in Section 5, if the STN of a causally valid plan includes a negative cycle, we must force the SAT solver to find a different solution. This can be done simply by adding an extra blocking clause to the current SAT formula to prevent at least one of the events of that negative cycle from reoccurring in its current step. Alternatively, we introduced a more elaborate method for doing the same thing, by adding the encoding of certain FSMs into the encoding. In this method, when the STN of a causally valid plan with k steps includes a negative cycle, an FSM that detects that negative cycle will be encoded into a SAT formula, and the solver will be restarted. In order to decrease the number of restarts, whenever a sequence of events corresponding to a negative cycle is found, ITSAT tries to find other potential negative cycles by replacing the actions of current sequence by other actions of the problem and checking the STN of the resulting sequence for negative cycles.

Table 5 shows the collective size of the SAT encodings of the FSMs required for solving the problems of the above domains. As our base encoding, we have used the relaxed $\exists$-step encoding with both mutual exclusion analysis and action compression. For each domain, the results are shown in Table 5 only for the hardest problem solved by ITSAT. The *turnandopen* domain has been excluded from Table 5, as no negative cycle is encountered when solving problems of this domain. The different columns of Table 5 represent the following items: the name of the domain, the problem number, the number of clauses and variables divided by 1000, and the amount of memory needed to produce the formula. The results for the number of causes, number of variables, and the used memory are presented with separated columns for the base encoding and the encoding of the FSMs.

As it can be seen in Table 5, our negative cycle prevention method helps ITSAT solve a considerable number of problems with required concurrency. Nevertheless, the SAT encoding of the required FSMs is significantly larger than the base encoding in the domains where the number of restarts are relatively high. On the other hand, as the number of the restarts increase, the speed of ITSAT declines. That is because after each restart, the SAT solver must verify the satisfiability of the formula, from scratch. In fact, these numerous restarts are the main reason for the poor performance of ITSAT in the time-window version of *satellite*.

6.5 ITSAT Versus the State-of-the-art Temporal Planners

We now compare ITSAT with three efficient temporal planners, namely, OPTIC (Benton, Coles, & Coles, 2012), TFD (Eyerich et al., 2009), and LPG-td (Gerevini et al., 2006). Because of the similarities between the approach used in ITSAT and that of SCP2 (Lu et al., 2013), we have also included the results of this planner in our experimental results.

OPTIC is the newest version of POPF (Coles et al., 2010). It is a heuristic state-space temporal planner based on the so-called temporally-lifted progression planning (Cushing et al., 2007). Using this approach of planning enables OPTIC to solve problems with required concurrency. Besides, OPTIC handles self-overlapping actions, which makes it more expressive than ITSAT. Although handling self-overlapping actions is hardly necessary for solving non-numerical temporal planning problems (Fox & Long, 2007), among the current benchmark domains, *zenotravel*, *rovers*, and *airport* do permit such actions due to some modeling errors. For a fair comparison between ITSAT and OPTIC, we have used the corrected versions of these three domains[1] in our evaluations. For guiding its search, OPTIC benefits from a heuristic function that is based on the relaxed planning graph (Hoffmann & Nebel, 2001).

TFD is another heuristic state-space temporal planner. TFD is based on the so-called decision epoch planning (Cushing et al., 2007). The planners that use this approach are not as temporally expressive as the planners that are based on temporally lifted progression planning. In other words, in theory, there are temporal planning problems defined in PDDL2.1 that can be solved by ITSAT and OPTIC but not by TFD. However, all the current benchmark problems can potentially be solved by using decision epoch planning. For guiding its search, TFD benefits from a temporal version of the so-called Context-enhanced Additive Heuristic (Helmert & Geffner, 2008).

LPG-td is a very fast temporal planner, which is not temporally expressive. In fact, LPG-td at first generates a sequential plan for a given problem, and then tries to reschedule the plan to produce one with an improved quality. This renders LPG-td incapable of solving any problem in *matchcellar*, *turnandopen*, *tms*, *driverlogshift*, or *matchlift*. Similar to OPTIC, LPG-td benefits from a heuristic based on the relaxed planning graph. However, instead of searching in the state space of the problem, LPG-td performs its search by making some local improvements to a structure that is similar to partial plans, which is called Linear Action Graph. Two different configurations of LPG-td can be used based on whether we prefer speed of the planner or the quality of produced plans. Here, we only present the results of the *quality* configuration of LPG-td, as it produced better results than the *speed* configuration in our experiments.

SCP2 (Lu et al., 2013), is a SAT-based temporal planner that uses a discrete representation of time. This planner assigns explicit discrete time labels to each step of the encoding. In this approach, each step i is exactly one time unit ahead of step $i + 1$. As a result, if an action with duration d starts in step i, it is forced to end in step $i + d$. This means that the number of layers required for producing a plan π is greater than or equal to the makespan of π. SPC2 starts with a formula with only one step, an increases the number of steps by one, every time the formula is unsatisfiable. This enables SPC2 to find the optimal plan for

1. The corrected version of mentioned domains can be downloaded from the official website of POPF planner.

a number of given problems. To obtain a better performance, SCP2 uses ∃-step semantic to allow causal relations between actions in each time point.

We have compared ITSAT with the above planners based on the number of problems they can solve in each domain and also by the total score given to each planner using the scoring strategy of recent IPCs; that is, if a planner cannot solve a problem, it will get a score of 0 for that problem; Otherwise, its score will be equal to the makespan of the best produced plan divided by the makespan of the plan found by this planner. The results are presented in Table 6.

As it can be seen in Table 6, ITSAT significantly outperforms OPTIC, TFD, and SCP2. In fact, ITSAT solves 162 problems more than OPTIC, 145 problems more than TFD, and 282 problems more than SCP2. ITSAT also solves 64 problems more than LPG-td. However, this is mainly because LPG-td is incapable of solving problems with required concurrency. If we exclude *satellite-tw*, *airport-tw*, *matchcellar*, *turnandopen*, *tms*, *driverlogshift*, and *matchlift* that are the domains in which LPG-td cannot solve any problem, ITSAT solves only 31 problems less than LPG-td. This shows that ITSAT is quite competitive with LPG-td even in solving the problems without required concurrency.

As it is shown in Table 6, OPTIC solves zero problems in *parcprinter*, *driverlog*, *floortile*, *mapanalyser*, *matchcellar*, *rtam*, *storage*, and *tms*. In all of these domains, the main reason of poor performance of OPTIC is that it runs out of memory, early during its search. TFD solves zero problems in *satellite-tw*, *airport-tw*, *parcprinter*, *driverlog*, *floortile*, *rtam*, *storage*, and *tms*. Except for *parcprinter*, in which TFD runs out of memory, in other domains TFD performs poorly because it is unable to find a plan within 1800 seconds. As we mentioned before, LPG-td solves zero problems in the domains with required concurrency. The performance of SCP2 is rather poor in many of the benchmark domains. The reason of the poor performance of SCP2 is that, for many of the benchmark problems, the makespan of the optimal plan is relatively large. As a result, in most of the problems, SCP2 is unable to check the satisfiability of all the formulae with numbers of steps less than the makespan of the optimal plan, within the 1800 seconds time limit.

To compare the quality of the plans produced by ITSAT with those of other competing planners, consider Table 7. The numbers presented in Table 7 are the average makespan ratio of plans mutually solved by the corresponding planner and ITSAT in the corresponding domain. Ratios less than one indicate better average quality of the solutions produced by ITSAT in comparison with the competing planners. For those cases that neither ITSAT nor the competing planner has been able to solve any problem of a domain, the corresponding cell of Table 7 has remained blank.

We have also performed some experiments based on a number of two planners portfolios of different pairs of the above planners. The portfolios enabled us to combine the advantages of two planners. To do this, the 30 minutes time limit is divided equally between each pair of planners. The results of running these portfolios are presented in Table 8. The results show that the best configuration has been obtained by combining ITSAT and LPG-td. The resulting planner was capable of solving 423 out of 542 benchmark problems. Moreover, each of these planners has produced its best results when it was combined with ITSAT.

		solved					IPC score				
domain	N	ITSAT	OPTIC	TFD	LPG-td	SCP2	ITSAT	OPTIC	TFD	LPG-td	SCP2
zenotravel	20	18	12	12	20	1	11.41	10.56	11.32	17.68	1
rovers	20	20	20	20	20	4	18.25	18.35	18.56	16.88	4
depots	22	20	7	5	21	6	9.52	4.21	3.04	19.43	6
satellite-tw	36	3	4	0	0	0	3	4	0	0	0
airport-tw	50	21	7	0	0	0	21	7	0	0	0
airport	50	39	24	20	43	0	35.2	23.35	18.11	39.68	0
pegsol	20	20	19	18	20	20	19.36	18.02	17.24	18.98	20
crewplanning	20	20	20	14	9	0	18.37	20	11.94	7.82	0
openstacks	20	13	20	20	20	0	7.26	17.01	19.83	15.23	0
elevators	20	0	1	3	9	0	0	1	3	7.44	0
sokoban	20	8	2	3	5	1	7.64	1.72	3	3.26	1
parcprinter	20	20	0	0	7	0	20	0	0	5.72	0
driverlog	20	4	0	0	14	0	3.89	0	0	13.38	0
floortile	20	20	0	0	20	10	17.05	0	0	16.51	10
mapanalyser	20	20	0	19	20	0	18.48	0	15.62	14.81	0
matchcellar	20	20	20	20	0	0	20	20	16	0	0
parking	20	17	16	20	20	6	5.05	13.98	18.72	19.02	6
rtam	20	0	0	0	20	0	0	0	0	20	0
satellite	20	16	4	13	20	0	2.61	3.55	7.05	20	0
storage	20	20	0	0	18	20	18.77	0	0	1.45	20
turnandopen	20	9	9	18	0	1	9	6.52	13.34	0	1
tms	20	18	0	0	0	0	18	0	0	0	0
driverlogshift	10	10	10	6	0	9	8.01	9.48	5.03	0	9
matchlift	14	14	13	14	0	13	14	13	14	0	13
total	542	370	208	225	306	88	305.87	191.75	195.8	256.29	88

Table 6: ITSAT Versus State-of-the-art Temporal Planners

domain	OPTIC	TFD	LPG-td	SCP2
zenotravel	1.47	1.50	1.41	1
rovers	0.99	1.03	0.89	1.27
depots	1.24	1.41	2.02	2.11
satellite-tw	1	—	—	—
airport-tw	1	—	—	—
airport	1.06	0.95	0.91	—
pegsol	0.98	0.98	0.96	1.02
crewplanning	1.11	0.89	0.88	—
openstacks	1.34	1.90	1.31	—
elevators	—	—	—	—
sokoban	0.86	1.18	0.69	1
parcprinter	—	—	0.81	—
driverlog	—	—	1.03	—
floortile	—	—	0.97	1.05
mapanalyser	—	0.86	0.79	—
matchcellar	1	0.80	—	—
parking	2.94	2.98	3.01	3.15
rtam	—	—	—	—
satellite	6.72	4.45	7.35	—
storage	—	—	0.11	1.05
turnandopen	0.72	0.48	—	1
tms	—	—	—	—
driverlogshift	1.14	0.98	—	1.19
matchlift	1	1	—	1

Table 7: Average Makespan Ratio

	ITSAT	OPTIC	TFD	LPG-td	SCP2
ITSAT	—	375	385	423	348
OPTIC	375	—	262	350	237
TFD	385	262	—	360	256
LPG-td	423	350	360		302
SCP2	348	237	256	302	—

Table 8: 2-planners Portfolio

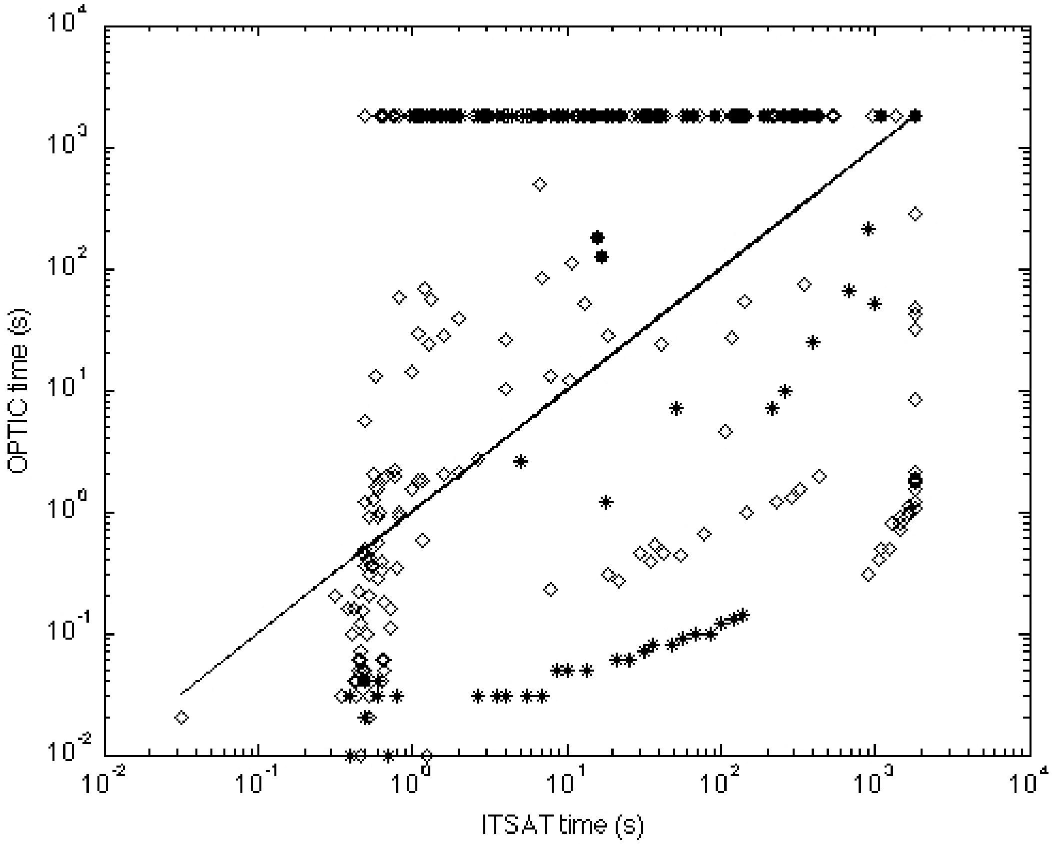

Figure 13. Speed Comparison Between ITSAT and OPTIC

Although ITSAT is quite competitive with the state-of-the-art temporal planners, our empirical results reveal a few drawbacks of the planner. We have compared the speed of ITSAT with that of OPTIC, TFD, LPG-td, and SCP2 on all of our benchmark problems. The results are presented in Figure 13, Figure 14, Figure 15, and Figure 16, respectively. In these figures, the results for the required concurrency domains have been separated from that of other domains by using different symbols in the scatterplots: the star symbol represents the problems with required concurrency, and the diamond symbol represents other problems. As it can be seen, ITSAT is slower than OPTIC, TFD, and LPG-td in a number of the benchmark problems. A major cause of this drawback is that in ITSAT, the SAT solver spends too much time refuting several formulae before it will finally find the first satisfiable formula. As it has been shown in the case of classical planning, the speed of SAT-based planners can be significantly improved by checking the satisfiability of several formulae with different number of steps in parallel. We discuss this in more detail in Section 7 as our future research.

