

# On Temporal Planning as CSP

Amol Dattatraya Mali  
Electrical Engg. & Computer Science,  
University of Wisconsin, Milwaukee, WI 53211  
mali@miller.cs.uwm.edu, Fax: 1-414-229-2769

## Abstract

(Appears as a regular paper in the proceedings of IEEE International Conference on Tools with Artificial Intelligence (ICTAI), IEEE Computer Society, Washington D.C. Nov. 2002, pp. 75-82.)

Recent advances in constraint satisfaction and heuristic search have made it possible to solve classical planning problems significantly faster. There is an increasing amount of work on extending these advances to solving more expressive planning problems which contain metric time, quantifiers and resource quantities. One can broadly classify classical planners into two categories: (i) planners doing refinement search and (ii) planners iteratively processing a representation of finite size like a SAT encoding or planning graph or a constraint satisfaction problem (CSP). One key challenge in the development of planners casting planning as SAT or CSP is the identification of constraints which are satisfied if and only if there is a plan of  $k$  steps. This task is even more complex for planners handling metric time and/or resource quantities and/or quantifiers. In this paper we show how such a SAT encoding can be synthesized for temporal planning. This encoding contains twenty kinds of constraints. We show how this encoding can be simplified. The set of constraints we identify makes it easier to develop temporal planners casting planning as a constraint satisfaction problem other than SAT, like integer linear programming (ILP). The SAT encoding can be easily adapted to more complex cases of temporal planning like the one in which different preconditions and effects of an action may be true at different times during its execution. We also discuss two additional SAT encodings of temporal planning. The encoding schemes make it easier to exploit progress in SAT and CSP solving to solve temporal planning problems.

## 1 Introduction

Recent advances in classical planning have made it possible to solve larger planning problems than before. Classical planning is the problem of finding an executable sequence of actions that achieves partially specified goal  $G$  from completely specified initial state  $I$ , given the set of actions  $O$ . In classical planning, perception is assumed to be perfect, actions are assumed to be deterministic, and, environment is assumed to be completely observable and static. Actions are assumed to be instantaneous, so time durations of actions are not considered in classical planning. Most classical planners use STRIPS representation. In this representation, a world state is a conjunction of function-free ground literals. An action has preconditions expressed as a conjunction of ground positive literals. All preconditions of an action must be true before it is applied. The effects of an action are expressed by a conjunction of ground literals which specify how a situation changes after the action is applied. An action (e.g.  $load(P_1, Plane_2, Heathrow)$ ) is a ground instance of an operator (e.g.  $load(x, y, z)$ ).  $load(P_1, Plane_2, Heathrow)$  is the action of loading package  $P_1$  in plane  $Plane_2$  at Heathrow airport. Its preconditions are  $package(P_1), plane(Plane_2), airport(Heathrow), at(P_1, Heathrow)$  and  $at(Plane_2, Heathrow)$ . Its effects are  $in(P_1, Plane_2), \neg at(P_1, Heathrow)$ .  $in(P_1, Plane_2)$  is also called “add effect” of the action and “ $at(P_1, Heathrow)$ ” is also called delete effect of the action.  $\neg$  denotes negation. Predicates like  $in(P_1, Plane_2)$  and  $airport(Heathrow)$  can be treated as propositions simply by rewriting them as  $inP_1Plane_2$  and  $airportHeathrow$  respectively.

Many of the recently developed classical planners cast planning as a CSP. Propositional satisfiability (SAT) is a specific kind of CSP in which all variables are boolean and the constraints involve variables connected with operators from boolean logic. Some of the recently developed efficient planners cast planning as propositional satisfiability. These include SAT-plan [4], [5], MEDIC [3] and the plan-

ner which casts plan reuse and plan merging as satisfiability [7]. SAT encodings of a large number of planning problems in benchmark domains contain a significant number of binary clauses. Simplification techniques for binary clauses have been shown to improve the performance of planning as SAT [2]. Advances in SAT solving like better branching heuristics can be exploited to further improve the performance of SAT-based planners.

More expressive planning problems contain quantifiers, conditional effects, metric time and resource quantities. Examples of such problems include many NASA planning applications [9]. In these applications, both spacecraft and planetary rovers use heaters to warm up various components and these actions may span several other actions or experiments [9]. There is an increasing interest in extending/adapting advances in classical planning to solving more expressive planning problems. Temporal Graphplan (TGP) is an extension of Graphplan [1] for handling actions with time durations. The LPSAT planner [10] solves planning problems involving resource quantities by transforming them into problems containing linear constraints and propositional clauses. Some of the constraints solved by LPSAT have a logical and a mathematical part, e.g. the constraint  $((a > 3) \Rightarrow (x \vee y))$ .

Development of more expressive planners involves several challenges. These include verification of soundness and completeness, besides getting optimal plans in a shorter time. The tasks of ensuring soundness and completeness are trivial for refinement planners. This is because refinement planners maintain different representations for different partial plans in the form of nodes in a search tree. In progression, an action sequence is executed starting at initial state and it is checked whether goal is true at the end of its execution. A partial plan is a set of constraints which may or may not be a plan. Partial plans generated by forward state-space planners and backward state-space planners are sequences of actions whose goal achieving capability can be verified by simple methods like progression. A partial plan generated by a partial-order planner contains a set of constraints like step-action bindings and partial orderings over steps. The goal achieving capability of such a partial plan can be verified simply by finding if there exists a total order over the steps which is consistent with the partial-order constraints in the partial plan such that the goal is achieved when the steps are executed in the order specified by the total order. The verification of soundness and completeness of more expressive planners which cast planning as CSP is non-trivial since a set of constraints needs to be identified such that these are solved if and only if there is a plan of  $k$  steps. This set needs to be loose enough to allow generation of all action sequences that are plans and tight enough to exclude generation of any action sequence that is not a plan.

