Incorporation of Temporal Logic Control into Plan Operators

Jussi Rintanen[1]

Abstract. Domain-specific control information is often essential in solving difficult planning problems efficiently. Temporal logics are a declarative and expressive representation for such control information. In this paper we investigate the integration of temporal logic control information into plan operators. For a given control formula Φ and operators O, we produce a new set O_Φ of operators that works like O under the control of Φ. We show that for a subclass of temporal logic formulae the compilation causes only a low-polynomial increase in the size of the operators, that the length of plans is not affected, and that the speed-up obtained is competitive with what is achieved with temporal logic control as formula progression. The translation uses operators with conditional effects. An important benefit of our approach is that the problem of using temporal logic formulae as control information is solved once and for all: temporal logic control can be implemented as a preprocessing step for all kinds of planning algorithms.

1 INTRODUCTION

The applicability of automated planners in many applications is highly dependent on the presence of control information that reliably guides the planner in choosing which operators to apply. In the past, the control information has been very procedural and dependent on the planning algorithm being used. Bacchus and Kabanza propose temporal logics as a language for expressing control information and show that a simple forward-chaining planner that interprets those formulae can be very efficient for planning problems that allow strong control strategies [2]. The declarativeness of temporal logic makes it possible to separate the control information from the implementation techniques of a planner, which is a great benefit. For example Huang et al. show how to use Bacchus and Kabanza's temporal logic control information in a very different kind of planner [7].

However, there are many different planning algorithms, and incorporating a temporal logic reasoner to each and every one may be complicated or otherwise undesirable. Considering that both plan operators and temporal logic have a high expressivity, there is a redundancy in the input as both the operators and the control information could be represented in either of the formalisms. Therefore there is no compelling need for algorithms that take input of both types. This idea suggests two non-hybrid approaches to planning with domain-specific control information: deductive planning in the framework of temporal logic, and (possibly non-deductive) planning based on algorithms that operate on plan operators only with control embedded in the operators. Both approaches are worth investigating, but in this paper we take the second because of its direct usefulness for almost any type of planning. Once the control information expressed as temporal logic formulae is compiled into plan operators (a preprocessing step), no planning algorithm needs to be aware of temporal logic.

In this paper we show how temporal logic control can often be effectively compiled away by implicitly representing the formulae in the plan operators. More precisely, for arbitrary sets of plan operators and for formulae from a subclass of linear temporal logic, we show how to produce sets of plan operators that work like the original sets under the control of the formulae. We assume that plan operators can represent conditional effects; that is, the sets of facts that change their truth-values may depend on the truth-values of other facts. With conditional effects we also represent disjunctive preconditions that also appear to be necessary for encoding control information. This approach suggests that rather than addressing the problem of control knowledge for every planning algorithm separately, it may be more productive to develop efficient domain-independent planning algorithms with more expressive input languages.

To demonstrate the feasibility of the approach we run a series of problems on two planners, TLPlan [2] and IPP [10], that respectively use forward and backward chaining. For the sample problems and both planners we obtain a speed-up of two orders of magnitude.

2 CONTROL FORMULAE IN TEMPORAL LOGIC

We consider finite models $M = \langle v_1, \ldots, v_n \rangle$ of a linear temporal logic [6] where v_i are valuations $v_i : A \to \{T, F\}$ that map propositional variables to truth values. The size of a model $|M|$ is defined as the number n of valuations. We define the truth of formulae at a point $i \in \{1, \ldots, |M|\}$ of a model recursively as follows.

$M \models_i p$ if and only if $v_i(p) = T$, where $p \in A$.

$M \models_i \neg\phi$ if and only if $M \not\models_i \phi$.

$M \models_i \phi \vee \phi'$ if and only if $M \models_i \phi$ or $M \models_i \phi'$.

$M \models_i \bigcirc\phi$ if and only if $M \models_{i+1} \phi$.

$M \models_i \phi\mathcal{U}\phi'$ if and only if $M \models_j \phi$ for all $j \in \{i, \ldots, n\}$ such that $M \models_k \phi'$ for no $k \in \{i, \ldots, j\}$.