Another observation is that, ITSAT performs rather slowly in solving a number of problems with required concurrency that are quickly solved by OPTIC and TFD. This is mainly due to restarting the SAT solver whenever negative cycles are encountered. As we

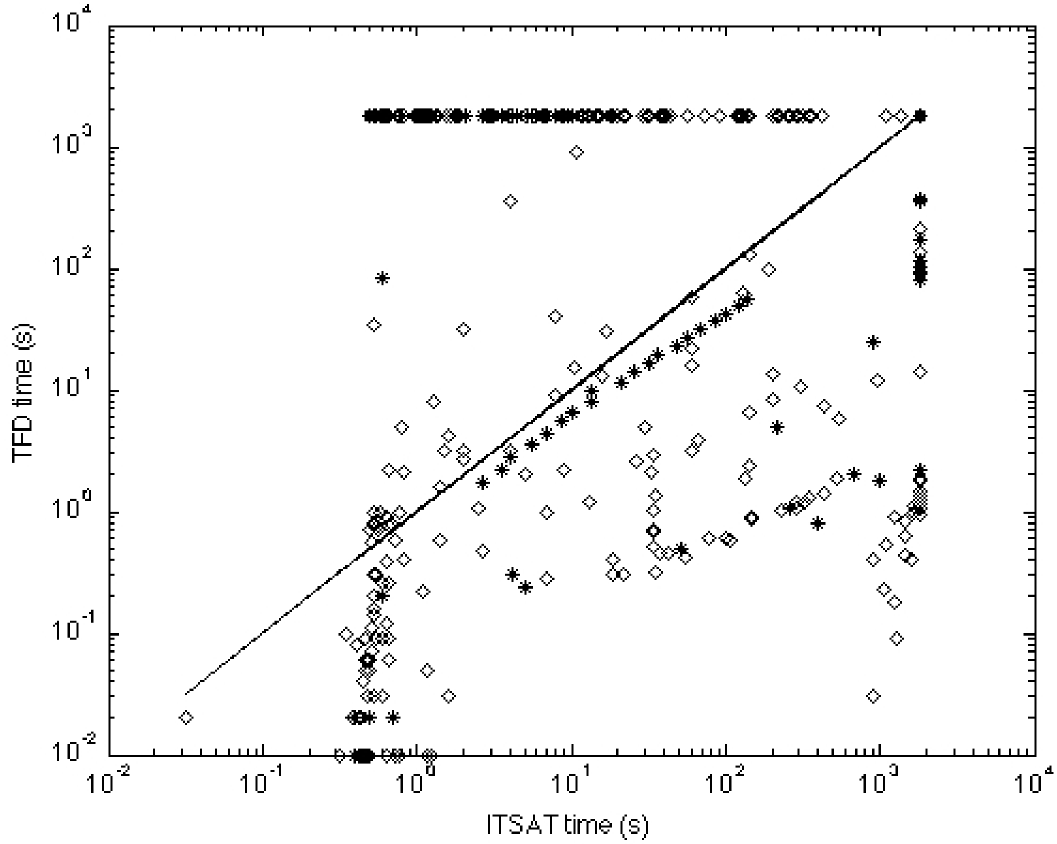

Figure 14. Speed Comparison Between ITSAT and TFD

explained earlier, when the STN of a causally valid plan with k steps includes a negative cycle, an FSM detecting that negative cycle will be encoded into a SAT formula, and the solver will try to satisfy the new formula with k steps, from scratch. In the domains where these negative cycles are abundant, the performance of ITSAT can be significantly affected by the numerous restarts of the SAT solver.

The performance of ITSAT is particularly poor in three domains, namely *elevators*, *driverlog*, and *rtam*. In these domains, the number of ground actions is higher than that of other domains. A linear increase in the number of ground actions may cause an exponential growth in the size of the search space of the problem. To tackle this problem, state-space based planners take advantage of heuristic functions that are devised specially for pruning the search space of planning problems. It has also been shown that using SAT solvers tailored for solving planning problems can result in a significant improvement of the performance of SAT-based classical planning (Rintanen, 2012). We think that employing this idea can also improve the speed of ITSAT in the mentioned domains.

Another drawback of ITSAT is the poor quality of its produced plans in some benchmark domains. Most notably, although ITSAT solves most of the problems in *depots*, *openstacks*, *parking*, and *satellite*, the quality of its plans is rather low in these domains, according to

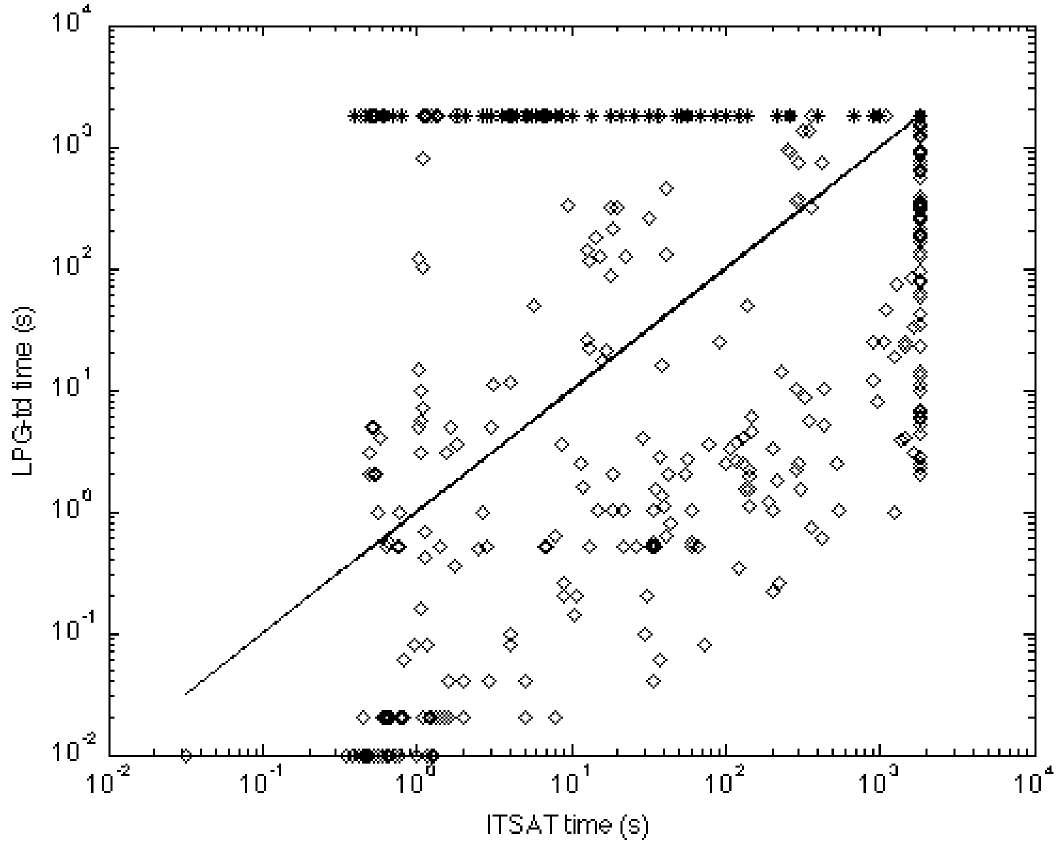

Figure 15. Speed Comparison Between ITSAT and LPG-td

Table 7. This is mainly due to the fact that ITSAT abstracts out the duration of actions, and thus, its SAT solver lacks the competency for evaluating the quality of the plans that are being produced. Nevertheless, the quality of the plans produced by ITSAT is generally comparable to that of other planners in most benchmark domains. In Section 7, we explain an idea for improving the quality of the plans produced by ITSAT.

7. Conclusions and Future Research

In this paper, we described ITSAT, a temporally expressive SAT-based planner. ITSAT is based on an approach that takes advantage of parallel encodings. In this approach, at first, the durations of all actions of a given problem are abstracted out. Then the abstract problem is encoded into a SAT formula using $\forall$-step and $\exists$-step semantics for causally valid plans. After generating a causally valid plan, ITSAT performs a scheduling process. During this process, ITSAT tries to satisfy those temporal constraints that are imposed by considering the durations of actions. This is done by solving a Simple Temporal Problem (STP). In the cases with an inconsistent STP, the cause, which is a negative cycle in the corresponding Simple Temporal Network (STN), is detected. ITSAT then adds certain clauses to the SAT formula at hand to prevent the reoccurrence of such negative cycles. This process is then

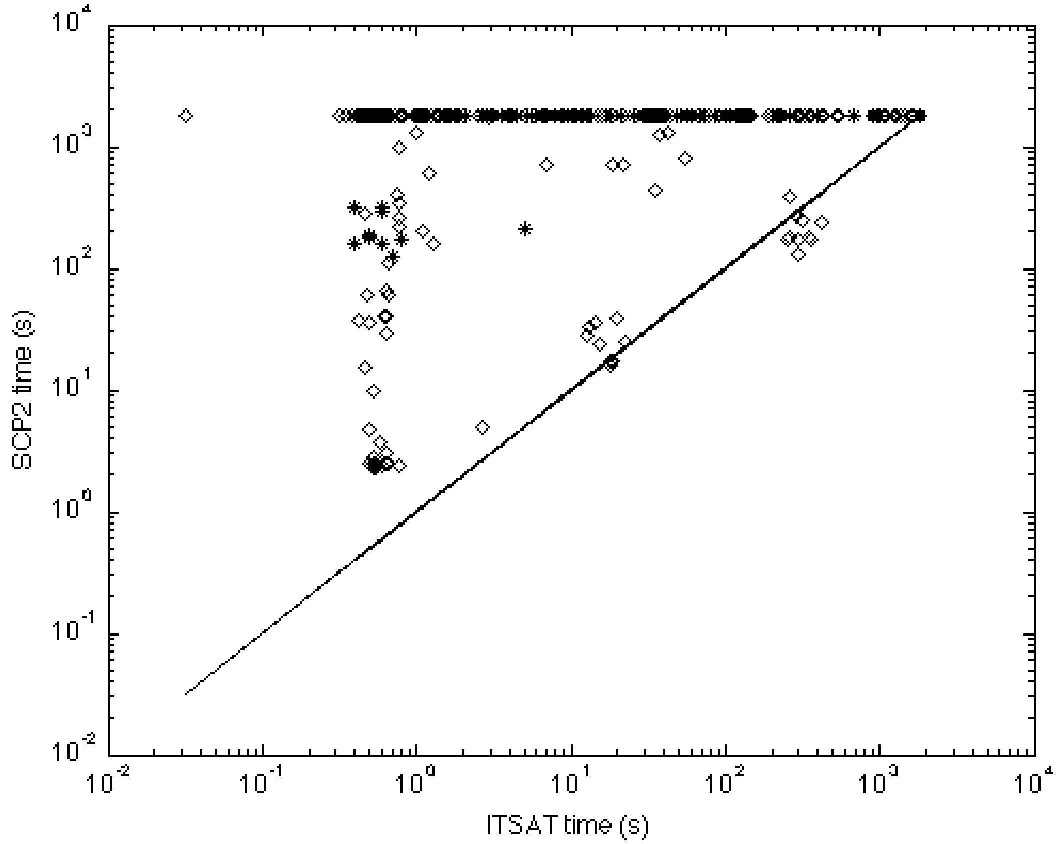

Figure 16. Speed Comparison Between ITSAT and SCP2

repeated until a temporally valid plan will be produced, or a predefined time limit will be reached.

The main contributions of this paper can be summarized as follows:

- We introduced a novel method to detect temporal actions that can be compressed into classical ones. The new compression technique is performed as a preprocessing task and thus is independent from the planning algorithm and can be used by any other temporal planner. This makes our compression technique more general than that of POPF, which is specifically tailored for that planner. Our empirical results showed that such action compression results in an improved performance for ITSAT. We also empirically showed that our method is capable of detecting more compression-safe temporal actions than the only previous action compression method, used by POPF.

- We introduced three new encoding methods based on the concept of parallel plans for SAT-based temporal planning. While two of these methods have been adopted from classical planning, our third method, which produces more compact formulae, has been employed by ITSAT for the first time. Our empirical results show that this new encoding can significantly enhance the performance of SAT-based temporal planning.

- We introduced a method to avoid producing the plans that are members of a given regular language over the set of all events. This was done by embedding the SAT encoding of a particular FSM that accepts the language into the SAT encoding of the input problem. We used this method for preventing the temporal inconsistencies of the produced causally valid plans from reoccurring in subsequent causally valid plans. Our experiments showed that this method contributed considerably to the performance of ITSAT in current benchmark problems with required concurrency.

According to our empirical results, by taking advantage of these new approaches, not only does ITSAT outperform the state-of-the-art temporally expressive planners, it is also competitive with the most efficient temporal planners that do not handle required concurrency. Nevertheless, we believe that the performance of ITSAT can be improved in several ways, which are discussed below.

In the current version of ITSAT, the satisfiability of formulae with different number of steps are checked in a sequential manner, starting with a formula encoding one step. This means that the SAT solver has to refute several formulae until it finds the first satisfiable formula. If the time required for checking the satisfiability of formulae increased with the number of steps, this policy would result in the best performance of ITSAT. However, this is almost never the case. As it has been shown in the case of classical planning, for a fixed planning problem, the time needed for finding a model for a satisfiable formula is usually considerably less than the time needed for refuting an unsatisfiable formula (Rintanen et al., 2006). Based on our experiments, the same phenomenon happens in the case of temporal planning, too. Similar to the SAT-based classical planning, one can take advantage of this phenomenon by checking for the satisfiability of formulae with different numbers of steps in parallel. The applicability of such parallelisms can be very sensistive to the amount of memory required for saving the formulae. As it was shown in Section 6, our newly introduced $\exists^*$-step encoding is considerably more efficient than the temporal version of the classical $\exists$-step encoding in terms of the memory usage. This suggests that our $\exists^*$-step encoding is more suitable for employing such parallelism.

As in linear sized classical $\forall$-step and $\exists$-step encodings, in all our encoding methods, we assume that there exists a predefined fixed ordering on all events of a given problem. This ordering can have a great impact on the number of steps needed for solving the input problem. For example, consider a sequential plan in which no ground action is applied more than once. If this potential plan is a subsequence of the mentioned fixed ordering, then only one step will be sufficient for finding the plan. On the other hand, if in this case we reversed this fixed ordering, the number of steps that would be required to find the model might be as large as the size of the plan itself. In the current implementation of ITSAT, the ordering in which the events are produced while constructing ground actions, is taken as the predefined fixed ordering of events. However, considering the causal relationships among the actions of a given problem, one might be able to find more effective orderings that would result in a fewer steps for solving the problem. We believe that this enhancement would result in an improved version of ITSAT, which will be more efficient in terms of both speed and memory usage.

The current version of ITSAT uses off-the-shelf general-purpose SAT solvers. This means that any advancement in designing such solvers can also improve the performance of ITSAT. Recent investigations in the field of SAT-based classical planning have shown that designing a SAT solver tailored for solving planning problems can result in a much improved performance of SAT-based planners. In particular, the most efficient SAT-based classical planner, M_p (Rintanen, 2012), has been able to be competitive with the sate-of-the-art state-space based planners by employing this idea. Since the causal structures of temporal planning problems are generally very similar to that of classical planning problems, we believe that ITSAT can benefit enormously from employing a planning-oriented SAT solver.

As mentioned in Section 6, one of the main drawbacks of ITSAT is the poor quality of its produced plans in some of the benchmark domains. This is mainly due to the fact that ITSAT abstracts out the duration of actions, and thus, the SAT solver does not have the needed resources for evaluating the quality of the plans that are being produced. Alternatively, one can add an explicit representation of the time into the encoding (Shin & Davis, 2005). This can be done by using SMT solvers (Armando & Giunchiglia, 1993), which handle continuous variables. However, as it was discussed in Section 1, this solution may result in a considerably slower search. We think that ITSAT can benefit from a combination of these two approaches: after that the first plan is produced by ITSAT, it can proceed by introducing appropriate numerical constraint to the SAT formula at hand, and then use an SMT solver to produce improved plans. This is the subject of our ongoing research.

Finally, we should mention that some of the components of ITSAT, can also be used in other fields of AI planning. Most notably, our $\exists^*$-step encoding can also be employed in SAT-based classical planning. Our empirical results show that this encoding method can be quite effective in reducing the number of steps needed to produce valid plans in several temporal planning domains that also have a classical version. We think that the improved performance of the $\exists^*$-step encoding in comparison to the $\exists$-step encoding can be achieved in classical planning, too. Moreover, in Section 5, we showed how to prevent members of any given regular language on the events of the input problem from being produced as the output plan. We used this method to prevent ITSAT from producing temporally invalid plans. The same method can be employed to enforce a variety of constraints on the plan that is being produced. For example, consider the case where we require certain actions to be executed only in a specific order. It must be clear that the set of all plans violating this constraint can be regarded as a regular language over the set of all actions. Therefore, such a constraint can be introduced to the encoding of the problem by the same method discussed in Section 5.

Acknowledgments

The authors would like to thank the handling editor, Jörg Hoffmann, and the anonymous reviewers for their invaluable contributions to the quality of this paper.