The adaptation of SAT-plan to handle metric time is challenging because of task of identifying all constraints that must be satisfied if and only if there is a plan of bounded length. Temporal planning problem is the problem of finding a set of  $\langle action_i, start\_time_i \rangle$  tuples for achieving a given goal, starting with a given completely described initial state using actions that have time durations.  $start\_time_i$  is the start time of action  $action_i$ . This is the time at which the execution of  $action_i$  starts. We use the following assumptions in temporal planning from [9] when setting up the SAT encodings. Effects of actions are undefined during their execution. Action durations are integers. Preconditions of an action should all be true at its start time. The effects of an action hold only at the end of its execution. The SAT encoding is such that it has a solution if and only if there is a plan of  $k$  time steps. In particular, we identify twenty kinds of constraints which together form the encoding when translated into clauses. We also show how the encoding can be simplified so that the number of clauses and variables are reduced without losing soundness and completeness. We show how the encoding can be adapted to more complex case of temporal planning in which it is not necessary for different preconditions of an action to be true at same time and it is not necessary for different effects to hold at the same time. We show how temporal planning can be cast as a CSP other than SAT.

The rest of the paper is organized as follows. In section 2, we explain the classical planning as SAT paradigm and the state-space SAT encoding with explanatory frame axioms for classical planning from [5]. In section 3, we describe our state-space SAT encoding for temporal planning. In section 4, we discuss how the size of this encoding can be reduced and how the encoding can be modified to handle a more complex case of temporal planning where different preconditions and/or effects of an action are true at different times. In this section we also show how temporal planning can be cast as CSP other than SAT and describe two additional SAT encodings of temporal planning. One of these is the causal encoding whose size is not affected by time durations of actions at all. Our work is summarized in section 5.

## 2 Background

In this section, we explain how SAT-based planners work and describe the state-space SAT encoding from [5] for classical planning where all actions have unit duration. An action is a ground instance of an operator. For example, if  $move(x, y, z)$  is an operator for moving block  $x$  from top of block  $y$  or table to top of block  $z$  or table,  $x \neq y, y \neq z, x \neq z, x \neq Table$ , then there are  $O(n^3)$  actions when there are  $n$  blocks. In the following,  $\wedge, \vee, \Rightarrow$  denote logical AND, OR and implication respectively.

SAT-plan [4] works in the following manner: Based on initial state, number of steps assumed to exist in plan ( $k$ ), goal state and action description, it generates an encoding (SAT instance) such that the instance has a model if and only if there is a plan of  $k$  steps. The steps range from 0 to  $(k - 1)$ . The encoding is simplified by rules of inference of boolean logic, e.g.  $a \wedge (\neg a \vee b)$  can be simplified to  $(a \wedge b)$  (this simplification step is optional but most SAT planners carry this out). The encoding is then passed to a SAT solver. If it is satisfied, the solution (truth assignment) is interpreted and plan is output. If it cannot be satisfied, then value of  $k$  is increased and the process of encoding generation, simplification and solving is repeated. Variables representing an occurrence of actions at various steps are step-action binding variables.

The explanatory frame axiom-based state-space encoding [5] contains the following constraints: (i) All propositions true in the initial state are true at time 0 and all propositions false in the initial state are false at time 0. (ii) If an action occurs at time  $t$ , its preconditions are true at time  $t$  and its effects are true at time  $(t + 1)$ . (iii) If an action  $o_i$  needs proposition  $p$  true and action  $o_j$  deletes  $p$ , then  $o_i$  and  $o_j$  cannot occur at same time. (iv) All propositions from goal are true at time  $k$ . (v) If a proposition  $p$  is true at time  $t$  and false at time  $(t + 1)$ , some action deleting  $p$  must occur at time  $t$ . If a proposition  $p$  is false at time  $t$  and true at time  $(t + 1)$ , some action making  $p$  true must occur at time  $t$ . These constraints are included to ensure that truth of a proposition cannot change unless an action causing the change occurs. These constraints are known as explanatory frame axioms.

### 3 SAT Encoding for Temporal Planning

To generate a SAT encoding, we first need to bound the number of steps in plan. We denote this bound by  $k$ .  $O$  denotes the set of all actions in domain.  $U$  denotes the set of all ground fluents in domain. A fluent is a proposition whose truth can change. Propositions that are not made true or false by any action are considered to be invariants whose truth remains unchanged. Before explaining the constraints that appear in the propositional encoding of a temporal planning problem, we explain some relevant notation for various boolean variables in the encoding.  $pt(t)$  is a boolean variable which denotes the truth of true state of fluent  $p$  at time  $t$ . If  $pt(t)$  is assigned true, it means that fluent  $p$  is true at time  $t$ . If  $pt(t)$  is assigned false, it means that the fluent  $p$  is not true at time  $t$  (so it is either false or undefined/unknown).  $pu(t)$  denotes that the truth of the undefined state of fluent  $p$  at time  $t$ . If  $pu(t)$  is assigned true, it means that the fluent  $p$  is undefined at time  $t$ . If  $pu(t)$  is assigned false, it means that the fluent  $p$  is not undefined at time  $t$ . In this case  $p$  is either true or false at time  $t$ .  $pf(t)$  denotes that

the truth of the false state of fluent  $p$  at time  $t$ . If  $pf(t)$  is assigned true, it means that the fluent  $p$  is false at time  $t$ . If  $pf(t)$  is assigned false, it means that the fluent  $p$  is not false at time  $t$ . In this case  $p$  is either true or undefined at time  $t$ .  $o_i(t)$  denotes the occurrence of the action  $o_i$  at time  $t$ .  $o_i(t)$  is an action variable.  $pf(t)$ ,  $pt(t)$  and  $pu(t)$  are fluent variables. If  $o_i(t)$  is assigned true, it means that  $o_i$  occurs at time  $t$ . If  $o_i(t)$  is assigned false, it means that  $o_i$  does not occur at time  $t$ .  $d_i$  denotes the duration of the action  $o_i$ . The time steps in the encoding at which actions may occur range from 0 to  $(k - 1)$ .  $d_{max}$  is the maximum of durations of all actions. The following constraints appear in the state-space encoding of a temporal planning problem. The encoding is a state-space encoding since it refers to states of all fluents at all time steps. We assume that actions do not have negated preconditions.