We say that a model M satisfies a formula ϕ if and only if $M \models_i \phi$ for all $i \in \{1, \ldots, |M|\}$. We define the operator always by $\Box\phi \equiv^{def} \phi\mathcal{U}\bot$. The constants true and false are respectively $\top$ and $\bot$. The above semantics for until $\mathcal{U}$ is only one of several possible ones. In this paper a semantics for until that requires that ϕ' eventually becomes true is also useful: $M \models_i \phi\mathcal{U}^2\phi'$ if and only if for some $j \in \{i, \ldots, n\}$ $M \models_j \phi'$ and $M \models_k \phi$ for all $k \in \{i, \ldots, j-1\}$. We define the operator eventually by $\Diamond\phi \equiv^{def} \top\mathcal{U}^2\phi$.

A main difference from the temporal logic used by Bacchus and Kabanza [2] is that we do not have a goal modality. This would be unnecessary because information concerning the goals can be repre-

[1] Albert-Ludwigs-Universität Freiburg, Institut für Informatik, Georges-Köhler-Allee, D-79110 Freiburg im Breisgau, Germany

sented as conventional facts. Another difference is that we assume that all propositions in the control formulae occur in the operators; that is, we do not have defined predicates. It is straightforward to replace the occurrences of defined predicates by their definitions. Of course, this may increase the size of the formula.

3 COMPILATION OF CONTROL INTO PLAN OPERATORS

It seems that representing complex control information in conventional plan operators $p \Rightarrow e$, where p and e are sets (conjunctions) of literals, is not in general feasible. A more expressive form of plan operators allows conditional effects which means that the set of facts changed by the application of an operator may depend on the current truth-values of facts. An operator $p \Rightarrow e$ with conditional effects may have both literals and rules $c \twoheadrightarrow f$ in e. When $p \Rightarrow e$ is applied, all literals in e become true and literals in f for $c \twoheadrightarrow f \in e$ become true if the literals in c were true.

There are a number of planners that support conditional effects [10, 1, 2] and incorporating them in many others is easy. There are many transition systems that can be represented exponentially more concisely with conditional effects than without, and this seems to be the case in encoding control information in plan operators. Notice that we still assume the original sets of operators to be unconditional, and the lifted version of the translation needs also universal and existential quantification over individuals.

For our translation we consider sets Φ of clauses $\kappa \vee (\phi \mathcal{U} \epsilon)$, $\kappa \vee (\phi \mathcal{U}^2 \epsilon)$, and $\kappa \vee \bigcirc\theta$, where ϕ, ϵ and θ are literals and κ is a disjunction of literals.[2] The modalities $\Box$ and $\Diamond$ are reduced to $\mathcal{U}$ and $\mathcal{U}^2$ like mentioned earlier. For literals l and sets s of literals we write $\overline{l}$ and $\overline{s}$ for the complement of l and the set consisting of the complements of literals in s, respectively.

Next we give the translation. It extends each operator so that it cannot be applied if it violates the control formulae. If we commit to the truth of $\phi \mathcal{U} \epsilon$ for $\kappa \vee (\phi \mathcal{U} \epsilon) \in \Phi$ at the current time point, we have to guarantee that the future truth-values of ϕ and ϵ obey the semantics of $\mathcal{U}$. For this we use auxiliary facts $\mathrm{U}^{\kappa}_{\phi,\epsilon}$ that become true when respective formulae κ become false, and they stay true until κ is true and ϵ is true in the current or in the preceding time point. The translation of $\mathcal{U}^2$ is like that of $\mathcal{U}$ (the only difference is in the definition of the new goal), and the translation of $\bigcirc$ is simple and does not require auxiliary variables.

Below, $\bot$ is a fact that is false in initial and goal states and is not made false by any operator: $\bot$ may not be an effect of any operator and c must be false for effects $c \twoheadrightarrow \bot$.

We incorporate Φ into $P = \langle I, O, G \rangle$, where I is an initial state, O is a set of operators and G is a goal, as $F_\Phi(P) = \langle I_\Phi, O_\Phi, G_\Phi \rangle$ where $O_\Phi = \{o' | o \in O, o \stackrel{\Phi}{\Longrightarrow} o'\}$ and $\stackrel{\Phi}{\Longrightarrow}$ is defined as follows. Let $p \Rightarrow e$ be an operator in O.