Appendix A. Proofs

Theorem 1. Let $\mathcal{P} = (I, G, A)$ be a temporal planning problem and $\mathcal{P}^c = (state(I), G, A^c)$ be the causal abstraction of $\mathcal{P}$. Assume that $\pi = \langle e_1, ..., e_n \rangle$ is a sequence of events that is applicable in I, and $s_n = succ(I, \pi)$. Then the following conditions must hold:

- If two propositions p and q are both members of $state(s_n)$, then p and q are non-mutex in the layer n of the planning graph of $\mathcal{P}^c$.

- If proposition p is a member of $state(s_n)$, and action a is a member of $agenda(s_n)$, then p and $open_a$ are non-mutex in layer n of the planning graph of $\mathcal{P}^c$.

Proof. We give the proof by induction on n (the length of π). For $n = 0$, i.e., when no event is applied to I, the conclusions obviously hold because every member of $state(I)$ is present in the first layer of the graph, we have no mutex in the first layer, and we have $agenda(I) = \emptyset$ by Definition 6. Now suppose the conclusions hold for $n = k - 1$. We show that they will also hold for $n = k$. Assume that $\pi = \langle e_1, ..., e_k\rangle$ is a sequence of events that is applicable in I, $s_k = succ(I, \pi)$, and $s_{k-1} = succ(I, \langle e_1, ..., e_{k-1}\rangle)$.

- Let p and q be two members of $state(s_k)$. There are three possible cases:

  Case 1: if both p and q are members of s_{k-1}, then by the induction hypothesis, p and q are non-mutex in layer $k - 1$ of the planning graph of $\mathcal{P}^c$ and thus $noop_p$ and $noop_q$ are non-mutex in layer $k - 1$. Hence, p and q will not be mutex in layer k.

  Case 2: if neither p nor q are members of $state(s_k)$, then by Definition 5, p and q must be both members of $add(e_k)$. Assume that e_k is the ending event of action a (the case where e_k is the starting event of a is analogous and thus is omitted here). Since e_k is applicable in state s_{k-1}, by Definition 4, all members of $pre(e_k)$ must be also members of $state(s_{k-1})$, and a must be a member of $agenda(s_{k-1})$. Therefore, by the induction hypothesis, all members of $pre(a^e)$ are non-mutex in layer $k - 1$. As a result, p and q, which based on Definition 10 are both added by a^e, will be non-mutex in layer k.

  Case 3: if only p is a member of $state(s_{k-1})$, then by Definition 5, q must be a member of $add(e_k)$, and p cannot be a member $del(e_k)$. Assume that e_k is the ending event of action a (again, the case where e_k is the starting event of a is analogous and thus is omitted here). By the induction hypothesis, all members of $pre(a^e)$ are non-mutex in layer $k - 1$, and by Definition 10, a^e does not delete p. Therefore, a^e will be present in layer $k - 1$ and it cannot be mutex with $noop_p$. As a result, p and q are non-mutex in layer k.

- Let p be a member of $state(s_k)$, and action a be a member of $agenda(s_k)$. There are two possible cases:

  Case 1: if a is also a member of $agenda(s_{k-1})$, then a is started but not yet ended before reaching s_{k-1}, and all invariants of a have to be members of $state(s_{k-1})$. By the induction hypothesis, these invariants must be non-mutex in layer $k-1$. Hence, a^i is present in layer $k - 1$. By Definition 10, a^i adds $open_a$. Now, we must show that p can be added by an action that is not mutex with a^i in layer $k-1$. If p is a member of $state(s_{k-1})$, then by the induction hypothesis, p is present in layer $k - 1$. Therefore, $noop_p$, which is not mutex with a^i, is applicable in layer $k - 1$ and, as a result, p and $open_a$ are not mutex in layer k. On the other hand, if p is not a member of $state(s_{k-1})$, it must be added by e_k. Since e_k is applicable in s_{k-1} and a is a member of $agenda(s_{k-1})$, by Definition 4, e_k cannot delete any invariant of a. Assume that e_k is the ending event of action b (the case where e_k the starting event of b is analogous and thus is omitted here). By the induction hypothesis, b^e, which is not mutex with

a^i must be applicable in layer $k-1$. This means that p and $open_a$ will not be mutex in layer k.

Case 2: if a is not a member of $agenda(s_{k-1})$, then by Definition 5, e_k must be the starting event of a, and a does not delete p. Moreover, by Definition 4, all starting and invariants of a must be present in $state(s_{k-1})$. Therefore, by the induction hypothesis, a^s is applicable in layer $k-1$. If p is not present in $state(s_{k-1})$, then it must be added by e_k. By Definition 10, all propositions added by the starting event of a are also added by a^s. Since a^s also adds $open_a$, then p and $open_a$ cannot be mutex in layer k. On the other hand, if p is a member of $state(s_{k-1})$, then by the induction hypothesis, p is present in layer $k-1$. Therefore, $noop_p$, which is not mutex with a^s, is applicable in layer $k-1$. This means that p and $open_a$ will not be mutex in layer k.

□

Theorem 2. Let $\mathcal{P} = (I, G, A)$ be a solvable temporal planning problem. Let A' be the set of every member of A that is either compressible towards its start or compressible towards its end. A' is compression-safe for $\mathcal{P}$.

Proof. Let π_0 be a causally valid plan for $\mathcal{P}$ (Such a plan must exist because $\mathcal{P}$ is solvable). Starting from π_0, we produce a sequence of causally valid plans by swapping the events that are next to each other in the plan at hand. Assume an arbitrary order $\langle a_1, ..., a_n \rangle$ on all members of A'. Without loss of generality, we assume that no action is repeated in π_0 (otherwise, different names can be given to the different occurrences of the same action to eliminate such a repetition). For producing the causally valid plan π_i, we consider the causally valid plan π_{i-1}. If a_i is compressible towards its start, we keep swapping the ending event of a_i with its previous event in π_{i-1} until that previous event becomes the starting event of a_i. In fact, doing these swaps collectively cause a_i to become compressed towards its start. Doing such swaps can never falsify the causally valid plan at hand: assume that e is the event immediately prior to the ending event of a_i in a causally valid plan, and e is not the starting event of a_i. Then each precondition and effect of e must be present in at least one state whose agenda includes a_i. Thus, by Theorem 1, no precondition or effect of e can be mutex with $open_{a_i}$ in the last layer of the levelled-off planning graph of the causal abstraction of $\mathcal{P}$. Then by Definition 13, e must be swappable with the ending event a_i. Therefore, π_i is a causally valid plan in which the starting and ending events of a_i are located next to each other. Similarly, if a_i is compressible towards its end, we keep swapping the starting event of a_i with its next event in π_{i-1} until that next event becomes the ending event of a_i. As a result, in π_n, the ending event of all members of A' are located next to their corresponding starting events, and therefore, according to Definition 11, A' is compression-safe for $\mathcal{P}$. □

Lemma 1. Let $S = \{e_1, ..., e_n\}$ be a set of events, E and R, be two subsets of S. Assume that S' is a subset of S such that if $e_i \in S' \cap E$, we have $e_j \notin S' \cap R$ for all $j > i$. Let M be the function that is defined by the following rules and assigns a value of $true$ or $false$ to each SAT variable of $chain(e_1, ..., e_n; E; R; k; m)$:

- For each i, $M(e_i^k) = true$ if and only if e_i is a member of S'.
- For each i, $M(b_{i,m}^k) = true$ if and only if there is $e_j \in S' \cap E$ such that $j < i$.

Then M satisfies $chain(e_1, ..., e_n; E; R; k; m)$.

Proof. We show that M satisfies formulae (C-1) to (C-3), and therefore it satisfies $chain(e_1, ..., e_n; E; R; k; m)$.

(C-1) Consider an arbitrary formula $e_i^k \rightarrow b_{j,m}^k$ from the formula (C-1). If $e_i \notin S'$ then $M(e_i^k) = false$, and thus the formula is trivially satisfied. Now consider the case where $e_i \in S'$. By the definition of formula (C-1), we know that $i < j$ and $e_i \in E$. Therefore, according to the definition of M, we must have $M(b_{j,m}^k) = true$, and thus again the formula is satisfied.

(C-2) Consider an arbitrary formula $b_{i,m}^k \rightarrow b_{j,m}^k$ from the formula (C-2). If $M(b_{i,m}^k) = false$, the formula is trivially satisfied. On the other hand, if $M(b_{i,m}^k) = true$, then there must exist an l, such that $l < i$ and $e_l \in S' \cap E$. Since $i < j$, we must have $l < j$, and therefore, we have $M(b_{j,m}^k) = true$, and hence the formula is satisfied.

(C-3) Consider an arbitrary formula $b_{i,m}^k \rightarrow \neg e_i^k$ from the formula (C-3). If $M(b_{i,m}^k) = false$, the formula is trivially satisfied. On the other hand, if $M(b_{i,m}^k) = true$, then there must exist an l, such that $l < i$ and $e_l \in S' \cap E$. Then according to the properties of S', we must have $e_i \notin S' \cap R$. However, by the definition of $chain(e_1, ..., e_n; E; R; t; m)$, we have $e_i \in R$ and, as a result, $e_i \notin S'$. Therefore, $M(e_i^k) = false$ and the formula is satisfied.

□

Lemma 2. Let $S = \{e_1, ..., e_n\}$ be a set of events, and E and R be two subsets of S. Assume that M is a model for $chain(e_1, ..., e_n; E; R; k; m)$. If $e_i \in E$ and $M(e_i^k) = true$, then for all $j > i$ such that $e_j \in R$, we have $M(b_{j,m}^k) = true$, and consequently $M(e_j^k) = false$.

Proof. Suppose that in sequence $e_1, ..., e_n$, event e_j is the first event after e_i such that $e_j \in R$. Since in the formula (C-1) of $chain(e_1, ..., e_n; E; R; k; m)$, we have $e_i^k \rightarrow b_{j,m}^k$, then $M(b_{j,m}^k) = true$. Similarly, if in sequence $e_1, ..., e_n$, event $e_{j'}$ is the first event after e_j such that $e_{j'} \in R$, then we must have the formula $b_{j,m}^k \rightarrow b_{j',m}^k$ in (C-2), which implies that $M(b_{j',m}^k) = true$. By repeating the argument of the latter case, we infer that for all $j > i$ such that $e_j \in R$, we have $M(b_{j,m}^k) = true$, and according to (C-3), we have $M(e_j^k) = false$. □

Theorem 3 (completeness of temporal ∀-step encoding). Let $\mathcal{P} = (I, G, A)$ be a solvable temporal planning problem, $\{e_1, ..., e_n\}$ be the set of all events of $\mathcal{P}$, and $\pi = \langle Step_1, ..., Step_l \rangle$ be a causally valid ∀-step plan for $\mathcal{P}$. There exists a model M for $\phi_l^\forall$ such that $\pi = plan(M)$.

Proof. We construct a function M that assigns $true$ or $false$ to all SAT variables of the formula $\phi_l^{\forall}$. Let $\langle s_0, ..., s_l \rangle$ be the state transition sequence of π. M is defined by the following rules:

- For each proposition p, and each k such that $0 \leq k \leq l$, $M(p^k) = true$ iff p is a member of $state(s_k)$.
- For each action $a \in A$, and each k such that $0 \leq k \leq l$, $M(a^k) = true$ iff a is a member of $agenda(s_k)$.
- For each each i such that $1 \leq i \leq n$, and each k such that $1 \leq k \leq l$, $M(e_i^k) = true$ iff e_i is a member of $Step_k$. Moreover, for each k such that $1 \leq k \leq l$, $M(e_0^k) = M(e_{n+1}^k) = false$.
- For each proposition p, each i such that $0 \leq i \leq n+1$, and each k such that $0 \leq k \leq l$, $M(b^k_{i,m_1^p}) = true$ iff there exists and event e_j, such that $j < i$, $e_j \in E_p^-$, and $e_j \in Step_k$.
- For each proposition p, each i such that $0 \leq i \leq n+1$, and each k such that $0 \leq k \leq l$, $M(b^k_{i,m_2^p}) = true$ iff there exists and event e_j, such that $j > i$, $e_j \in E_p^-$, and $e_j \in Step_k$.

We show that M satisfies all formulae (∀-1) to (∀-13), and therefore is a model for $\phi_l^{\forall}$. Note that by the way M is constructed, we directly have $\pi = plan(M)$.

(∀-1) According to Definition 15, we have $s_0 = I$, and thus formula (∀-1) is clearly satisfied.

(∀-2) According to Definition 15, we have $G \subseteq state(s_l)$, and thus formula (∀-2) is clearly satisfied.

(∀-3) According to Definition 6, we have $agenda(I) = \emptyset$, and thus formula (∀-3) is clearly satisfied.

(∀-4) According to Definition 15, we have $agenda(s_l) = \emptyset$, and thus formula (∀-4) is clearly satisfied.

(∀-5) Let p be an arbitrary proposition, and e be an event such that $e \in R_p$. If $e \notin Step_k$, then $M(e^k) = false$ and, therefore, formula (∀-5) is trivially satisfied. Consider the case where $e \in Step_k$. According to Definition 15, $Step_k$ must be a ∀-step from s_{k-1} to s_k. Thus, by Definition 14, for any possible ordering on the events of $Step_k$, we must be able to execute these events according to the ordering, starting from state s_{k-1}. One possible ordering can be an specific ordering that puts e in the front of all other events. Therefore, e must be applicable to s_{k-1}. Then, by Definition 4, we have $p \in state(s_{k-1})$. This implies that $M(p^{k-1}) = true$, from which the satisfaction of formula (∀-5) easily follows.

(∀-6) Let p be an arbitrary proposition, and e be an event such that $e \in E_p^+$. If $e \notin Step_k$, then $M(e^k) = false$ and, therefore, formula (∀-6) is trivially satisfied. Consider the case where $e \in Step_k$. Similar to the previous case, starting from state s_{k-1}, we must be able to execute the events of $Step_k$ by any possible ordering and reach s_k.

One such possible ordering is one that puts e after all other events. Therefore, add-effects of e must be members of $state(s_j)$, and by Definition 5, we have $p \in state(s_k)$. This implies that $M(p^k) = true$, from which the satisfaction of formula ($\forall$-6) easily follows.

($\forall$-7) Let p be an arbitrary proposition, and e be an event such that $e \in E_p^-$. If $e \notin Step_k$ then $M(e^k) = false$ and, therefore, formula ($\forall$-7) is trivially satisfied. Otherwise, by the same argument given case ($\forall$-6), we have $p \notin state(s_k)$, and the satisfaction of formula ($\forall$-7) easily follows.

($\forall$-8) Let p be an arbitrary proposition. Consider the nontrivial case where p is a member of $state(s_k)$ but not $state(s_{k-1})$. It can be easily derived from Definition 5, that if p is added by the application of a sequence of events, then at least one of those events must add p. This implies the satisfaction of formula ($\forall$-8).

($\forall$-9) This case is analogous to case ($\forall$-8)

($\forall$-10) Lemma 1 is used to prove that M satisfies $chain(e_1, ..., e_{n+1}; E_p^-; R_p \cup \{e_{n+1}\}; k; m_1^p)$. It is straightforward to see that M has all the properties of the model M in Lemma 1. We only need to show that for any proposition p, provided that $e_i \in Step_k \cap E_p^-$ and $j > i$, then $e_j \notin Step_k \cap R_p$. Suppose that $e_j \in Step_k$. By Definition 14, for any possible ordering on the events of $Step_k$, we must be able to execute those events according to the ordering, starting from the state s_{k-1}. One such possible ordering can be one that puts e_i immediately before e_j. Notice that e_i deletes p, because $e_i \in E_p^-$. Thus, e_j cannot have p as its precondition, and $e_j \notin R_p$. We can infer that $e_j \notin Step_k \cap R_p$. Therefore, by Lemma 1, M satisfies $chain(e_1, ..., e_{n+1}; E_p; R_p \cup \{e_{n+1}\}; k; m_1^p)$. Now, we show that M also satisfies $b^k_{n+1,m_1^p} \to \neg a^k$. Consider the nontrivial case, where $M(b^k_{n+1,m_1^p}) = true$. Based on the way we construct M, at least one of $e_1, ..., e_n$, say e_j, must delete p. Again, since $Step_k$ is a $\forall$-step from s_{k-1} to s_k, we must be able to execute the events of $Step_k$ by any possible ordering and reach s_k. Consider an specific ordering that puts e_j after all other events. Now, if $a \in agenda(s_k)$, according to Definition 5, e_j cannot be the ending event of a. Therefore, by Definition 4, a is also a member of the agenda of the state to which e_j is applied. This clearly contradicts the applicability of e_j, because e_j deletes p, which is an invariant of a. Therefore, $M(a^k) = false$, which implies that M satisfies $b^k_{n+1,m_1^p} \to \neg a^k$.