1. The constraints of this kind state that each fluent is either false or true or undefined at every time step. For all fluents  $p$ , for all times  $t \in [0, k]$ ,  $(pt(t) \vee pf(t) \vee pu(t))$ . There are  $(k + 1) \cdot |U|$  clauses of this kind. These contain  $3 \cdot (k + 1) \cdot |U|$  variables.
2. The constraints of this kind state that at a time, a fluent cannot be in more than one of the following states: true, false and undefined. For all fluents  $p$ , for all times  $t \in [0, k]$ ,  $(pt(t) \Rightarrow \neg pf(t))$ ,  $(pt(t) \Rightarrow \neg pu(t))$ ,  $(pf(t) \Rightarrow \neg pu(t))$ . There are  $3 \cdot (k + 1) \cdot |U|$  clauses of this kind.
3. Fluents true in initial state are true at time 0. Fluents false in initial state are false at time 0. These constraints contribute  $|U|$  unit clauses to the encoding. If the initial state is  $(a \wedge b \wedge \neg c \wedge \neg d)$ , the encoding contains  $(at(0) \wedge bt(0) \wedge cf(0) \wedge df(0))$ .
4. Fluents from goal are true at time  $k$ . If goal is a conjunction of  $s$  fluents, these constraints contribute  $s$  unit clauses to the encoding.
5. If an action  $o_i$  occurs (starts) at time  $t$ ,  $0 \leq t \leq (k - d_i)$ , its preconditions are true at  $t$  and its effects are true at time  $(t + d_i)$ . These constraints contribute  $O(k \cdot |O| \cdot m)$  clauses to the encoding,  $m$  being the maximum sum of the number of preconditions and the number of effects of an action. These constraints contribute  $k \cdot |O|$  variables to the encoding. For example, if  $x$  and  $y$  are preconditions of action  $o_3$ , and  $\neg x$  and  $z$  are its effects, this constraint generates  $(o_3(j) \Rightarrow (xt(j) \wedge yt(j) \wedge xf(j + d_3) \wedge zt(j + d_3)))$ , where  $j \in [0, k - d_3]$ . Note that we require  $t \leq (k - d_i)$  because if  $o_i$  starts at  $t > (k - d_i)$ , it will finish at time  $(k + 1)$  or later and the encoding has only  $(k + 1)$  time steps ranging from 0 to  $k$ .
6. If an action  $o_i$  does not delete any of its preconditions, then if  $o_i$  starts at time  $t$ , where  $0 \leq t \leq (k - d_i)$ , then  $o_i$  cannot start at any time  $t' \in [t + 1, (t + d_i - 1)]$ ,  $d_i$  being duration of  $o_i$ . These constraints prevent an occurrence of an action during its execution, when the action does not

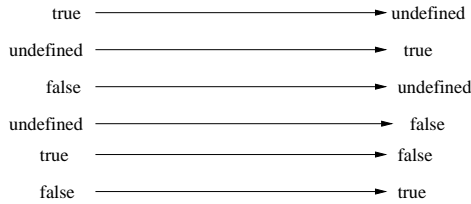
delete any of its preconditions. These can be considered as mutual exclusion relationships of actions with themselves. These constraints contribute  $O(k.q.d')$  clauses to the encoding, where  $q$  is the number of actions which do not delete any of their own preconditions and  $d'$  is the maximum of the durations of these actions.

**7.** If an action  $o_i$  does not delete its precondition  $x$ , then if  $o_i$  starts at time  $t$ ,  $x$  remains true over the closed interval  $[t + 1, t + d_i]$  when  $d_i > 1$ . If there are  $m$  actions that do not delete any of their own preconditions, these constraints contribute  $O(m.d_{max}.k.m')$  clauses to the encoding, where  $m'$  is the maximum number of preconditions of an action not deleted by the action.

**8.** The preconditions of an action  $o_i$  which are deleted by  $o_i$  are undefined over the closed interval  $[t + 1, t + d_i - 1]$  if  $o_i$  starts at  $t$ . If there are  $m_1$  actions that delete some of their own preconditions, these constraints contribute  $O(m_1.d_{max}.k.m'')$  clauses to the encoding, where  $m''$  is the maximum number of preconditions of an action deleted by the action. For example, if effects of action  $o_4$  are  $\neg p$  and  $q$ , such that  $p$  is one of its preconditions, and duration of  $o_4$  is 3, then this constraint yields  $(o_4(j) \Rightarrow (pu(j + 1) \wedge pu(j + 2)))$ .

**9.** Effects of an action  $o_i$  (except preconditions of  $o_i$  which  $o_i$  deletes) are undefined from time  $(t + 1)$  until time  $(t + d_i - 1)$ , including both these times, if  $o_i$  starts at  $t$ . These constraints contribute  $O(|O| .d_{max}.k.m''')$  clauses to the encoding, where  $m'''$  is the maximum number of effects of an action, excluding the preconditions deleted by it. For example, if effects of action  $o_4$  are  $\neg p$  and  $q$ , such that  $p$  is one of its preconditions, and duration of  $o_4$  is 3, then this constraint yields  $(o_4(j) \Rightarrow (qu(j + 1) \wedge qu(j + 2)))$ .