1. The operator may make the non-modal literals in a clause false and therefore require the satisfaction of $\phi \mathcal{U} \epsilon$ (or be inapplicable if neither ϕ nor ϵ is true.) Let

$$\begin{aligned} E_1 &= \{\overline{r} \backslash e \twoheadrightarrow \bot | \kappa \vee (\phi \mathcal{U} \epsilon) \in \Phi, r = \kappa \cup \{\phi, \epsilon\}, \\ &\qquad e \cap \overline{r} \neq \emptyset, e \cap r = \emptyset\}, \\ E_2 &= \{\overline{\kappa} \backslash e \twoheadrightarrow \mathrm{U}^{\kappa}_{\phi,\epsilon} | \kappa \vee (\phi \mathcal{U} \epsilon) \in \Phi, \\ &\qquad e \cap \overline{\kappa} \neq \emptyset, e \cap \kappa = \emptyset\}. \end{aligned}$$

2. The operator may not be applied if it falsifies an active $\phi \mathcal{U} \epsilon$ by making both ϕ and ϵ false. The conditions $\kappa \cap e = \emptyset$ are needed for the case in which κ becomes true and ϵ is true in the current or in the preceding time point: then the new values of ϕ and ϵ do not matter. Let

$$\begin{aligned} E_3 &= \{\mathrm{U}^{\kappa}_{\phi,\epsilon}, \overline{\phi} \twoheadrightarrow \bot | \kappa \vee (\phi \mathcal{U} \epsilon) \in \Phi, \\ &\qquad \overline{\epsilon} \in e, \phi \notin e, \kappa \cap e = \emptyset\}, \\ E_4 &= \{\mathrm{U}^{\kappa}_{\phi,\epsilon}, \overline{\epsilon} \twoheadrightarrow \bot | \kappa \vee (\phi \mathcal{U} \epsilon) \in \Phi, \overline{\phi} \in e, \epsilon \notin e\}, \\ E_5 &= \{\mathrm{U}^{\kappa}_{\phi,\epsilon} \twoheadrightarrow \bot | \kappa \vee (\phi \mathcal{U} \epsilon) \in \Phi, \\ &\qquad \{\overline{\phi}, \overline{\epsilon}\} \subseteq e, \kappa \cap e = \emptyset\}. \end{aligned}$$

3. The operator may deactivate $\mathrm{U}^{\kappa}_{\phi,\epsilon}$ by reaching a state in which κ is true and ϵ is true in the current or in the preceding state. Let

$$\begin{aligned} E_6 &= \{\neg \mathrm{U}^{\kappa}_{\phi,\epsilon} | \kappa \vee (\phi \mathcal{U} \epsilon) \in \Phi, \epsilon \in e, \kappa \cap e \neq \emptyset\}, \\ E_7 &= \{l \twoheadrightarrow \neg \mathrm{U}^{\kappa}_{\phi,\epsilon} | \kappa \vee (\phi \mathcal{U} \epsilon) \in \Phi, l \in \kappa, \overline{l} \notin e, \epsilon \in e\}, \\ E_8 &= \{\epsilon \twoheadrightarrow \neg \mathrm{U}^{\kappa}_{\phi,\epsilon} | \kappa \vee (\phi \mathcal{U} \epsilon) \in \Phi, \kappa \cap e \neq \emptyset\}. \end{aligned}$$

4. For the translation of formulae $\kappa \vee \bigcirc\theta \in \Phi$, an operator may not be applied if it makes θ false and κ was false at the preceding time point, or if it does not make θ true when it and κ were false before.

$$\begin{aligned} E^{\circ}_1 &= \{\overline{\kappa} \twoheadrightarrow \bot | \kappa \vee \bigcirc\theta \in \Phi, \overline{\theta} \in e\} \\ E^{\circ}_2 &= \{\overline{\kappa}, \overline{\theta} \twoheadrightarrow \bot | \kappa \vee \bigcirc\theta \in \Phi, \theta \notin e\} \end{aligned}$$

A problem with E°_2 is that for every $\kappa \vee \bigcirc\theta \in \Phi$, every operator – even when they do not affect any of the literals in κ or θ – potentially violates this formula by not making θ true when it was false before. However, often $\overline{\theta} \in \kappa$ and the antecedent of the effect from E°_2 is therefore inconsistent and the effect can be eliminated.

Now $(p \Rightarrow e) \stackrel{\Phi}{\Longrightarrow} (p \Rightarrow (e \cup \bigcup_{i=1}^{8} E_i \cup \bigcup_{i=1}^{2} E^{\circ}_i))$.