($\forall$-11) In this case, too, Lemma 1 is used to prove that M satisfies $chain(e_n, ..., e_0; E_p; R_p \cup \{e_0\}; k; m_1^p)$. Similar to the previous case, it is straightforward to confirm that M has all the properties of Lemma 1. Since the ordering in $chain(e_1, ..., e_{n+1}; E_p^-; R_p \cup \{e_{n+1}\}; k; m_1^p)$ is reversed in $chain(e_n, ..., e_0; E_p^-; R_p \cup \{e_0\}; k; m_1^p)$, we need to show that for any proposition p, provided that $e_i \in Step_k \cap E_p^-$ and $j < i$, then $e_j \notin Step_k \cap R_p$. Suppose that $e_j \in Step_k$. By Definition 14, for any possible ordering on the events of $Step_k$, we must be able to execute these events according to the ordering, starting from state s_{k-1}. One such possible ordering can be one that puts e_i immediately before e_j. By the same argument as the one given in case

($\forall$-10), we infer that $e_j \notin Step_k \cap R_p$ and, therefore, by Lemma 1, M satisfies $chain(e_n, ..., e_0; E_p^-; R_p \cup \{e_0\}; k; m_1^p)$. Now, we show that M also satisfies $b_{0,m_2^p}^k \rightarrow \neg a^{k-1}$. Consider the nontrivial case, where $M(b_{0,m_2^p}^k) = true$. Based on the way we construct M, at least one of $e_1, ..., e_n$, say e_j, must delete p. Again, because $Step_k$ is a $\forall$-step from s_{k-1} to s_k, starting from the state s_{k-1}, we must be able to execute the events of $Step_k$ by any possible ordering. Consider the specific ordering that puts e_j before all other events. By Definition 4, since e_j deletes p, which is an invariant of a, a cannot be a member of $agenda(s_{k-1})$. Therefore, $M(a^k) = false$, which implies that M satisfies $b_{0,m_2^p}^k \rightarrow \neg a^{k-1}$.

($\forall$-12) Assume that e is the starting event of action a. Because $Step_k$ is a $\forall$-step from s_{k-1} to s_k, starting from the state s_{k-1}, we must be able to execute the events of $Step_k$ by any possible ordering. Consider the specific ordering that puts e before all other events. Therefore, e must be applicable to s_{k-1}. Then, by Definition 4, a cannot be a member of $agenda(s_{k-1})$, and thus $M(a^{k-1}) = false$. To see that $M(a^k) = true$, consider the specific ordering of events that puts e after all other events. If the events are executed by this ordering, the results of applying e will appear in state s_k, and thus a must be a member of $agenda(s_k)$. Hence, M satisfies $e^k \rightarrow \neg a^{k-1} \wedge a^k$.

($\forall$-13) Analogous to case ($\forall$-12), assume that e is the ending event of action a. Since $Step_k$ is a $\forall$-step from s_{k-1} to s_k, starting from state s_{k-1}, we must be able to execute the events of $Step_k$ by any possible ordering. Consider a specific ordering that puts e before all other events. Therefore, e must be applicable to s_{k-1}. Then, by Definition 4, a must be a member of $agenda(s_{k-1})$, and thus $M(a^{k-1}) = true$. To see that $M(a^k) = false$, consider the specific ordering of events that puts e after all other events. If the events are executed by this ordering, the results of applying e will appear in state s_k, and thus a cannot be a member of $agenda(s_k)$. We conclude that M satisfies $e^k \rightarrow a^{k-1} \wedge \neg a^k$.

□

Theorem 4 (soundness of $\forall$-step encoding). Let $\mathcal{P} = (I, G, A)$ be a temporal planning problem, $\{e_1, ..., e_n\}$ be the set of all events of $\mathcal{P}$, and $\phi_l^{\forall}$ be the $\forall$-step encoding for $\mathcal{P}$. If $\phi_l^{\forall}$ has a model M, then $plan(M)$ is a causally valid $\forall$-step plan for $\mathcal{P}$.

Proof. We can obtain $plan(M)$ as follows. For each k such that $1 \leq k \leq l$, let $Step_k$ be the set of all events e for which $M(e^k) = true$. For each k such that $0 \leq k \leq l$, let s_k be a temporal state. Assume that $state(s_k)$ is the set of all propositions p for which $M(p^k) = true$, and $agenda(s_k)$ is the set of all actions a for which $M(a^k) = true$. We now show that $\pi = plan(M) = \langle Step_1, ..., Step_l \rangle$ is a causally valid $\forall$-step plan for $\mathcal{P}$ with the state transition sequence $\langle s_0, ..., s_l \rangle$.
From formula ($\forall$-1), it immediately follows that $I = state(s_0)$. Formula ($\forall$-2) implies that $G \subseteq state(s_l)$. Formulae ($\forall$-3) and ($\forall$-4) respectively imply that $agenda(s_0)$ and $agenda(s_n)$ are empty sets. Now, we only need to show that for each k such that $1 \leq k \leq l$, $Step_k = \{e'_1, ..., e'_m\} \subseteq \{e_1, ..., e_n\}$ is a $\forall$-step from s_{k-1} to s_k. We first show that for any

proposition p, $Step_k$ cannot include two different events e_i and e_j such that $e_i \in R_p$, and $e_j \in E_p^-$. If $j < i$, since M satisfies formula ($\forall$-10), by Lemma 2, we can infer $M(e_i^k)$ and $M(e_j^k)$ cannot be both equal to $true$. On the other hand, if $i < j$, since M satisfies formula ($\forall$-11), again by Lemma 2, we can infer $M(e_i^k)$ and $M(e_j^k)$ cannot be both equal to $true$. Thus, e_i and e_j cannot be both members of $Step_k$. Let $O : \{1, ..., m\} \to \{1, ..., m\}$ be an arbitrary ordering function. We now show by induction on k', where $k' \leq m$, the sequence $\langle e'_{O(1)}, ..., e'_{O(k')}\rangle$ is applicable to s_{k-1}. For $k' = 0$ (i.e., the case where no event is to be applied to s_{k-1}), the conclusion trivially holds. As the induction hypothesis, let $s'_{k'}$ be the temporal state resulting from applying the sequence $\langle e'_{O(1)}, ..., e'_{O(k')}\rangle$ to s_{k-1}. Let $e'_{O(k'+1)}$ be the starting event of action a (we omit the very similar case where $e'_{O(k'+1)}$ is the ending event of a). We show that conditions (1) to (3) of Definition 4 hold and thereby $e'_{O(k'+1)}$ is applicable to $s'_{k'}$. As a result, $\langle e'_{O(1)}, ..., e'_{O(k'+1)}\rangle$ will be applicable to s_{k-1}.

(1) From formula ($\forall$-5), it easily follows that all preconditions of $e'_{O(k'+1)}$ and all invariants of a (except for those invariants of a that are added by $e'_{O(k'+1)}$) are members if $state(s_{k-1})$. As mentioned above, neither of these propositions can be deleted by another member of $Step_k$. Thus, these propositions are also members of $state(s'_{k'})$.

(2) From formula ($\forall$-7), it easily follows that a is not a member $agenda(s_{k-1})$. Notice that according to Definition 5, the starting event of a, i.e., $e'_{O(k'+1)}$, is the only event that can add a to the agenda of any state. Therefore, a cannot be a member of $agenda(s'_{k'})$, either.

(3) Let a' be any action with an invariant p such that $p \in del(e'_{O(k'+1)})$. Clearly we have $start(a') \in R_p$, and thus, as it was argued above, $start(a')$ and $e'_{O(k+1)}$ cannot be both members of $Step_k$. Hence, $start(a') \notin Step_k$. On the other hand, since p is deleted in step k, and M satisfies $chain(e_n, ..., e_0; E_p; R_p \cup \{e_0\}; k; m_2^p)$, then according to Lemma 2, we have: $M(b_{0,m_2^p}) = true$. Therefore, by formula ($\forall$-11), we can infer that $M(a^{k-1}) = false$, and thus, $a' \notin agenda(s_{k-1})$. $start(a') \notin Step_k$ and $a' \notin agenda(s_{k-1})$ jointly imply that $a' \notin agenda(s'_{k'})$.

We now show that s'_m, which is the result of applying $\langle e'_{O(1)}, ..., e'_{O(m)}\rangle$ to s_{k-1}, is equal to s_k.

- Let p be an arbitrary member of $state(s'_m)$. From formulae ($\forall$-6) and ($\forall$-7) it follows that if p is deleted by any member of $Step_k$, it cannot be added by any other member of $Step_k$ and thus, p cannot be a member of $state(s'_m)$. Therefore, p is not deleted by any member of $Step_k$, and the formula $\bigvee_{e \in E_p^-} e^k$ is not satisfied by M. Besides, if p is not added by any member of $Step_k$, it must be a member of $state(s_{k-1})$, and thus $M(p^{k-1}) = true$. Now, by formula ($\forall$-9), we can infer that $M(p^k) = true$, and hence $p \in state(s_k)$. On the other hand, if p is added by a member of $Step_k$, from formula ($\forall$-6), we can deduce that $M(p^k) = true$, and again we have $p \in state(s_k)$. Therefore, $state(s'_m) \subseteq state(s_k)$.

- Let p be an arbitrary member of $state(s_k)$. According to formula ($\forall$-7), p cannot be deleted by any member of $Step_k$. Besides, by formula ($\forall$-8), p is either a member of

$state(s_{k-1})$ or is added by a member of $Step_k$. In both cases, Definition 5 implies that $p \in state(s'_m)$. Therefore, $state(s_k) \subseteq state(s'_m)$.

- Let a be an arbitrary member of $agenda(s'_m)$. From formulae ($\forall$-12) and ($\forall$-13), it follows that the starting and ending events of no single action can be both members of $Step_k$. If $a \in agenda(s_{k-1})$, then because a is still open in state s'_m, we can infer that $end(a) \notin Step_k$. Therefore, according to formula ($\forall$-13), $M(a^k) = true$, and a must be a member of $agenda(s_k)$. On the other hand, if $a \notin agenda(s_{k-1})$, then $start(a)$ must be a member of $Step_k$. Then, by formula ($\forall$-12), we have: $M(a^k) = true$, and again, a must be a member of $agenda(s_k)$. Therefore, $agenda(s'_m) \subseteq agenda(s_k)$.

- Let a be an arbitrary member of $agenda(s_k)$. According to formula ($\forall$-13), $start(a)$ cannot be a member of $Step_k$. Besides, by formula ($\forall$-12), a is either a member of $agenda(s_{k-1})$ or $start(a)$ is a member of $Step_k$. In both cases, Definition 5 implies that $a \in agenda(s'_m)$. Therefore, $agenda(s_k) \subseteq agenda(s'_m)$.

Above argument shows that $state(s_k) = state(s'_m)$ and $agenda(s_k) = agenda(s'_m)$. Hence, $s_k = s'_m = succ(s_{k-1}, \langle e'_{O(1)}, ..., e'_{O(m)} \rangle)$. Therefore, $Step_k$ is a $\forall$-step from s_{k-1} to s_k. □

Theorem 5 (completeness of $\exists$-step encoding). Let $\mathcal{P} = (I, G, A)$ be a solvable temporal planning problem, $\{e_1, ..., e_n\}$ be the set of all events of $\mathcal{P}$, and $\pi = \langle Step_1, ..., Step_l \rangle$ be a causally valid $\forall$-step plan for $\mathcal{P}$. There exists a model M for $\phi_l^\exists$ such that $\pi = plan(M)$.

Proof. By Theorem 3, there exists a model M for $\phi_l^\forall$ such that $\pi = plan(M)$. We show that M can be translated into a model for $\phi_l^\exists$. Since formulae (1) to (10) are shared between $\phi_l^\forall$ and $\phi_l^\exists$, then M also satisfies all these formulae. We now show that M also satisfies all formulae ($\exists$-11) to ($\exists$-20), and therefore can be translated into a model for $\phi_l^\exists$. In the following cases, a is an arbitrary temporal action, e_i is the starting event of a, and e_j is the ending event of a.

($\exists$-11) If $M(e_i^k) = false$, then formula ($\exists$-11) is trivially satisfied. If $M(e_i^k) = true$, then by formula ($\forall$-12) we have: $M(a^{k-1}) = false$, and therefore formula ($\exists$-11) is satisfied.

($\exists$-12) If $M(e_i^k) = false$, then formula ($\exists$-12) is trivially satisfied. If $M(e_i^k) = true$, then by formula ($\forall$-12) we have: $M(a^k) = true$, and therefore formula ($\exists$-12) is satisfied.

($\exists$-13) If $M(e_j^k) = false$, then formula ($\exists$-13) is trivially satisfied. If $M(e_j^k) = true$, then by formula ($\forall$-13) we have: $M(a^k) = false$, and therefore formula ($\exists$-13) is satisfied.

($\exists$-14) If $M(e_j^k) = false$, then formula ($\exists$-14) is trivially satisfied. If $M(e_j^k) = true$, then by formula ($\forall$-13) we have: $M(a^{k-1}) = true$, and therefore formula ($\exists$-14) is satisfied.

($\exists$-15) Exactly the same as case ($\exists$-11).

($\exists$-16) Exactly the same as case ($\exists$-12).

($\exists$-17) Exactly the same as case ($\exists$-13).

($\exists$-18) Exactly the same as case ($\exists$-14).

($\exists$-19) Follows immediately from the fact that M satisfies formula ($\forall$-12).

($\exists$-20) Follows immediately from the fact that M satisfies formula ($\forall$-13).

$\square$

Theorem 6 (soundness of $\exists$-step encoding). Let $\mathcal{P} = (I, G, A)$ be a temporal planning problem, $\{e_1, ..., e_n\}$ be the set of all events of $\mathcal{P}$, and $\phi_l^{\exists}$ be the $\exists$-step encoding for $\mathcal{P}$. If $\phi_l^{\exists}$ has a model M, then $plan(M)$ is a causally valid $\exists$-step plan $\mathcal{P}$.

Proof. We can obtain $plan(M)$ as follows. For each k such that $1 \le k \le l$, let $Step_k$ be the set of all events e for which we have $M(e^k) = true$. Moreover, for each k such that $0 \le k \le l$, let s_k be a temporal state. Assume that $state(s_k)$ is the set of all propositions p such that $M(p^k) = true$ and $agenda(s_k)$ is the set of all actions a such that $M(a^k) = true$. We construct $\pi = plan(M) = \langle Step_1, ..., Step_l \rangle$ and show that π is a causally valid $\exists$-step plan for $\mathcal{P}$ with state transition sequence $\langle s_0, ..., s_l \rangle$.

From formula ($\exists$-1), it immediately follows that $I = state(s_0)$. Formula ($\exists$-2) implies that $G \subseteq state(s_l)$. Formulae ($\exists$-3) and ($\exists$-4) imply that $agenda(s_0)$ and $agenda(s_n)$ are both empty sets. Now, we only need to show that for each k such that $1 \le k \le l$, $Step_k = \{e'_1, ..., e'_m\} \subseteq \{e_1, ..., e_n\}$ is a $\exists$-step from s_{k-1} to s_k. Without loss of generality, assume that the sequence $\langle e'_1, ..., e'_m \rangle$ is ordered by the fixed ordering $\langle e_1, ..., e_n \rangle$. Note that since M satisfies formula ($\exists$-10), by Lemma 2, for any proposition p, $Step_k$ cannot include any two events e_i and e_j such that $e_i \in R_p$, $e_j \in E_p^-$, and $j < i$. By induction on k', we show that for every $k' \le m$, the sequence $\langle e'_1, ..., e'_{k'} \rangle$ is applicable to s_{k-1}. For $k' = 0$ (i.e., the case where no event is applied to s_{k-1}), the conclusion obviously holds. As the induction hypothesis, let $s'_{k'}$ be the temporal state resulting from applying the sequence $\langle e'_1, ..., e'_{k'} \rangle$ to s_{k-1}. Assume that $e'_{k'+1}$ is the starting event of action a (we omit the very similar case where $e'_{k'+1}$ is the ending event of a). We show that conditions (1) to (3) of Definition 4 hold and thereby $e'_{k'+1}$ is too applicable to $s'_{k'}$.