Constraints 10, 11, 12, 13, 14 and 15 are all explanatory frame axioms. These contribute  $O(k. |U|)$  clauses to the encoding. These constraints prevent changes in states of fluents without occurrences of actions causing these changes. As shown in Fig. 1, there are 6 kinds of changes that a state of a fluent may undergo.



**Figure 1. Six kinds of changes in states of a fluent**

**10.** If fluent  $p$  is true at time  $t$  and false at time  $(t + 1)$ , some action of duration 1 which deletes  $p$  must occur (start) at time  $t$ . For example, if actions  $o_4, o_6$  and  $o_9$  are the only actions of duration 1 which delete fluent  $x$ , this constraint

generates  $((xt(t) \wedge xf(t + 1)) \Rightarrow (o_4(t) \vee o_6(t) \vee o_9(t)))$ .

**11.** If fluent  $p$  is false at time  $t$  and true at time  $(t + 1)$ , some action of duration 1 which makes  $p$  true must occur (start) at time  $t$ .

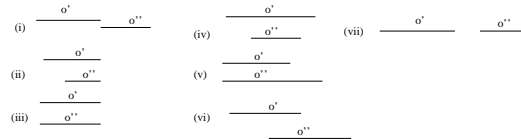
**12.** If fluent  $p$  is true at time  $t$  and undefined at time  $(t + 1)$ , some action of duration greater than 1 which deletes  $p$  must occur (start) at time  $t$ . For example, if  $k = 20$  and the only actions of duration more than 1 which delete fluent  $x$  are  $o_7, o_8$  and  $o_{11}$  such that  $d_7 = 4, d_8 = 12$  and  $d_{11} = 7$ , then we have  $((xt(t) \wedge xu(t + 1)) \Rightarrow (o_7(t) \vee o_8(t) \vee o_{11}(t)))$ ,  $t \in [0, 8]$ . We have  $((xt(t) \wedge xu(t + 1)) \Rightarrow (o_7(t) \vee o_{11}(t)))$ ,  $t \in [9, 13]$ . We have  $((xt(t) \wedge xu(t + 1)) \Rightarrow o_7(t))$ ,  $t \in [14, 16]$ . This constraint does not lead to any clauses for the change in state of  $x$  from true to unknown for  $t \in [17, 19]$ . This is because any action changing  $x$  in this fashion will end after  $t = 20$  in case it starts at time 17 or later.

**13.** If fluent  $p$  is false at time  $t \in [0, k - 2]$  and undefined at time  $(t + 1)$ , some action of duration greater than 1 which makes  $p$  true must occur (start) at time  $t$ .

**14.** If fluent  $p$  is undefined at time  $t \in [0, k - 1]$  and false at time  $(t + 1)$ , some action  $o_i$  of duration greater than 1 which makes  $p$  false must start at time  $(t + 1 - d_i)$ . For example, if  $o_5, o_7$  and  $o_{10}$  are the only actions which delete fluent  $y$  and  $d_5 = 2, d_7 = 5$  and  $d_{10} = 7$ , then this constraint is represented by  $((yu(t) \wedge yf(t + 1)) \Rightarrow (o_5(t - 1) \vee o_7(t - 4) \vee o_{10}(t - 6)))$ .

**15.** If fluent  $p$  is undefined at time  $t \in [0, k - 1]$  and true at time  $(t + 1)$ , some action  $o_i$  with duration greater than 1 which makes  $p$  true must start at time  $(t + 1 - d_i)$ .

All of the following constraints represent mutual exclusion relations between actions. These contribute  $O(k. |O|^2)$  clauses to the encoding. These constraints are identified after taking into account the all possible kinds of temporal relationships between two actions (seven types) from Figure 2 and all possible effects two actions may have on a fluent  $p$ . Given two actions  $o_i$  and  $o_j$  and a fluent  $p$ , the actions can affect the fluent in the following ways: (i) both  $o_i$  and  $o_j$  need  $p$  and both delete  $p$ , (ii) only  $o_i$  needs  $p$  and both  $o_i$  and  $o_j$  delete  $p$ , (iii) neither  $o_i$  nor  $o_j$  needs  $p$  and both delete  $p$ , (iv) neither  $o_i$  nor  $o_j$  needs  $p$  and both make  $p$  true, (v)  $o_i$  makes  $p$  true and  $o_j$  only deletes  $p$ , (vi)  $o_i$  only needs  $p$  and  $o_j$  only deletes  $p$  and, (vii)  $o_i$  makes  $p$  true and  $o_j$  both needs  $p$  and deletes  $p$ .



**Figure 2. Temporal relations between two actions  $o'$  and  $o''$**

**16.** If two actions  $o_i$  and  $o_j$  have same duration such that

$o_i$  deletes precondition of  $o_j$ , then  $o_i$  and  $o_j$  cannot start at same time  $t$ . The clauses  $\neg(o_i(t) \wedge o_j(t)), t \in [0, k - 1]$  are generated for all pairs of such actions to represent this constraint in the encoding.

**17.** If  $o_i$  with duration greater than 1 deletes precondition  $p$  of action  $o_j$  which has duration 1 such that  $o_j$  does not delete its own precondition  $p$ , then  $o_i$  and  $o_j$  cannot start at same time  $t$ . The clauses  $\neg(o_i(t) \wedge o_j(t)), t \in [0, k - 1]$  are generated for all pairs of such actions to represent this constraint in the encoding.

**18.** If  $o_i$  has precondition  $p$  which it also deletes and action  $o_j$  only deletes  $p$  ( $o_j$  does not need  $p$ ), then if  $o_i$  starts at  $t$ , then  $o_j$  cannot start at a time  $t' \geq t$  if  $(t' + d_j) = (t + d_i)$ . To represent this in the encoding, the clauses  $\neg(o_i(t) \wedge o_j(t'))$  are generated for all pairs of such actions, for all times  $t$  and  $t'$  such that  $(t + d_i) = (t' + d_j), t' \geq t, t \in [0, k - 1], t' \in [0, k - 1]$ .

**19.** If actions  $o_i$  and  $o_j$  both only delete some fluent  $p$  and both do not need  $p$ , then it cannot be the case that  $o_i$  ends at starting time of  $o_j$ . This avoids consecutive deletions of a fluent without making the fluent true. To represent this in the encoding, the clauses  $\neg(o_i(t) \wedge o_j(t'))$  are generated for all pairs of such actions, for all times  $t$  and  $t'$  such that  $(t + d_i) = t', t \in [0, k - 1], t' \in [0, k - 1]$ .

**20.** If two actions  $o_i$  and  $o_j$  both only delete some fluent  $p$  and both do not need it, then it cannot be the case that  $o_i$  ends at ending time of  $o_j$ . To represent this in the encoding, the clauses  $\neg(o_i(t) \wedge o_j(t'))$  are generated for all pairs of such actions, for all times  $t$  and  $t'$  such that  $(t + d_i) = (t' + d_j), t \in [0, k - 1], t' \in [0, k - 1]$ .

The encoding contains  $O(k \cdot (|O| + |U|))$  variables and  $O(k \cdot (|O| + |U|) + k \cdot |O|^2)$  clauses. The number of variables and clauses in the SAT encoding of a temporal planning problem are respectively higher than the the number of variables and clauses in the encoding of the same problem with all action durations set to 1. This is mainly because of the action durations and the three variables needed to model three different states of each fluent.

## 4 Discussion

We showed how temporal planning can be encoded as a SAT problem and found the asymptotic number of variables and clauses in the encoding. In this section, we show how the size of this encoding can be reduced without losing soundness and completeness. We also show how this encoding can be adapted to the more complex case of temporal planning in which it is not necessary for all preconditions of an action to be true at the same time and different effects of an action may become true at different times during its execution. We also show how the encoding is useful in casting temporal planning as a CSP other than SAT.

### 4.1 Reducing Encoding Size

Though a smaller encoding is not always easier to solve, smaller encodings have been shown to be solvable faster [4], [3], [7]. The size of an encoding can be reduced by propagating the truth assigned to unit clauses. Such unit clauses are available in the specification of initial state and goal in the encoding. Almost all SAT simplification techniques which apply to the case where all actions have duration 1 also apply to the case where actions have unequal durations and we will not discuss these.

One can select a better value of  $k$  to avoid solving several encodings generated with unacceptable lower values of  $k$ . The planning graph [1] can be used to wisely choose the value of  $k$ . Graphplan works in 2 phases. The first involves growing a **planning graph** and is called the **plan-graph** construction phase. This is a forward phase, beginning with the initial state. The second phase is a solution extraction phase. This is backward search phase starting with the goal. Plangraph, or planning graph (PG), has two kinds of levels called **action levels** and **proposition levels**. The 0th proposition level is the same as the initial state. The 0th proposition level occurs before the 0th action level which in turn occurs before the 1st proposition level which precedes the 1st action level, etc. In general, the  $i$  th proposition level is immediately succeeded by the  $i$  th action level. And, the  $i$  th action level immediately precedes  $(i + 1)$  th proposition level. A proposition level and an action level can be considered as sets whose members are the same as the contents of these levels. The  $i$  th action level in the plangraph contains all actions whose all preconditions appear in the  $i$  th proposition level. There is also a dummy action called a **no-op**, **maintenance action**, or **persistence action** in the  $i$  th action level for each proposition in the  $i$  th proposition level. The precondition and effect of this action is the proposition for which the action was created. This action is included in the plangraph because if no action changing the truth of the proposition occurs, the truth of the proposition remains same. The  $(i + 1)$  th proposition level is the union of the  $i$  th proposition level and the effects of the actions in the  $i$  th action level. Thus, proposition level  $i$  is a superset of proposition level  $(i - 1)$ . Similarly, action level  $i$  is a superset of action level  $(i - 1)$ .

There are three kinds of edges in plangraph: **(i)** edges from propositions in proposition level  $i$  to the same propositions in proposition level  $(i + 1)$  (for no-ops), **(ii)** edges from propositions in proposition level  $i$  to actions (whose precondition list contains these propositions) in action level  $i$  and **(iii)** edges from actions in action level  $i$  to propositions (which are effects of these actions) in proposition level  $(i + 1)$ .

A key to the efficiency of Graphplan is the inference of binary **mutex** (mutually exclusive) relations. Two kinds of

mutexes are found: (i) mutexes between actions, and (ii) mutexes between propositions. Each of these two kinds of mutexes could be static or dynamic. Static mutexes are found by examining preconditions and effects of actions. Dynamic mutexes are found by propagating static mutexes using truths of conditions in the initial state. Note that dynamic mutexes may be permanent or temporary.