(1) From formula ($\exists$-5), it clearly results that all preconditions of $e'_{k'+1}$ and all invariants of a (except for those invariants of a that are added by $e'_{O(k'+1)}$) are members of $state(s_{k-1})$. As we stated before, neither of these propositions can be deleted by e'_i for $i < k'$. Thus, these propositions are also members of $state(s'_{k'})$.

(2) There are two possible cases. Consider the first case, where according to the fixed ordering $\langle e_1, ..., e_n \rangle$, the ending event of a is located after $e'_{k'+1}$. Formula ($\exists$-11) implies that a cannot be a member $agenda(s_{k-1})$. Notice that according to Definition 5, the starting event of a, i.e., $e'_{k'+1}$, is the only event that can add a to the agenda of any state. Therefore, a cannot be a member of $agenda(s'_{k'})$. In the other case, where the ending event of a is located before $e'_{k'+1}$, formula ($\exists$-15) implies that either a is not a member of $agenda(s_{k-1})$, or $end(a)$ is a member of $Step_k$. However, if $end(a)$ is a member of $Step_k$, it will certainly remove a from the agenda of its resulting state. Since $e'_{k'+1}$ is the only event that can add a to the agenda of any state, we can conclude that a cannot be a member of $agenda(s'_{k'})$. Therefore, in neither of these two cases, a is a member of $agenda(s'_{k'})$.

(3) Let a' be an action that has $p \in del(e'_{k'+1})$ as an invariant. Since p is deleted in step k, and M satisfies $chain(e_1, ..., e_n; E^-_p; R_p \cup \{e_{n+1}\}; k; m^p_1)$, then according to Lemma 2, we have $M(b_{n+1,n^p_1}) = true$. Therefore, by formula (∃-10), we have $a' \notin agenda(s_k)$. On the other hand, we clearly have $end(a') \in R_p$, and thus, as we argued before, if $end(a')$ is a member of $Step_k$, it cannot be located after $e'_{k'+1}$ in the fixed ordering $\langle e_1, ..., e_n \rangle$. Hence, if a' is not a member of $agenda(s_k)$, it cannot be a member of $agenda(s'_{k'})$, either. Therefore, we can infer that $a' \notin agenda(s'_{k'})$.

We now show that s'_m, which is the result of applying $\langle e'_1, ..., e'_m \rangle$ to s_{k-1}, is equal to s_k.

- By the same argument given in the proof of Theorem 4, we have $state(s'_m) \subseteq state(s_k)$ and $state(s_k) \subseteq state(s'_m)$, hence, $state(s'_m) = state(s_k)$.

- Let a be an arbitrary member of $agenda(s'_m)$. Let e_i and e_j be the starting and ending events of a, respectively. There are three possible cases. Consider the first case where $a \in agenda(s_{k-1})$, $e_i \notin Step_k$, and $e_j \notin Step_k$ (i.e., a is open immediately before step k and is neither started nor ended in step k). In this case, since M satisfies (∃-20), we have $M(a^k) = true$, and therefore, $a \in agenda(s_k)$. Consider the second case where $a \in agenda(s_{k-1})$, $e_i \in Step_k$, $e_j \in Step_k$, and $j < i$ (i.e., a is open immediately before step k, and is first ended and then started again in step k). In this case, since M satisfies formula (∃-16), we have $M(a^k) = true$, and therefore, $a \in agenda(s_k)$. Finally, consider the third case where $a \notin agenda(s_{k-1})$, $e_i \in Step_k$, and $e_j \notin Step_k$ (i.e., a is not open immediately before step k, and it is started but not ended in step k). In this case, if $j < i$, then M must satisfy formula (∃-16) and we have $M(a^k) = true$. On the other hand, if $i < j$, then M must satisfy formula (∃-12) and, since $M(e^k_j) = false$, we must have $M(a^k) = true$, and therefore, $a \in agenda(s_k)$. Consequently, in all these three cases, a must be a member of $agenda(s_k)$; hence $agenda(s'_m) \subseteq agenda(s_k)$.

- Let a be an arbitrary member of $agenda(s_k)$, i.e., $M(a^k) = true$. There are two possible cases. Case 1) a is not a member of $agenda(s_{k-1})$, and hence $M(a^{k-1}) = false$. By formula (∃-19), we have: $M(e^k_i) = true$. This means that a is started in step k. Now, if $j < i$, the ending event of a cannot happen after its starting event, and therefore, a must remain open after the execution of step k, i.e., $a \in agenda(s'_m)$. On the other hand, if $i < j$, by formula (∃-14) we have: $M(e^k_j) = false$. This means that a is started but not ended in step k, and therefore a must remain open after the execution of step k, i.e., $a \in agenda(s'_m)$. Now consider case 2) a is a member of $agenda(s_{k-1})$, and hence $M(a^{k-1}) = true$. If $i < j$, by formula (∃-11) we have $M(e^k_i) = false$, and by formula (∃-13) we have $M(e^k_j) = false$. This means that a is open immediately before the execution of step k, and is neither started nor ended in step k. Therefore, a must also be open after the execution of step k, i.e., $a \in agenda(s'_m)$. On the other hand, if $j < i$, since both $M(a^k)$ and $M(a^{k-1})$ are false, formulae (∃-15) and (∃-17) can be combined to form the formula $(e^k_i \leftrightarrow e^k_j)$. This means that a is ended in step k if and only if it is later started again in the same step. Therefore, again a must be open after the execution of step k, and we have $a \in agenda(s'_m)$. Therefore, $agenda(s_k) \subseteq agenda(s'_m)$.

Above arguments show that $state(s_k) = state(s'_m)$ and $agenda(s_k) = agenda(s'_m)$. Hence, $s_k = s'_m = succ(s_{k-1}, \langle e'_1, ..., e'_m \rangle)$. Therefore, for the ordering functions $O : \{1, ..., m\} \rightarrow \{1, ..., m\}$, such that $O(i) = i$, we have $s_k = succ(s_{k-1}, \langle e'_{O(1)}, ..., e'_{O(m)} \rangle)$, and thus $Step_k$ is a $\exists$-step from s_{k-1} to s_k. □

Lemma 3. Let M be a model for $chain^*(e_1, ..., e_n; E_p^+; E_p^-; R_p; k; m_p^*)$, e_j be a member of R_p, and $E_p^M = \{e | e \in E_p^+ \cup E_p^-, M(e^k) = true\}$. M have the following properties:

- If there exists no event e_i such that $e_i \in E_p^M$ and $i < j$, then $M(b^k_{j,m_p^*}) = M(p^{k-1})$.
- If there exists an event e_i such that $e_i \in E_p^+$, $i < j$, $\{e_{i+1}, ..., e_{j-1}\} \cap E_p^M = \emptyset$, and $M(e_i^k) = true$, then $M(b^k_{j,m_p^*}) = true$.
- If there exists an event e_i such that $e_i \in E_p^-$, $i < j$, $\{e_{i+1}, ..., e_{j-1}\} \cap E_p^M = \emptyset$, and $M(e_i^k) = true$, then $M(b^k_{j,m_p^*}) = false$.

Proof.

- Assume that there exists no event e_i such that $e_i \in E_p^M$ and $i < j$. Consider the case where we have $M(p^{k-1}) = true$. Let $\{e_{i'_0}, ..., e_{i'_m}\}$ be the set of all events $e_{i'}$ such that $e_{i'} \in R_p \cup E_p^+ \cup E_p^-$, and $0 \leq i' \leq j$. Without loss of generality, we can assume that $0 = i'_0 < i'_1 < ... < i'_m = j$. Since M must satisfy (C*-7), we know that $M(b^k_{i'_0,m_p^*}) = true$. Assume that for an arbitrary s, $M(b^k_{i'_s,m_p^*}) = true$. If $e_{i'_s} \in E_p^+ \cup E_p^-$, then we know that $M(e^k_{i'_s}) = false$, and by (C*-4) we have $M(b^k_{i'_{s+1},m_p^*}) = true$. On the other hand, if $e_{i'_s} \in R_p - E_p^+ \cup E_p^-$, by (C*-3), we have $M(b^k_{i'_{s+1},m_p^*}) = true$. We can infer that for each $1 \leq s \leq j$, we have $M(b^k_{i'_s,m_p^*}) = true$, and thereby $M(b^k_{j,m_p^*}) = M(p^{k-1})$. The proof for the case where we have $M(p^{k-1}) = false$ is analogous, except that instead of (C*-4), we need to use (C*-5) .
- Assume that there exists an event e_i such that $e_i \in E_p^+$, $i < j$, $\{e_{i+1}, ..., e_{j-1}\} \cap E_p^M = \emptyset$, and $M(e_i^k) = true$. Let $\{e_{i'_0}, ..., e_{i'_m}\}$ be the set of all events $e_{i'}$ such that $e_{i'} \in R_p \cup E_p^+ \cup E_p^-$, and $i \leq i' \leq j$. Without loss of generality, we can assume that $i = i'_0 < i'_1 < ... < i'_m = j$. Since M must satisfy (C*-1), we know that $M(b^k_{i'_1,m_p^*}) = true$. Assume that for an arbitrary $s \geq 1$, $M(b^k_{i'_s,m_p^*}) = true$. If $e_{i'_s} \in E_p^+ \cup E_p^-$, then we know that $M(e^k_{i'_s}) = false$, and by (C*-4), we have $M(b^k_{i'_{s+1},m_p^*}) = true$. On the other hand, if $e_{i'_s} \in R_p - E_p^+ \cup E_p^-$, by (C*-3), we have $M(b^k_{i'_{s+1},m_p^*}) = true$. We can infer that for each $1 \leq s \leq j$, we have $M(b^k_{i'_s,m_p^*}) = true$, and thereby $M(b^k_{j,m_p^*}) = true$.
- Assume that there exists an event e_i such that $e_i \in E_p^-$, $i < j$, $\{e_{i+1}, ..., e_{j-1}\} \cap E_p^M = \emptyset$, and $M(e_i^k) = true$. Let $\{e_{i'_0}, ..., e_{i'_m}\}$ be the set of all events $e_{i'}$ such that $e_{i'} \in R_p \cup E_p^+ \cup E_p^-$, and $i \leq i' \leq j$. Without loss of generality, we can assume that $i =$

$i'_0 < i'_1 < ... < i'_m = j$. Since M must satisfy (C*-1), we know that $M(b^k_{i'_1,m^*_p}) = false$. Assume that for an arbitrary $s \geq 1$, $M(b^k_{i'_s,m^*_p}) = false$. If $e_{i'_s} \in E^+_p \cup E^-_p$, then we know that $M(e^k_{i'_s}) = false$, and by (C*-5), we have $M(b^k_{i'_{s+1},m^*_p}) = false$. On the other hand, if $e_{i'_s} \in R_p - E^+_p \cup E^-_p$, by (C*-3), we have $M(b^k_{i'_{s+1},m^*_p}) = false$. We can infer that for each $1 \leq s \leq j$, we have $M(b^k_{i'_s,m^*_p}) = false$, and thereby $M(b^k_{j,m^*_p}) = false$.

□

Lemma 4. Let M be a model for $chain(e_1, ..., e_{n+1}; E^-_p; R_p; k; m^{of}_p)$. Assume that $e_i \in E^-_p$, $M(e^k_i) = true$, and $p \in inv(a)$. Let e_j and e_{j+1} be the starting event and ending event of a, respectively. M has the following properties:

- If $M(e^k_{j+1}) = true$, and $i < j$, then $M(e^k_j) = true$.
- If $M(a^k) = true$, then $M(e^k_j) = true$.

Proof. Let $\{e_{i_1}, ..., e_{i_m}\}$ be equal to the set $\{e_s | e_s \in O_p, i < s \leq n+1\}$. Without loss of generality, we can assume that $i_1 < i_2 < ... < i_m = n+1$. Since $M(e^k_i) = true$, by (C^{of}-1), we can infer that $M(b_{i_1,m^{of}_p}) = true$. For each s, such that $M(b_{i_s,m^{of}_p}) = true$, by (C^{of}-2), we can deduce that $M(b_{i_{s+1},m^{of}_p}) = true$. Therefore, we have $M(b_{n+1,m^{of}_p}) = true$. Furthermore, if $i < j$, we have $e_{j+1} \in \{e_{i_1}, ..., e_{i_m}\}$, and thus $M(b_{j+1,m^{of}_p}) = true$. Besides, if $M(e^k_{j+1}) = true$, then by (C^{of}-3), we have $M(e^k_j) = true$. On the other hand, if $M(a^k) = true$, we can infer from formula (C^{of}-4) that $M(e^k_j) = true$. □

Lemma 5. Let M be a model for $chain(e_0, ..., e_n; E^-_p; R_p; k; m^{ob}_p)$. Assume that $e_i \in E^-_p$, $M(e^k_i) = true$, and $p \in inv(a)$. Let e_j and e_{j+1} be the starting event and ending event of a, respectively. M has the following properties:

- If $M(e^k_j) = true$, and $j+1 < i$, then $M(e^k_{j+1}) = true$.
- If $M(a^{k-1}) = true$, then $M(e^k_{j+1}) = true$.

Proof. The proof is very analogous to that of Lemma 4, and thus is omitted. □

Theorem 7 (completeness of the relaxed ∃-step encoding). Let $\mathcal{P} = (I, G, A)$ be a temporal planning problem and formulae $\phi^{\exists}_l$ and $\phi^{\exists^*}_l$ be two ∃-step encodings of $\mathcal{P}$ explained in Section 4. If M is a model for $\phi^{\exists}_l$, then $\phi^{\exists^*}_l$ has a model M' such that $plan(M') = plan(M)$.

Proof. Let M be a model for $\phi^{\exists}_l$. We construct the function M' to assign a value of *true* or $false$ to each binary variable of $\phi^{\exists^*}_l$, by using the following rules:

(R-1) For $1 \leq i \leq n$ and $1 \leq k \leq l$, $M'(e^k_i) = M(e^k_i)$.

(R-2) For $1 \leq k \leq l$, $M'(e_0^k) = M'(e_{n+1}^k) = false$.

(R-3) For $0 \leq k \leq l$ and each proposition p, $M'(p^k) = M(p^k)$.

(R-4) For $0 \leq k \leq l$ and each action a, $M'(a^k) = M(a^k)$.

(R-5) For $0 \leq i \leq n+1$, $1 \leq k \leq l$, and each proposition p, if there exist $j < i$ such that $M(e_j^k) = true$ and $e_j \in E_p^+ \cup E_p^-$ then $M'(b_{i,m_p^*}^k) = M(p^k)$; otherwise, $M'(b_{i,m_p^*}^k) = M(p^{k-1})$.

(R-6) For $1 \leq i \leq n+1$, $1 \leq k \leq l$, and each proposition p, if there exist $j < i$ such that $M(e_j^k) = true$ and $e_j \in E_p^-$ then $M'(b_{i,m_p^{of}}^k) = true$; otherwise, $M'(b_{i,m_p^{of}}^k) = false$.

(R-7) For $0 \leq i \leq n$, $1 \leq k \leq l$, and each proposition p, if there exist $j > i$ such that $M(e_j^k) = true$ and $e_j \in E_p^-$ then $M'(b_{i,m_p^{ob}}^k) = true$; otherwise, $M'(b_{i,m_p^{ob}}^k) = false$.

We now show that M' satisfies all formulae ($\exists^*$-1) to ($\exists^*$-13), and therefore is a model for $\phi_l^{\exists^*}$. From above rules, it should be clear that we have $plan(M') = plan(M)$.

($\exists^*$-1) Formula ($\exists^*$-1) is exactly the same as ($\exists$-1). Besides, M and M' assign the same value to each variable of this formula.

($\exists^*$-2) Formula ($\exists^*$-2) is exactly the same as ($\exists$-2). Besides, M and M' assign the same value to each variable of this formula.