If the plangraph has a proposition level that contains all conditions from the goal such that no two of these are mutex, Graphplan starts a backward search for a plan. If no solution is found, Graphplan extends planning graph by one action level and 1 proposition level and tries the solution extraction again. Graphplan is guaranteed to report unsolvability of a problem. A plangraph is said to **level off** if the none of the following change when the plangraph is grown further: (i) the number of actions in last action level, (ii) the number of propositions in last proposition level, (iii) the number of pairs of actions that are mutex in last action level, and (iv) the number of pairs of mutex propositions in last proposition level. In the planning graph in Fig. 3, false propositions are not shown in proposition levels for readability. Because of same reason, mutex relations are not shown. The planning graph has 3 proposition levels and 2 action levels. M1, M2, M3 and M4 are actions in the domain. The preconditions and effects of these actions are shown on the left and right sides of the boxes respectively. Here is the planning graph figure. The goal is achievable with the plan Step 0: M2, Step 1: M1 & M3. The actions M1 and M2 are static mutex and the actions M3, M4 are dynamic mutex in the first action level. Kautz & Selman have shown that planning graph can be used to reduce the size of a SAT encoding for planning with actions of unit duration [6].

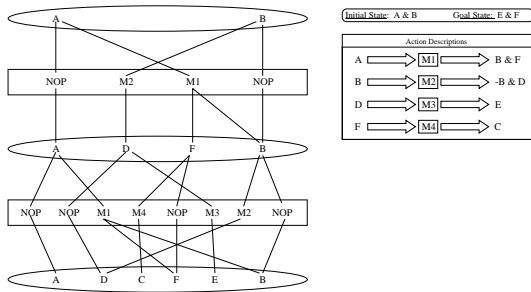


Figure 3. A planning graph

It is clear that actions that do not appear in the last action level of the leveled off planning graph are not relevant to solving the problem. Such actions need not be represented in the SAT encoding. Let us consider a leveled off planning graph with four action levels which are  $\{o_2, o_4\}$ ,  $\{o_2, o_4, o_5\}$ ,  $\{o_2, o_4, o_5, o_8, o_9\}$ , and  $\{o_2, o_4, o_5, o_8, o_9, o_{11}\}$  respectively, excluding no-ops from

the action levels. This planning graph is constructed by Graphplan [1]. If  $O = \{o_i \mid 1 \leq i \leq 20\}$ , then it can be concluded that actions  $o_1, o_3, o_6, o_7, o_{10}, o_{12}, o_{13}, o_{14}, o_{15}, o_{16}, o_{17}, o_{18}, o_{19}$  and  $o_{20}$  are not relevant to solving the problem. These need not be represented in the encoding for temporal planning. Since  $o_5$  occurs in the first action level of the planning graph, it is clear that in case  $o_5$  occurs in temporal plan, its start time will be greater than or equal to  $\min(d_2, d_4)$ . This is because the planning graph shows that  $o_5$  can occur only after one or more of the actions from  $\{o_2, o_4\}$  occur. This means that we need not create variables  $o_5(t), t \in [0, \min(d_2, d_4) - 1]$ . Similar argument shows that we need not create the following variables:  $o_8(t_1), t_1 \in [0, (\min(d_2, d_4) + \min(d_2, d_4, d_5)) - 1]$ ,  $o_9(t_2), t_2 \in [0, (\min(d_2, d_4) + \min(d_2, d_4, d_5)) - 1]$  and  $o_{11}(t_3), t_3 \in [0, (\min(d_2, d_4) + \min(d_2, d_4, d_5) + \min(d_2, d_4, d_5, d_8, d_9)) - 1]$ . This significantly reduces the number of variables and clauses in the encoding. This also reduces the number of literals in various clauses. The number of steps  $k$  can then be chosen so that  $\theta' \leq k \leq \theta''$  where  $\theta' = (\min(d_2, d_4) + \min(d_2, d_4, d_5) + \min(d_2, d_4, d_5, d_8, d_9) + \min(d_2, d_4, d_5, d_8, d_9, d_{11}))$  and  $\theta'' = (\max(d_2, d_4) + \max(d_2, d_4, d_5) + \max(d_2, d_4, d_5, d_8, d_9) + \max(d_2, d_4, d_5, d_8, d_9, d_{11}))$

## 4.2 More Complex Temporal Planning

We assumed that all preconditions of an action are true at the same time which is same as the time of start of execution of the action. We also assumed that all effects of an action hold at the same time which is same as the time of end of execution of the action. In reality, different effects may hold at different times and it may not be necessary for all preconditions of an action to hold at the same time. Consider the action  $move(A, B, C)$  which moves block  $A$  from top of block  $B$  to top of block  $C$ . Its preconditions are  $clear(A), on(A, B)$  and  $clear(C)$ . Let us assume that there are several grippers and one does not need the precondition  $hand - empty$ . The effects of this action are  $on(A, C), \neg clear(C), clear(B)$  and  $\neg on(A, B)$ .  $clear(A)$  and  $on(A, B)$  have to be true at same time. However  $clear(C)$  can become true later during the execution of the action since some other action may move block from the top of  $C$  before  $A$  is put on  $C$ . The effect  $\neg on(A, B)$  holds immediately and the effect  $on(A, C)$  becomes true much later. Consider the action  $fly(P, London, Paris)$  which flies plane  $P$  from  $London$  to  $Paris$ . Its effects are  $at(P, Paris), \neg at(P, London)$ . It is clear that  $\neg at(P, London)$  becomes true much earlier than  $at(P, Paris)$ . Such an action may be specified with times at which various preconditions are needed true relative to  $s$  and various effects become true, relative to  $s$ , where  $s$  is start time of the action.