($\exists^*$-3) Formula ($\exists^*$-3) is exactly the same as ($\exists$-3). Besides, M and M' assign the same value to each variable of this formula.

($\exists^*$-4) Formula ($\exists^*$-4) is exactly the same as ($\exists$-4). Besides, M and M' assign the same value to each variable of this formula.

($\exists^*$-5) Formula ($\exists^*$-4) is the conjunction of formulae (C*-1) to (C*-8). We show that M' satisfies all formulae (C*-1) to (C*-8), and thereby, it satisfies ($\exists^*$-5).

- Consider an arbitrary formula $e_i^k \rightarrow b_{j,m_p^*}^k$ from (C*-1). If $M(e_i^k) = false$, then the formula is trivially satisfied. If $M(e_i^k) = true$, then by (R-5), we have $M'(b_{i,m_p^*}^k) = M(p^k)$. On the other hand, because M satisfies ($\exists$-6), we have $M(p^k) = true$. Therefore, $M'(b_{i,m_p^*}^k) = true$, and the formula is satisfied again.
- Consider an arbitrary formula $e_i^k \rightarrow \neg b_{j,m_p^*}^k$ from (C*-2). If $M(e_i^k) = false$, then the formula is trivially satisfied. If $M(e_i^k) = true$, then by (R-5), we have $M'(b_{i,m_p^*}^k) = M(p^k)$. On the other hand, because M satisfies ($\exists$-7), we have $M(p^k) = false$. Therefore, $M'(b_{i,m_p^*}^k) = false$, and the formula is satisfied again.
- Consider an arbitrary formula $b_{i,m_p^*}^k \leftrightarrow b_{j,m_p^*}^k$ from (C*-3). Since e_i is not a member of $E_p^+ \cup E_p^-$, and none of the events located between e_i and e_j in the fixed ordering are members of $E_p^+ \cup E_p^-$, then by (R-5), we can easily show that $M'(b_{i,m_p^*}^k) = M'(b_{j,m_p^*}^k)$. Thus, the formula is satisfied.

- Consider an arbitrary formula $b^k_{i,m^*_p} \wedge \neg e^k_i \rightarrow b^k_{j,m^*_p}$ from (C*-4). If $M(e^k_i) = true$, then the formula is trivially satisfied. If $M(e^k_i) = false$, then since none of the events located between e_i and e_j in the fixed ordering are members of $E^+_p \cup E^-_p$, then by (R-5), we can easily show that $M'(b^k_{i,m^*_p}) = M'(b^k_{j,m^*_p})$. Thus, the formula is satisfied.
- Consider an arbitrary formula $\neg b^k_{i,m^*_p} \wedge \neg e^k_i \rightarrow \neg b^k_{j,m^*_p}$ from (C*-5). By the same argument as the one given for (C*-4), we can infer that $M'(b^k_{i,m^*_p}) = M'(b^k_{j,m^*_p})$. Thus, the formula is satisfied.
- Consider an arbitrary formula $\neg b^k_{i,m^*_p} \rightarrow \neg e^k_i$ from (C*-6). If $M'(b^k_{i,m^*_p}) = true$, then the formula is trivially satisfied. If $M'(b^k_{i,m^*_p}) = false$, then there exist two possible cases. Case 1: there exists an event e_j such that $j < i$, $M(e^k_j) = true$, and $e_j \in E^+_p \cup E^-_p$. In this case, by (R-5), we have $M(p^k) = M'(b^k_{i,m^*_p}) = false$. Since M must also satisfy (∃-6), we have $e_j \notin E^+_p$, and thus $e_j \in E^-_p$. Besides, M must satisfy (∃-10), which implies $M(e^k_i) = false$. By (R-1), we have $M'(e^k_i) = M(e^k_i) = false$, and therefore, the formula is satisfied. Case 2: there does not exist any event e_j such that $j < i$, $M(e^k_j) = true$, and $e_j \in E^+_p \cup E^-_p$. In this case, by (R-5), we have $M(p^{k-1}) = M'(b^k_{i,m^*_p}) = false$. Now, since M must satisfy (∃-5), we can infer that $M(e^k_i) = false$. By (R-1), we have $M'(e^k_i) = M(e^k_i) = false$, and therefore, the formula is satisfied again.
- Consider the formula $b^k_{0,m^*_p} \leftrightarrow p^{k-1}$ from (C*-6). From (R-5), it can be easily deducted that $M'(b^k_{0,m^*_p})$ is always equal to $M(p^{k-1})$. By (R-3), we have $M'(p^{k-1}) = M(p^{k-1})$. As a result, $M'(b^k_{0,m^*_p}) = M'(p^{k-1})$, and the formula is satisfied.
- Consider the formula $b^k_{n+1,m^*_p} \leftrightarrow p^k$ from (C*-6). There are two possible cases. Case 1: there exists an event e such that $M(e^k) = true$, and $e \in E^+_p \cup E^-_p$. In this case, by (R-5), we have $M'(b_{n+1,m^*_p}) = M(p^k)$. Now, by (R-5), we have $M(p^k) = M'(p^k)$. Therefore, $M'(b_{n+1,m^*_p}) = M'(p^k)$, and the formula is satisfied. Case 2: there does not exist any event e such that $M(e^k) = true$ and $e \in E^+_p \cup E^-_p$. In this case, by (R-5), we have $M'(b_{n+1,m^*_p}) = M(p^{k-1})$. Besides, since M satisfies formulae (∃-8) and the right hand side of (∃-8) becomes $false$, the left hand side of (∃-8), i.e., $\neg p^{k-1} \wedge p^k$, has to be $false$, too. Thus, if $M(p^{k-1}) = false$, then we have $M(p^k) = false$. A similar argument about (∃-9) can show that if $M(p^{k-1}) = true$, then we have $M(p^k) = true$. Thus, $M(p^{k-1}) = M(p^k)$, and by (R-3), we have $M'(p^k) = M(p^{k-1})$. Therefore, $M'(b_{n+1,m^*_p}) = M(p^{k-1}) = M'(p^k)$, and the formula is satisfied again.

(∃*-6) We show that M' satisfies all formulae (C^{of}-1) to (C^{of}-4), and thereby, it satisfies (∃*-6).

- Consider an arbitrary formula $e^k_i \rightarrow b^k_{j,m^{of}_p}$ from (C^{of}-1). We know from (C^{of}-1) that $i < j$. If $M'(e^k_i) = false$, the formula is trivially satisfied. If $M'(e^k_i) = true$, by (R-6), we have $M'(b_{j,m^{of}_p}) = true$, and the formula is satisfied.

- Consider an arbitrary formula $b^k_{i,m_p^{of}} \rightarrow b^k_{j,m_p^{of}}$ from (C^{of}-2). If $M'(b^k_{i,m_p^{of}})$ $= false$, the formula is trivially satisfied. If $M'(b^k_{i,m_p^{of}}) = true$, by the rule (R-6), there must exist an event $e_{i'}$, such that $i' < i$, $M(e^k_{i'}) = true$, and $e_{i'} \in E_p^-$. Since we have $i < j$, we must also have $i' < j$. Now, by (R-6), we have $M'(b_{j,m_p^{of}}) = true$, and the formula is satisfied.
- Consider an arbitrary formula $b^k_{j,m_p^{of}} \wedge e^k_j \rightarrow e^k_i$ from (C^{of}-3). If $M'(b^k_{j,m_p^{of}}) =$ $false$, the formula is trivially satisfied. If $M'(b^k_{j,m_p^{of}}) = true$, by (R-6), there must exist an event $e_{j'}$ such that $j' < j$, $M(e^k_{j'}) = true$, and $e_{j'} \in E_p^-$. Now, since M satisfies (∃-10), we can infer that $M(e^k_j) = false$. By (R-1), we have $M'(e^k_j) = M(e^k_j) = false$, and therefore, the formula is satisfied.
- Consider an arbitrary formula $b^k_{n+1,m_p^{of}} \wedge a^k \rightarrow e^k_i$ from (C^{of}-4). If $M'(b^k_{n+1,m_p^{of}})$ $= false$, the formula is trivially satisfied. If $M'(b^k_{n+1,m_p^{of}}) = true$, by (R-6), there must exist an event e_j such that $M(e^k_j) = true$ and $e_j \in E_p^-$. Now, since M satisfies (∃-10), we can infer that $M(a^k) = false$. By (R-4), we have $M'(a^k) = M(a^k) = false$, and therefore, the formula is satisfied.

(∃*-7) We show that M' satisfies all formulae (C^{ob}-1) to (C^{of}-4), and thereby, it satisfies (∃*-7).

- Consider an arbitrary formula $e^k_i \rightarrow b^k_{j,m_p^{ob}}$ from (C^{ob}-1). We know from (C^{ob}-1) that $j < i$. If $M'(e^k_i) = false$, the formula is trivially satisfied. If $M'(e^k_i) =$ $true$, by (R-7), we have $M'(b_{j,m_p^{ob}}) = true$, and the formula is satisfied.
- Consider an arbitrary formula $b^k_{i,m_p^{ob}} \rightarrow b^k_{j,m_p^{ob}}$ from (C^{ob}-2). If $M'(b^k_{i,m_p^{ob}}) =$ $false$, the formula is trivially satisfied. If $M'(b^k_{i,m_p^{ob}}) = true$, by the rule (R-7), there must exist an event $e_{i'}$ such that $i < i'$, $M(e^k_{i'}) = true$, and $e_{i'} \in E_p^-$. Since we have $j < i$, we must also have $j < i'$. Now, by (R-7), we have $M'(b_{j,m_p^{ob}}) = true$, and the formula is satisfied.
- Consider an arbitrary formula $b^k_{i,m_p^{ob}} \wedge e^k_i \rightarrow e^k_j$ from (C^{ob}-3). If $M'(b^k_{i,m_p^{ob}}) =$ $false$ or $M'(e^k_i) = false$, the formula is trivially satisfied. If $M'(b^k_{i,m_p^{ob}}) = true$ and $M'(e^k_i) = true$, by (R-7), there must exist an event $e_{i'}$, such that $i < i'$, $M(e^k_{i'}) = true$, and $e_{i'} \in E_p^-$. Since M satisfies (∃-10), we can infer that $M(a^k) = false$. However, we know that $i = j - 1 < j$; thus M must satisfy (∃-12). Therefore, we have $M(e^k_j) = true$. By (R-1), we have $M'(e^k_j) = M(e^k_j) =$ $true$, and therefore, the formula is satisfied.
- Consider an arbitrary formula $b^k_{0,m_p^{ob}} \wedge a^{k-1} \rightarrow e^k_j$ from (C^{ob}-4). If $M'(b^k_{0,m_p^{ob}}) =$ $false$ or $M'(a^{k-1}) = false$, the formula is trivially satisfied. If $M'(b^k_{0,m_p^{ob}}) =$ $true$ and $M'(a^{k-1}) = true$, by (R-7), there must exist an event $e_{i'}$ such that $M(e^k_{i'}) = true$, and $e_{i'} \in E_p^-$. Since M satisfies (∃-10), we can infer that $M(a^k) = false$. However, M must satisfy (∃-20). Therefore, we have $M(e^k_j) =$

$true$. By (R-1), we have $M'(e_j^k) = M(e_j^k) = true$, and therefore, the formula is satisfied.

($\exists^*$-8) Consider an arbitrary formula $e_i^k \rightarrow \neg a^{k-1}$ from ($\exists^*$-8). Let e_j be the ending event of a. We know that $i = j - 1 < j$. Therefore, M must satisfy ($\exists$-11). ($\exists^*$-8) is exactly the same as ($\exists$-11). Besides, M and M' assign the same value to each variable of these formulae. Thus, ($\exists^*$-8) is satisfied by M'.

($\exists^*$-9) Consider an arbitrary formula $e_i^k \rightarrow a^k \vee e_j^k$ from ($\exists^*$-9). We know that $i = j-1 < j$. Therefore, M must satisfy formula ($\exists$-12). ($\exists^*$-9) is exactly the same as ($\exists$-12). Besides, M and M' assign the same value to each variable of these formulae. Thus, ($\exists^*$-9) is satisfied by M'.

($\exists^*$-10) Consider an arbitrary formula $e_j^k \rightarrow \neg a^k$ from ($\exists^*$-10). Let e_i be the starting event of a. We know that $i = j - 1 < j$. Therefore, M must satisfy ($\exists$-13). ($\exists^*$-10) is exactly the same as ($\exists$-13). Besides, M and M' assign the same value to each variable of these formulae. Thus,($\exists^*$-10) is satisfied by M'.

($\exists^*$-11) Consider an arbitrary formula $e_j^k \rightarrow a^{k-1} \vee e_i^k$ from ($\exists^*$-11). We know that $i = j - 1 < j$. Therefore, M must satisfy ($\exists$-14). ($\exists^*$-11) is exactly the same as ($\exists$-14). Besides, M and M' assign the same value to each variable of these formulae. Thus, ($\exists^*$-11) is satisfied by M'.

($\exists^*$-12) ($\exists^*$-12) is exactly the same as ($\exists$-19). Besides, M and M' assign the same value to each variable of these formulae. Thus, ($\exists^*$-12) is satisfied by M'.

($\exists^*$-13) ($\exists^*$-13) is exactly the same as ($\exists$-20). Besides, M and M' assign the same value to each variable of these formulae. Thus, ($\exists^*$-13) is satisfied by M'.

□

Theorem 8 (soundness of the relaxed $\exists$-step encoding). Let $\mathcal{P} = (I, G, A)$ be a temporal planning problem, $\{e_1, ..., e_n\}$ be the set of all events of $\mathcal{P}$, and $\phi_l^{\exists^*}$ be the relaxed $\exists$-step encoding for $\mathcal{P}$. If $\phi_l^{\exists^*}$ has a model M, then $plan(M)$ is a causally valid $\exists$-step plan $\mathcal{P}$.

Proof. We can obtain $plan(M)$ as follows. For each k such that $1 \leq k \leq l$, let $Step_k$ be the set of all events e for which we have $M(e^k) = true$. For each k such that $0 \leq k \leq l$, let s_k be a temporal state. Assume that $state(s_k)$ is the set of all propositions p such that $M(p^k) = true$, and $agenda(s_k)$ is the set of all actions a such that $M(a^k) = true$. We construct $\pi = plan(M) = \langle Step_1, ..., Step_l \rangle$ and show that π is a causally valid $\exists$-step plan for $\mathcal{P}$ with state transition sequence $\langle s_0, ..., s_l \rangle$.

From ($\exists^*$-1), it immediately follows that $I = state(s_0)$. Also, ($\exists^*$-2) implies that $G \subseteq state(s_l)$. Besides, ($\exists^*$-3) and ($\exists^*$-4) imply that $agenda(s_0)$ and $agenda(s_n)$ are empty sets, respectively. Now we only need to show that for each k such that $1 \leq k \leq l$, $Step_k = \{e_{i_1}, ..., e_{i_m}\} \subseteq \{e_1, ..., e_n\}$ is a $\exists$-step from s_{k-1} to s_k. Without loss of generality, we assume that the sequence $\langle e_{i_1}, ..., e_{i_m}\rangle$ is ordered according to the fixed ordering $\langle e_1, ..., e_n\rangle$, i.e., $i_1 < i_2 < ... < i_m$.

By induction on k', we can conclude that for each $k' \leq m$, the sequence $\langle e_{i_1}, ..., e_{i_{k'}}\rangle$ is applicable to s_{k-1}. For $k' = 0$ (i.e., the case, where no event is applied to s_{k-1}), the conclusion obviously holds. Let $s'_{k'}$ be the temporal state resulting from applying $\langle e_{i_1}, ..., e_{i_{k'}}\rangle$ to s_{k-1}. Assume that $e_{i_{k'+1}}$ is the starting event of action a. We omit the very similar case where $e_{i_{k'+1}}$ is an ending event of a. We show that conditions (1) to (3) of Definition 4 holds and thereby $e_{i_{k'+1}}$ is applicable to $s'_{k'}$.