Let us see whether one needs to change constraints from section 3 to handle such actions and if so how. Constraints 1 through 4 do not need any change to handle such actions. Constraint 5 needs a minor change. The new constraint specifies that different preconditions and effects are true at different times. Constraints 6 and 7 do not need any change. Constraint 8 needs a minor change. The new constraint states that precondition of an action deleted by itself is undefined over the interval  $[t + 1, t + r - 1]$ , when the precondition is deleted  $r$  time units after the start of the action,  $t$  being start time of the action. Constraint 9 needs a minor change to specify that different effects of an action are undefined over different time intervals. Constraints 10 and 11 do not need any change. Constraint 12 needs a minor change. The new constraint states that if a fluent  $p$  is true at time  $t$  and undefined at time  $(t + 1)$ , some action of duration greater than 1 which deletes  $p$  at two time units or more later than the time of start of its execution must occur at time  $t$ . The remaining constraints can be easily adapted to handle such actions.

### 4.3 Temporal Planning as CSP other than SAT

Since SAT can be easily transformed into 0-1 ILP, an ILP encoding of temporal planning can be directly created using our SAT encoding. For example, the clause  $(x \vee y \vee \neg z \vee \neg b)$  can be translated into the linear constraint  $(x' + y' + (1 - z') + (1 - b')) \geq 1$  where the domain of the integer variables  $x', y', z', b'$  is  $\{0, 1\}$ . The direct translation of our SAT encoding into 0-1 ILP encoding leads to  $O(k \cdot (|O| + |U|))$  integer variables and  $O(k \cdot (|O| + |U|) + k \cdot |O|^2)$  linear constraints among these.

One can create a CSP using the constraints that lead to the SAT encoding. Specifically, one can have  $k \cdot |O|$  boolean variables whose values represent occurrence/absence of actions at various time steps. One can have  $(k + 1) \cdot |U|$  variables with the domain  $\{t, f, u\}$  to represent various states of the fluents at all time steps. Note that in SAT encoding we need  $3 \cdot (k + 1) \cdot |U|$  boolean variables to represent various states of all fluents at all time steps. In the CSP formulation, we need only  $(k + 1) \cdot |U|$  variables. Though the domains of these variables contain 3 values, the worst-case size of the space of assignments to the fluent variables is lower in the CSP formulation. Since by definition each variable in CSP is assigned a unique value, we do not need constraints to specify that a fluent cannot be in more than one state at any time. Note that in the SAT encoding we need  $3 \cdot (k + 1) \cdot |U|$  binary clauses to specify that a fluent cannot be in more than one state at any time. In SAT encoding, we need  $(k + 1) \cdot |U|$  clauses to specify that each fluent is true or false or undefined at each time step. No constraints are needed in the CSP to specify this requirement. Remaining constraints leading to the SAT en-

coding can be translated to complete the CSP formulation. This shows how the constraints we developed to generate a SAT encoding are useful in casting temporal planning as a CSP. Note that no extra effort is needed to verify the correctness of the CSP formulation.

### 4.4 Other SAT Encodings

In this subsection, we discuss two additional SAT encodings of temporal planning. The first is a variation of our state-space encoding with explanatory frame axioms for temporal planning. The other encoding is an augmentation of the causal encoding for classical planning from [5].

Let us consider our state-space encoding with explanatory frame axioms for temporal planning. The variables for encoding undefined states of fluents can be eliminated from this encoding. The only use of these variables is that they prevent certain conflicting actions from overlapping. For example, if  $o_1$  deletes  $p$ ,  $p$  is undefined during the execution of  $o_1$ . If  $o_2$  needs  $p$  and does not delete it, then execution of  $o_2$  and  $o_1$  cannot overlap. Since  $p$  cannot be both true and undefined at the same time, executions of  $o_1$  and  $o_2$  cannot overlap. Then one does not need an explicit mutual exclusion constraint to prevent overlapping of  $o_1$  and  $o_2$ . The variables for encoding undefined states of fluents can be eliminated and explicit mutual exclusion constraints can be added to an encoding. If the variable for encoding undefined state of a fluent is eliminated, the variables for encoding true and false states can also be eliminated. So for a fluent  $p$ , one can eliminate  $pt(t)$ ,  $pf(t)$  and  $pu(t)$  variables and just use  $p(t)$  variable such that if  $p(t) = true$ ,  $p$  is true at time  $t$  and if  $p(t) = false$ ,  $p$  is false at time  $t$ . This elimination reduces the number of variables in the encoding. This allows us to remove the following types of clauses as well:  $\neg(pt(t) \wedge pf(t))$ ,  $\neg(pu(t) \wedge pf(t))$ ,  $\neg(pt(t) \wedge pu(t))$ . Instead of 6 kinds of explanatory frame axioms, only two kinds of explanatory frame axioms are needed, for change from *true* to *false* and vice versa, when the variables for undefined state are eliminated. This does however require an inclusion of several clauses for mutual exclusion between actions. This may not be a concern since the experiments on SAT encodings of planning show that reducing the number of variables is more important than reducing the number of clauses.

Let us consider the causal encoding for classical planning from [5]. Though it has been shown [8] that causal encodings are harder to solve than state-space encodings in classical planning, causal encodings may be more useful for temporal planning than state-space encodings. This is because causal encodings represent partial order on steps and because there is no explicit representation of time. As a result if the difference in the durations of actions is high, this will not blow up the size of the causal en-

coding. Causal encodings represent all step-action bindings. For example, if there are 4 actions  $o_1, o_2, o_3$  and  $o_4$  and there are 3 steps  $p_1, p_2$  and  $p_3$ , the causal encoding contains the following constraints among many others:

$$\begin{aligned} &((p_1 = o_1) \vee (p_1 = o_2) \vee (p_1 = o_3) \vee (p_1 = o_4) \vee (p_1 = \phi)), \\ &((p_2 = o_1) \vee (p_2 = o_2) \vee (p_2 = o_3) \vee (p_2 = o_4) \vee (p_2 = \phi)), \\ &((p_3 = o_1) \vee (p_3 = o_2) \vee (p_3 = o_3) \vee (p_3 = o_4) \vee (p_3 = \phi)), \\ &\neg((p_1 = o_1) \wedge (p_1 = \phi)), \neg((p_1 = o_2) \wedge (p_1 = \phi)), \\ &\neg((p_1 = o_3) \wedge (p_1 = \phi)), \neg((p_1 = o_4) \wedge (p_1 = \phi)), \\ &\neg((p_1 = o_1) \wedge (p_1 = o_2)), \neg((p_1 = o_1) \wedge (p_1 = o_3)), \\ &\neg((p_1 = o_2) \wedge (p_1 = o_3)), \neg((p_1 = o_2) \wedge (p_1 = o_4)), \\ &\neg((p_1 = o_1) \wedge (p_1 = o_4)). \end{aligned}$$