(1) Assume that $p \notin state(s'_{k'})$, where p is either a precondition of $e_{i_{k'+1}}$ or an invariant of a that is not added by $e_{i_{k'+1}}$. There are two possible cases. Case 1: p is not a member of $state(s_{k-1})$, and is not added or deleted by any member of $\{e_{i_1}, ..., e_{i_{k'}}\}$. In this case, we have $M(p^{k-1}) = false$. Moreover, there exists no event e_i such that $e_i \in E^+_p \cup E^-_p$, $i < k' + 1$, and $M(e^k_i) = true$. Case 2: p is deleted by an event $e_i \in Step_k$ and is not added or deleted by any event $e_j \in Step_k$, such that $i < j \leq k'$. In this case, we have $e_i \in E^-_p$, $i < i_{k'+1}$, $\{e_{i+1}, ..., e_{i_{k'+1}}\} \cap E^M_p = \emptyset$, and $M(e^k_i) = true$, where $E^M_p = \{e | e \in E^+_p \cup E^-_p, M(e^k) = true\}$. In both cases, by Lemma 3, we have $M(b^k_{i_{k'+1}, m^*_p}) = false$, which contradicts the fact that M satisfies (C*-6).

(2) Since M satisfies ($\exists$*-8), a is not a member of $agenda(s_{k-1})$. However, $e_{i_{k'+1}}$ is the only event in $Step_k$ that can add a to the agenda of any state. Thus, a is not a member of $agenda(s'_{k'})$.

(3) Let b be any action other than a, with an invariant $p \in del(e_{i_{k'+1}})$. Let e_j and e_{j+1} be the starting and ending events of b, respectively. As mentioned earlier, we assume that the ending event of each action is located immediately after its starting event in the fixed ordering. We show that b cannot be a member of $agenda(s'_{k'})$. There are two possible cases in which b may be a member of $agenda(s'_{k'})$. Case 1: b is an open action immediately before the execution of $Step_k$; and b it is not ended in $Step_k$ until $e_{i_{k'+1}}$ is executed. In this case, we have $M(b^{k-1}) = true$, and $M(e^k_{i_{k'+1}}) = true$. Since M satisfies ($\exists$*-7), by Lemma 5, we have $M(e^k_{j+1}) = true$. As we just assumed that b is not ended until the execution of $e_{i_{k'+1}}$, we have $i_{k'+1} < j + 1$. On the other hand, since e_j is the starting event of b, we have $i_{k'+1} \neq j$, and thus, $i_{k'+1} < j$. Therefore, by Lemma 4, we have $M(e^k_j) = true$. This contradicts the fact that M satisfies ($\exists$*-8), because here we have $M(b^{k-1}) = true$, $M(e^k_j) = true$, and $e_j = start(b)$. Case 2: b is started in step k, and it is not ended during step k until the execution of $e_{i_{k'+1}}$. In this case, we have $M(e^k_j) = true$, $M(e^k_{j+1}) = false$, $j + 1 < i_{k'+1}$, and $M(e^k_{i_{k'+1}}) = true$. Since M satisfies ($\exists$*-7), by Lemma 5, we must have $M(e^k_{j+1}) = true$, which again is a contradiction.

We now show that s'_m, which is the result of applying $\langle e_{i_1}, ..., e_{i_m}\rangle$ to s_{k-1}, is equal to s_k.

- Let p be an arbitrary proposition. If $p \in state(s'_m)$, there are two possible cases. Case 1: p is a member of $state(s_{k-1})$, and is not added or deleted by any member of $\{e_{i_1}, ..., e_{i_m}\}$. In this case, we have $M(p^{k-1}) = true$. Moreover, there exists no event e_i such that $e_i \in E^+_p \cup E^-_p$, $i < n+1$, and $M(e^k_i) = true$. Case 2: p is added by an event

$e_i \in Step_k$ and is not added or deleted by any event $e_j \in Step_k$, such that $i < n+1$. In this case, we have $e_i \in E_p^+$, $i < n+1$, $\{e_{i+1}, ..., e_n\} \cap E_p^M = \emptyset$, and $M(e_i^k) = true$, where $E_p^M = \{e | e \in E_p^+ \cup E_p^-, M(e^k) = true\}$. In both cases, by Lemma 3, we have $M(b_{n+1,m_p^*}^k) = true$. Since M satisfies (C*-8), we have $M(p^k) = true$, and thus $p \in state(s_k)$. Therefore, $state(s'_m) \subseteq state(s_k)$.

- Let p be an arbitrary proposition. If $p \notin state(s'_m)$, there are two possible cases. Case 1: p is not a member of $state(s_{k-1})$, and is not added or deleted by any member of $\{e_{i_1}, ..., e_{i_m}\}$. In this case, we have $M(p^{k-1}) = false$. Moreover, there exists no event e_i such that $e_i \in E_p^+ \cup E_p^-$, $i < n+1$, and $M(e_i^k) = true$. Case 2: p is deleted by an event $e_i \in Step_k$ and is not added or deleted by any event $e_j \in Step_k$, such that $i < j < n+1$. In this case, we have $e_i \in E_p^-$, $i < n+1$, $\{e_{i+1}, ..., e_n\} \cap E_p^M = \emptyset$, and $M(e_i^k) = true$, where $E_p^M = \{e | e \in E_p^+ \cup E_p^-, M(e^k) = true\}$. In both cases, by Lemma 3, we have $M(b_{n+1,m_p^*}^k) = false$. Since M satisfies (C*-8), we have $M(p^k) = false$, and thus $p \notin state(s_k)$. Therefore, $state(s_k) \subseteq state(s'_m)$.

- Let a be an arbitrary action, and e_i and e_j be its starting event and ending event, respectively. If $a \in agenda(s'_m)$, since we assume that the ending event of each action is located immediately after its starting event in the fixed ordering, there are only two possible cases. Case 1: a is open immediately before step k, and is not ended during step k. In this case, we have $M(a^{k-1}) = true$ and $M(e_j^k) = false$. Since M satisfies ($\exists$*-13), we must have $M(a^k) = true$. Therefore, $a \in agenda(s_k)$. Case 2: a is started but not ended in step k. In this case, we have $M(e_i^k) = true$ and $M(e_j^k) = false$. As M satisfies ($\exists$*-9), we must have $M(a^k) = true$. Therefore, $a \in agenda(s_k)$. Since in both cases, we have $a \in agenda(s_k)$, we can infer that $agenda(s'_m) \subseteq agenda(s_k)$.

- Let a be an arbitrary action, and e_i and e_j be its starting event and ending event of a, respectively. If $a \notin agenda(s'_m)$, since we assume that ending event of each action is located immediately after its starting event in the fixed ordering, there are only two possible cases. Case 1: a is not open immediately before execution of step k, and is not started during step k. In this case, we have $M(a^{k-1}) = false$ and $M(e_i^k) = false$. Since M satisfies ($\exists$*-12), we must have $M(a^k) = false$. Therefore, $a \notin agenda(s_k)$. Case 2: a is ended in step k. In this case, we have $M(e_j^k) = true$. Since M satisfies ($\exists$*-10), we must have $M(a^k) = false$. Therefore, $a \notin agenda(s_k)$. Because in both cases, we have $a \notin agenda(s_k)$, we can infer that $agenda(s_k) \subseteq agenda(s'_m)$.

The above arguments show that $state(s_k) = state(s'_m)$ and $agenda(s_k) = agenda(s'_m)$. Hence, we have $s_k = s'_m$ and $s_k = succ(s_{k-1}, \langle e_{i_1}, ..., e_{i_m}\rangle)$. Therefore, for the ordering functions $O : \{1, ..., m\} \to \{1, ..., m\}$, where $O(i) = i$, we have $s_k = succ(s_{k-1}, \langle e_{O(i_1)}, ..., e_{O(i_m)}\rangle)$, and thus $Step_k$ is a $\exists$-step from s_{k-1} to s_k. □

Theorem 9. Let $\mathcal{P} = (I, G, A)$ be a temporal planning problem, $\pi = \langle e_1, ..., e_n\rangle$ be a causally valid plan for $\mathcal{P}$, and $\tau_\pi : \{1, ..., n\} \to \mathbb{Q}$ be a relaxed scheduling function for π. There exists a valid temporal plan for $\mathcal{P}$.

Proof. By using the *bubble sort* algorithm, we can sort the events of π in an increasing order according the values given to them by τ_π. This algorithm takes two consecutive members of a sequence, and swaps them only if the value of the first one is greater than that of the second one. It continues doing the swaps until the whole sequence is properly sorted. Let e_i and e_j be two events swapped by the *bubble sort* in any stage of the algorithm (assume that e_i is located before e_j in the sequence prior to swapping). Then, we must have $\tau_\pi(i) > \tau_\pi(j)$. Thus, according to (S-1), we know that e_i and e_j are swappable (c.f., Definition 12). As a result, if the whole sequence was a causally valid plan prior to swapping, it would also be a causally valid plan after that swapping. This means that sorting π according to the values given by τ_π will result in another causally valid plan, say π'. The plan π' obviously satisfies the two conditions of Definition 9, and therefore, (π', τ_π) is a valid temporal plan for $\mathcal{P}$. $\square$

Theorem 10. Let $\mathcal{P} = (I, G, A)$ be a solvable temporal planning problem, and $\mathcal{COM}$ be the set of every member of A that is either compressible towards its start or compressible towards its end (Definition 13). There exists a valid temporal plan (π, τ_π) for $\mathcal{P}$ such that π is a causally valid plan for $\mathcal{P}$, π is compressed with respect to $\mathcal{COM}$, and τ_π is a relaxed scheduling function for π.

Proof. Let $\pi_1 = e_1, ..., e_i, e_{i+1}, ..., e_n$ be a causally valid plan for $\mathcal{P}$ such that e_i and e_{i+1} are two swappable events. Let $\pi_2 = e_1, ..., e_{i+1}, e_i, ..., e_n$ be the result of swapping e_i and e_{i+1} in π_1. We show that if τ is a relaxed scheduling function for π_1, then it is also a relaxed scheduling function for π_2.

- Consider any two events e_j and e_k such that in π_2, e_j is located before e_k. If $j \neq i+1$ or $k \neq i$, then in π_1, e_j is definitely located before e_k, too, and therefore the property (S-1) holds for e_j and e_k. On the other hand, if $j = i+1$ and $k = i$, e_j and e_k are swappable and therefore the property (S-1) trivially holds for e_j and e_k.

- Assume that e_j is the starting event of a particular action a, and e_k is the pairing event of e_j in π_2. From Definition 8, we can easily infer that if in π_2, e_i is located between e_j and e_k, then e_i cannot be the starting or ending event of a. Similarly, if e_{i+1} is located between e_j and e_k, then e_{i+1} cannot be the starting or ending event of a. On the other hand, since e_i and e_{i+1} are swappable, we know that they cannot be both some events of the same action, and therefore, either $j \neq i+1$ or $k \neq i$. Thus, swapping e_i and e_{i+1} cannot falsify the fact that e_j and e_k are pairing events. In other words, in π_1, too, e_k is the pairing event of e_j. This implies that the property (S-2) holds for e_j and e_k.

Let $(\pi', \tau_{\pi'})$ be an arbitrary valid temporal plan for $\mathcal{P}$. Since $\tau_{\pi'}$ is a scheduling function for π', it can obviously be also regarded as a relaxed scheduling function for π'. As we showed in Section 3.2, π' can be transformed to a causally valid plan π that is compressed with respect to $\mathcal{COM}$, by doing a series of swaps, where each swapping occurs between a pair of consecutive swappable events. Therefore, $\tau_{\pi'}$ must also be a relaxed scheduling function for π, and (π, τ_π) is a valid temporal plan for $\mathcal{P}$, where τ_π is the same scheduling function as $\tau_{\pi'}$. $\square$

Theorem 11. Let $\mathcal{P} = (I, G, A)$ be a temporal planning problem, $\sum = \{e_1, ..., e_n\}$ be the set of all events of $\mathcal{P}$, ϕ_l be any of the three formulae $\phi_l^{\forall}$, $\phi_l^{\exists}$, $\phi_l^{\exists^*}$ (defined in Section 4), and π be a non-empty causally valid plan for $\mathcal{P}$ obtained by solving ϕ_l. Let $\psi = (S_\psi, \sum, T_\psi, x_0, A_\psi)$ be an FSM that accepts a subsequence $\pi' = \langle e'_1, ..., e'_m \rangle$ of π, and ϕ_l^ψ be the encoding of ψ presented by (ψ-1) to (ψ-6). There does not exist any model M for $\phi_l \wedge \phi_l^\psi$ such that $\pi = plan(M)$.

Proof. We give the proof by contradiction. Assume that there exists a model M for $\phi_l \wedge \phi_l^\psi$ such that $\pi = plan(M)$. Let $f : \{1, ..., m\} \rightarrow \{1, ..., n\}$ be a function such that for each i, $f(i)$ is equal to the index of the i-th event of π' in $\sum$. Moreover, let $g : \{1, ..., m\} \rightarrow \{1, ..., l\}$ be a function such that for each i, $g(i)$ is equal to the step number of the SAT variable in ϕ_l that corresponds to the i-th event of π'. Assume $x_0, ..., x_m$ to be a sequence of states of ψ such that for $0 < i \leq m$, we have $x_i = T_\psi(x_{i-1}, e_{f(i)})$. Since ψ accepts π', we must have $x_m \in A_\psi$. As M satisfies (ψ-5), we have $M(x_0^{f(1),g(1)}) = true$. Here, two cases can be considered. case 1: $g(2) = g(1)$. In this case, since $\pi = plan(M)$, for $f(1) < j < f(2)$, we must have $M(e_j^{g(1)}) = false$. Now, by considering $(\psi - 1)$ and $(\psi - 2)$, we can infer that $M(x_1^{g(2),f(2)}) = true$. Case 2: $g(2) > g(1)$. In this case, by considering $(\psi - 1)$ and $(\psi - 2)$, we can infer that $M(x_1^{g(1),n+1}) = true$. Then by, considering $(\psi - 4)$, we can deduce that $M(x_1^{g(1)+1,0}) = true$. By the same argument plus considering $(\psi - 3)$ we can show that $M(x_1^{g(2),f(2)}) = true$. The whole deduction can be repeated to show that $M(x_{i-1}^{g(i),f(i)}) = true$ for $1 \leq i \leq m$. Therefore, we have $M(x_{m-1}^{g(m),f(m)}) = true$. Since $x_m = T_\psi(x_{m-1}, e_{f(m)})$, by considering $(\psi - 1)$, we can infer that for j, such that $e_j \in E_m^{out} \cup E_m^{in} \cup \{e_{n+1}\}$ and $\{e_{f(m)+1}, ..., e_{j-1}\} \cap (E_m^{out} \cup E_m^{in}) = \emptyset$, we have $M(x_m^{g(m),j}) = true$. However, since $x_m \in A_\psi$, this contradicts the assumption that M satisfies $(\psi - 6)$. □

Theorem 12. Let $\mathcal{P} = (I, G, A)$ be a temporal planning problem, $\sum = \{e_1, ..., e_n\}$ be the set of all events of $\mathcal{P}$, and ϕ_l be any of the three formulae $\phi_l^{\forall}$, $\phi_l^{\exists}$, $\phi_l^{\exists^*}$ (defined in Section 4). Let M be a model that satisfies ϕ_l, and $\pi = \langle e'_1, ..., e'_m \rangle = plan(M)$. Let $\psi = (S_\psi, \sum, T_\psi, x_0, A_\psi)$ be an FSM that does not accept any subsequence of π, and ϕ_l^ψ be the encoding of ψ composed of (ψ-1) to (ψ-6). There exists a model M' for $\phi_l \wedge \phi_l^\psi$ such that $\pi = plan(M')$.

Proof. Let us introduce a total order relation $\prec$ on those SAT variables of ϕ_l that correspond to events of the input problem. For any two sat variables e_i^k and $e_{i'}^{k'}$, we have $e_i^k \prec e_{i'}^{k'}$ if and only if one of the following two conditions holds: 1) $k < k'$. 2) $k = k'$ and $i < i'$. Assume that $f : \{1, ..., m\} \rightarrow \{1, ..., n\}$ is a function such that for each i, $f(i)$ is equal to the index of the i-th event of π in $\sum$. Moreover, assume that $g : \{1, ..., m\} \rightarrow \{1, ..., l\}$ is a function such that for each k, $g(k)$ is equal to the step number of the SAT variable in ϕ_l that corresponds to the k-th event of π. Let $\pi_u^{k,i} = \langle e'_u, ..., e'_t \rangle$ denote a subsequence of π with the following properties:

- $M(e_{f(t)}^{g(t)}) = true$.

- For all i' and k' such that $e_{f(t)}^{g(t)} \prec e_{i'}^{k'} \prec e_i^k$, we have $M(e_{i'}^{k'}) = false$.

In fact, $\pi_u^{k,i}$ is a substring of π that spans from the u-th event of π to the last event of π whose corresponding SAT variable is located before e_i^k in ϕ_l. We define the model M' for $\phi_l \wedge \phi_l^{\psi}$ by the following rules:

(R-1) For each SAT variable v of ϕ_l, $M'(v) = M(v)$.

(R-2) For $1 \leq k \leq l$ and $1 \leq i \leq n$, $M'(x_0^{k,i}) = true$.

(R-3) For $1 \leq k \leq l$, $1 < i \leq n$, and $x_s \in S_\psi$, $M'(x_s^{k,i}) = true$ iff for some j, the sequence $\pi_j^{k,i}$ transforms ψ from x_0 to x_s.

From (R-1), we can infer that M' satisfies ϕ_l. We now show that M' also satisfies all formulae (ψ-1) to (ψ-6), and thereby, it satisfies ϕ_l^{ψ}.

(ψ-1) Let $e_i^k \wedge x_s^{k,i} \rightarrow x_t^{k,j}$ be an arbitrary formula from (ψ-1). If $M'(e_i^k) = false$ or $M'(x_s^{k,i}) = false$, the formula is trivially satisfied. Assume that $M'(e_i^k) = M'(x_s^{k,i}) = true$. By (R-3), for some u, the sequence $\pi_u^{k,i}$ transforms ψ from x_0 to x_s. Since $e_i^k \prec e_j^k$, by the way we defined $\pi_u^{k,j}$, we can deduce that $\pi_u^{k,j} = \pi_u^{k,i} \cdot \langle e_i \rangle \cdot \pi'$, where $(\cdot)$ denotes the concatenation operator and π' is a sequence of events from $\{e_{i+1}, ..., e_{j-1}\}$. By ($\psi$-1), we have $T_\psi(x_s, e_i) = x_t$, and therefore e_i causes ψ to transit from x_s to x_t. Besides, $\{e_{i+1}, ..., e_{j-1}\} \cap E_t^{out} = \emptyset$, and thus, no member of π' can cause ψ to transit to a state other than x_t. Therefore, $\pi_u^{k,j}$ transforms ψ from x_0 to x_t, and $M'(x_t^{k,j}) = true$. Hence, the formula is satisfied.

(ψ-2) Let $\neg e_i^k \wedge x_s^{k,i} \rightarrow x_s^{k,j}$ be an arbitrary formula from (ψ-2). If $M'(e_i^k) = true$ or $M'(x_s^{k,i}) = false$, the formula is trivially satisfied. Assume that $M'(e_i^k) = false$ and $M'(x_s^{k,i}) = true$. By (R-3), for some u, the sequence $\pi_u^{k,i}$ transforms ψ from x_0 to x_s. Since $e_i^k \prec e_j^k$, by the way we defined $\pi_u^{k,j}$, we can deduce that $\pi_u^{k,j} = \pi_u^{k,i} \cdot \pi'$, where π' is a sequence of events from $\{e_{i+1}, ..., e_{j-1}\}$. Besides, $\{e_{i+1}, ..., e_{j-1}\} \cap E_s^{out} = \emptyset$, and thus, no member of π' can cause ψ to transit to a state other than x_s. Therefore, $\pi_u^{k,j}$ transforms ψ from x_0 to x_s, and $M'(x_s^{k,j}) = true$. Hence, the formula is satisfied.

(ψ-3) Let $x_s^{k,0} \rightarrow x_s^{k,i}$ be an arbitrary formula from (ψ-3). If $M'(x_s^{k,0}) = false$, the formula is trivially satisfied. Assume that $M'(x_s^{k,0}) = true$. By (R-3), for some u, the sequence $\pi_u^{k,0}$ transforms ψ from x_0 to x_s. Since $e_0^k \prec e_i^k$, by the way we defined $\pi_u^{k,i}$, we can deduce that $\pi_u^{k,i} = \pi_u^{k,0} \cdot \pi'$, where π' is a sequence of events from $\{e_1, ..., e_{i-1}\}$. Besides, $\{e_1, ..., e_{i-1}\} \cap E_s^{out} = \emptyset$, and thus, no member of π' can cause ψ to transit to a state other than x_s. Therefore, $\pi_u^{k,i}$ transforms ψ from x_0 to x_s, and $M'(x_s^{k,i}) = true$. Hence, the formula is satisfied.

(ψ-4) Let $x_s^{k,n+1} \leftrightarrow x_s^{k+1,0}$ be an arbitrary formula from (ψ-4). By the way we defined $\pi_u^{k+1,0}$, we can deduce that $\pi_u^{k,n+1} = \pi_u^{k+1,0}$ for every u. Therefore, $M'(x_s^{k,n+1}) = M'(x_s^{k+1,0})$. Hence, the formula is satisfied.

(ψ-5) According to (R-2), any formula from (ψ-5) is directly satisfied by M', .

(ψ-6) Let $\neg x^{k,i}$ be an arbitrary formula from (ψ-6). Accorrding to our assumptions, $\pi_u^{k,i}$ cannot cause ψ to transit to any of its accepting states. Since we have $x \in A_\psi$, (R-3) implies that $M'(x^{k,i}) = false$. Hence, the formula is satisfied.

□

Theorem 13. Let $N = x_{i_1}, ..., x_{i_m}$ be a negative cycle in the STN corresponding to a causally valid plan $\pi = e_1, ..., e_n$ of a temporal problem $\mathcal{P}$, where x_{i_k} is the node corresponding to event e_{i_k} of π. Let π' be another causally valid plan for $\mathcal{P}$. If a subsequence of π' is a member of L_N (defined in Section 5), the corresponding STN of π' will also have N as a negative cycle.

Proof. Let $e_{i_1}, e'_{2,1}..., e'_{2,k_2}, e_{i_2}, ..., e_{i_{m-1}}, e'_{m,1}, ..., e'_{m,k_m}, e_{i_m}$ be a subsequence of π', where $e'_{j,1}, ..., e'_{j,k_j}$ is a string of symbols in Π_j, for $1 < j \leq m$. Consider two arbitraty events e_{i_j} and $e_{i_{j'}}$ from this sequence, such that $i_j < i_{j'}$. We show that any temporal constraints between $\tau_\pi(e_{i_j})$ and $\tau_\pi(e_{i_{j'}})$ is also present between $\tau_{\pi'}(i_j)$ and $\tau_{\pi'}(i_{j'})$.

- If we have the constraint $\tau_\pi(i_j) < \tau_\pi(i_{j'})$, then by the scheduling constraint (S-1) explained in Section 5, e_{i_j} and $e_{i_{j'}}$ are not swappable. Besides, in π', e_{i_j} is clearly located before $e_{i_{j'}}$. Consequently, we must have $\tau_{\pi'}(i_j) < \tau_{\pi'}(i_{j'})$ according to the scheduling constraint (S-1).
- If we have the constraint $\tau_\pi(i_{j'}) - \tau_\pi(i_j) = dur(a)$, then by the scheduling rule (S-2), e_{i_j} and $e_{i_{j'}}$ have to be the starting event and the ending event of a, respectively. Moreover, for $j < j'' < j'$, we have $action(e_{i_{j''}}) \neq a$. This indicates that for $j < j'' \leq j'$, a is in $O_{j''}$, and therefore $e_{i_{j'}} \notin \Pi_{j''}$. Since $e'_{j'',1}..., e'_{j'',k_{j''}}$ is a string of symbols in $\Pi_{j''}$, we conclude that in π', a is not yet ended before reaching $e_{i_{j'}}$. This means that e_{i_j} and $e_{i_{j'}}$ are pairing events in π'. Thus, by the scheduling constraint (S-2), we have $\tau_{\pi'}(i_{j'}) - \tau_{\pi'}(i_j) = dur(a)$.

This shows that any edge between x_{i_j} and $x_{i_{j'}}$ in the corresponding STN of π is also present in the corresponding STN of π', and thus the latter STN has N as its negative cycle. □

References

Allen, J. F. (1984). Towards a general theory of action and time. *Artif. Intell.*, *23*(2), 123–154.

Armando, A., & Giunchiglia, E. (1993). Embedding complex decision procedures inside an interactive theorem prover. *Ann. Math. Artif. Intell.*, *8*(3-4), 475–502.

Benton, J., Coles, A. J., & Coles, A. (2012). Temporal planning with preferences and time-dependent continuous costs. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012.*

Biere, A. (2009). P{re,i}cosat@sc'09. solver description for SAT competition 2009. In *SAT 2009 Competitive Event Booklet.*

Biere, A. (2013). Lingeling, Plingeling and Treengeling entering the sat competition 2013. In *Proceedings of SAT Competition 2013*.

Blum, A., & Furst, M. L. (1997). Fast planning through planning graph analysis. *Artif. Intell.*, *90*(1-2), 281–300.

Castellini, C., Giunchiglia, E., & Tacchella, A. (2003). SAT-based planning in complex domains: Concurrency, constraints and nondeterminism. *Artif. Intell.*, *147*(1-2), 85–117.

Coles, A. J., Coles, A., Fox, M., & Long, D. (2009). Extending the use of inference in temporal planning as forwards search. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*.

Coles, A. J., Coles, A., Fox, M., & Long, D. (2010). Forward-chaining partial-order planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010*, pp. 42–49.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms (3. ed.).* MIT Press.

Cushing, W., Kambhampati, S., Mausam, & Weld, D. S. (2007). When is temporal planning really temporal?. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pp. 1852–1859.

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

Do, M. B., & Kambhampati, S. (2003). Sapa: A multi-objective metric temporal planner. *J. Artif. Intell. Res. (JAIR)*, *20*, 155–194.

Eén, N., & Biere, A. (2005). Effective preprocessing in SAT through variable and clause elimination. In *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19-23, 2005, Proceedings*, pp. 61–75.

Ernst, M. D., Millstein, T. D., & Weld, D. S. (1997). Automatic sat-compilation of planning problems. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Japan, August 23-29, 1997, 2 Volumes*, pp. 1169–1177.

Eyerich, P., Mattmüller, R., & Röger, G. (2009). Using the context-enhanced additive heuristic for temporal and numeric planning. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*.

Fox, M., & Long, D. (2002). PDDL+: Modelling continuous time-dependent effects. In *the Third International NASA Workshop on Planning and Scheduling for Space*.

Fox, M., & Long, D. (2003). PDDL2.1: An extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)*, *20*, 61–124.

Fox, M., & Long, D. (2007). A note on concurrency and complexity in temporal planning. In *the 26th Workshop of the UK Planning and Scheduling Special Interest Group*.

Garrido, A., Fox, M., & Long, D. (2002). A temporal planning system for durative actions of PDDL2.1. In *Proceedings of the 15th Eureopean Conference on Artificial Intelligence, ECAI'2002, Lyon, France, July 2002*, pp. 586–590.

Gerevini, A., Saetti, A., & Serina, I. (2006). An approach to temporal planning and scheduling in domains with predictable exogenous events. *J. Artif. Intell. Res. (JAIR)*, *25*, 187–231.

Gerevini, A., & Schubert, L. K. (1998). Inferring state constraints for domain-independent planning. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA.*, pp. 905–912.

Halsey, K. (2004). *CRIKEY! Its Co-ordination in Temporal Planning*. Ph.D. thesis, University of Durham.

Halsey, K., Long, D., & Fox, M. (2004). Multiple relaxations in temporal planning. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, pp. 1029–1030.

Haslum, P. (2006). Improving heuristics through relaxed search - an analysis of TP4 and HSP*a in the 2004 planning competition. *J. Artif. Intell. Res. (JAIR)*, *25*, 233–267.

Haslum, P., & Geffner, H. (2000). Admissible heuristics for optimal planning. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, Breckenridge, CO, USA, April 14-17, 2000*, pp. 140–149.

Helmert, M., & Geffner, H. (2008). Unifying the causal graph and additive heuristics. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia, September 14-18, 2008*, pp. 140–147.

Hoffmann, J., Gomes, C. P., Selman, B., & Kautz, H. A. (2007). SAT encodings of state-space reachability problems in numeric domains. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pp. 1918–1923.

Hoffmann, J., & Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *J. Artif. Intell. Res. (JAIR)*, *14*, 253–302.

Huang, R., Chen, Y., & Zhang, W. (2009). An optimal temporally expressive planner: Initial results and application to P2P network optimization. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009.*

Huang, R., Chen, Y., & Zhang, W. (2012). SAS+ planning as satisfiability. *J. Artif. Intell. Res. (JAIR)*, *43*, 293–328.

Kautz, H. A., & Selman, B. (1992). Planning as satisfiability. In *ECAI*, pp. 359–363.

Kautz, H. A., & Selman, B. (1996). Pushing the envelope: Planning, propositional logic and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, August 4-8, 1996, Volume 2.*, pp. 1194–1201.

Long, D., & Fox, M. (2003). Exploiting a graphplan framework in temporal planning. In *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003), June 9-13, 2003, Trento, Italy*, pp. 52–61.

Lu, Q., Huang, R., Chen, Y., Xu, Y., Zhang, W., & Chen, G. (2013). A SAT-based approach to cost-sensitive temporally expressive planning. *ACM TIST*, *5*(1), 18.

Mali, A. D., & Liu, Y. (2006). T-satplan: a SAT-based temporal planner. *International Journal on Artificial Intelligence Tools*, *15*(5), 779–802.

Rankooh, M. F., & Ghassem-Sani, G. (2013). New encoding methods for sat-based temporal planning. In *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling, ICAPS 2013, Rome, Italy, June 10-14, 2013*.

Rintanen, J. (2006). Compact representation of sets of binary constraints. In *ECAI 2006, 17th European Conference on Artificial Intelligence, August 29 - September 1, 2006, Riva del Garda, Italy, Including Prestigious Applications of Intelligent Systems (PAIS 2006), Proceedings*, pp. 143–147.

Rintanen, J. (2007). 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*, pp. 280–287.

Rintanen, J. (2012). Planning as satisfiability: Heuristics. *Artif. Intell.*, *193*, 45–86.

Rintanen, J., & Gretton, C. O. (2013). Computing upper bounds on lengths of transition sequences. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*.

Rintanen, J., Heljanko, K., & Niemelä, I. (2006). Planning as satisfiability: parallel plans and algorithms for plan search. *Artif. Intell.*, *170*(12-13), 1031–1080.

Robinson, N., Gretton, C., Pham, D. N., & Sattar, A. (2009). Sat-based parallel planning using a split representation of actions. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*.

Robinson, N., Gretton, C., Pham, D. N., & Sattar, A. (2010). Partial weighted MaxSAT for optimal planning. In *PRICAI 2010: Trends in Artificial Intelligence, 11th Pacific Rim International Conference on Artificial Intelligence, Daegu, Korea, August 30-September 2, 2010. Proceedings*, pp. 231–243.

Shin, J.-A., & Davis, E. (2005). Processes and continuous change in a SAT-based planner. *Artif. Intell.*, *166*(1-2), 194–253.

Smith, D. E., & Weld, D. S. (1999). Temporal planning with mutual exclusion reasoning. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*, pp. 326–337.

Streeter, M. J., & Smith, S. F. (2007). Using decision procedures efficiently for optimization. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007*, pp. 312–319.

Vidal, V. (2014). Yahsp3 and Yahsp3-mt in the 8th international planning competition. In *International Planning Competition*.

Vidal, V., & Geffner, H. (2006). Branching and pruning: An optimal temporal POCL planner based on constraint programming. *Artif. Intell.*, *170*(3), 298–335.

Wehrle, M., & Rintanen, J. (2007). Planning as satisfiability with relaxed ∃-Step plans. In *AI 2007: Advances in Artificial Intelligence, 20th Australian Joint Conference on Artificial Intelligence, Gold Coast, Australia, December 2-6, 2007, Proceedings*, pp. 244–253.

Younes, H. L. S., & Simmons, R. G. (2003). VHPOP: Versatile heuristic partial order planner. *J. Artif. Intell. Res. (JAIR)*, *20*, 405–430.