In brief, it contains clauses that represent that each step is bound to one and only action, including no-op  $\phi$ .  $\phi$  has no preconditions.  $\phi$  has no effects. The use of no-ops allows an extraction of plans with fewer steps than  $k$ . The causal encoding also contains clauses for resolving conflicts arising out of deletion of preconditions. It also contains clauses that specify anti-symmetry, irreflexivity and transitivity of partial ordering over steps, along with some other clauses. The variables of type  $p_i = o_j$  in the above clauses are boolean variables. If  $p_i = o_j$  is assigned true, then step  $p_i$  is bound to action  $o_j$ . If  $p_i = o_j$  is assigned false, then step  $p_i$  is not bound to action  $o_j$ . The causal encoding [5] also has variables of type  $p_i \prec p_j$ .  $\prec$  denotes partial order. If the boolean variable  $p_i \prec p_j$  is assigned true, it means that step  $p_i$  precedes step  $p_j$ . This means that if action bindings of  $p_i$  and  $p_j$  are  $o_p$  and  $o_q$  respectively, then  $o_p$  occurs before  $o_q$ . Let us consider the case of temporal planning where (i) all effects of an action hold at the same time which is same as the end of its execution, and, (ii) all preconditions of an action must be true at same time which is same as the start of execution of the action. For this type of temporal planning, the causal encoding [5] can be used with the following additions: **(i)** This is about interpreting truth assignment of the partial-order variables. If  $p_i \prec p_j$  is assigned true, the action bound to  $p_i$  ends before action bound to  $p_j$  starts. **(ii)** For each pair of actions  $o_i$  and  $o_j$  that cannot overlap, and, all pairs of steps  $p_a$  and  $p_b$ , the following clause should be added to the causal encoding from [5]:  $((p_a = o_i) \wedge (p_b = o_j)) \Rightarrow ((p_a \prec p_b) \vee (p_b \prec p_a))$ . This leads to additional  $O(k^2 \cdot |O|^2)$  clauses.  $k$  is the number of steps in the causal encoding.  $k$  does not represent the time duration of execution of plan. Once the encoding is solved, start times can be assigned to various actions in polynomial time using the truth assignments of the step-action binding variables (e.g.  $p_a = o_i$ ) and the variables for partial orders on steps (e.g.  $p_a \prec p_b$ ). That gives an executable temporal plan. Since durations of actions and their start times are not represented in the encoding, the times do not affect the size of the encoding at all.

## 5 Summary

There is an increasing amount of work on extending/adapting the recent advances in plan synthesis under classical assumptions to more expressive planning. More expressive planning involves dealing with metric time, conditional effects, quantifiers and resource quantities. Verifying the soundness and completeness of such planners which cast planning as some kind of CSP is non-trivial. We developed a SAT encoding for temporal planning such that the encoding is solvable if and only if there is a plan whose execution time is less than or equal to chosen bound  $(k + 1)$ . We showed how the size of the encoding can be significantly reduced using information in planning graph. We showed how the SAT encoding can be adapted to handle actions all of whose preconditions need not be true at same time and/or whose effects become true at different times. We also showed how the SAT encoding is useful in casting temporal planning as a 0-1 ILP and as a CSP different from SAT and 0-1 ILP. We also reported on two additional encodings for temporal planning. One of them is causal encoding whose size is not affected by durations of actions at all.

**Acknowledgement:** This work was funded by NSF grant IIS-0119630 to the author. The author thanks Subbarao Kambhampati, David Smith, Martha Pollack, Maria Fox, and Ronen Brafman for their useful comments on the ideas in this paper.

### References

- [1] Avrim Blum and Merrick Furst, Fast planning through planning graph analysis, *Artificial Intelligence* 90, 1997, 281-300.
- [2] Ronen I. Brafman, A simplifier for propositional formulas with many binary clauses, *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 2001.
- [3] Michael Ernst, Todd Millstein and Daniel Weld, Automatic SAT compilation of planning problems, *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 1997.
- [4] Henry Kautz and Bart Selman, Pushing the envelope: Planning, propositional logic and stochastic search, *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1996.
- [5] Henry Kautz, David McAllester and Bart Selman, Encoding plans in propositional logic, *Proceedings of Knowledge Representation and Reasoning conference (KR)*, 1996.
- [6] Henry Kautz and Bart Selman, Unifying SAT-based and graph-based planning, *Proceedings of IJCAI*, 1999.
- [7] Amol Mali, Plan merging and plan reuse as satisfiability, *Proceedings of European Conference on Planning (ECP)*, Durham, UK, 1999.
- [8] Amol Mali and Subbarao Kambhampati, On the utility of causal encodings, *Proceedings of the national conference on artificial intelligence (AAAI)*, 1999.

[9] David E. Smith and Daniel S. Weld, Temporal planning with mutual exclusion reasoning, Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 1999.

[10] Steven Wolfman and Daniel Weld, The LPSAT engine and its application to resource planning, Proceedings of IJCAI, 1999.