An initial state I is extended with a valuation for auxiliary variables to obtain I_Φ: for all $\kappa \vee (\phi \mathcal{U} \epsilon) \in \Phi$, $I_\Phi \models \mathrm{U}^{\kappa}_{\phi,\epsilon}$ iff $I \models \neg\kappa$. Similarly for $\mathcal{U}^2$. Of course, the control information contradicts the initial state if $I \models \neg(\kappa \vee \phi \vee \epsilon)$ for some $\kappa \vee (\phi \mathcal{U} \epsilon) \in \Phi$, and then there are no plans that satisfy Φ. For the modal operator $\mathcal{U}^2$ the goal $G_\Phi = G \cup \{\neg \mathrm{U}^{\kappa}_{\phi,\epsilon} | \kappa \vee (\phi \mathcal{U}^2 \epsilon) \in \Phi\}$ requires that ϵ for every $\phi \mathcal{U}^2 \epsilon$ eventually becomes true.

There is a polynomial upper bound on the size of the translation.

Theorem 1 *The size of O_Φ is $O(n|O| + m)$ where n is the size of Φ and m is the size of O.*

Proof: For $o \in O$ and $\phi \in \Phi$, the operator o' with $o \stackrel{\Phi}{\Longrightarrow} o'$ contains at most $k + 7$ new effects (from $E_1, \ldots, E_8, E^{\circ}_1, E^{\circ}_2$), where k is the number of literals in ϕ. The sizes of the effects from E_1, E_2, E°_1 and E°_2 are proportional to the size of ϕ. The other effects are of constant size. Hence the difference of the sizes of o' and o is $O(n)$. $\Box$

For a given plan s there is a temporal logic model $\langle v_1, \ldots, v_{|s|+1} \rangle$ where $v_1 \models I$ and v_{i+1} is obtained from v_i by executing the ith operator in s. By $|s|$ we denote the length of the plan. We say that a plan s satisfies a formula Φ if the corresponding temporal logic model satisfies Φ.

Theorem 2 *$P = \langle I, O, G \rangle$ has a plan that satisfies the formula Φ if and only if $P_\Phi = \langle I_\Phi, O_\Phi, G_\Phi \rangle$ has a plan.*

Proof: We have a detailed proof but it is 2 pages long and relatively straightforward. The following are the facts we establish by induction. The first two are used in showing that plans for P_Φ satisfy Φ

[2] We will often identify disjunctions or conjunctions of literals with corresponding sets of literals so that union, intersection and set membership can be used on them.

and hence there is a plan for P that satisfies Φ, and the third for showing that if a plan for P satisfies Φ then there is a corresponding plan for P_Φ.

1. Given model M_Φ of a plan for P_Φ, for all $i \in \{1, \ldots, |M_\Phi|\}$ and $\kappa \vee (\phi \mathcal{U} \epsilon) \in \Phi$, $M_\Phi \models_i \mathrm{U}^{\kappa}_{\phi,\epsilon}$ or $M_\Phi \models_i \kappa$.
2. Given model M_Φ of a plan for P_Φ, for all $i \in \{1, \ldots, |M_\Phi|\}$ and $\kappa \vee (\phi \mathcal{U} \epsilon) \in \Phi$, if $M_\Phi \models_i \mathrm{U}^{\kappa}_{\phi,\epsilon}$ then $M_\Phi \models_i \phi \mathcal{U} \epsilon$.
3. Given a sequence of operators from P_Φ that was obtained from a plan for P that satisfies Φ, for all $i \in \{1, \ldots, |M_\Phi|\}$ and $\kappa \vee (\phi \mathcal{U} \epsilon) \in \Phi$, $M_\Phi \models_i \mathrm{U}^{\kappa}_{\phi,\epsilon}$ if and only if $M_\Phi \not\models_i \kappa$ or there is $j \in \{1, \ldots, i-1\}$ such that $M_\Phi \not\models_j \kappa$ and for all $k \in \{j, \ldots, i\}$ $M_\Phi \not\models_k \epsilon$. The antecedents of rules $c \twoheadrightarrow \bot$ in the effect of the operator at step i are false.

The correctness of the translation for $\bigcirc$ is easy and for $\mathcal{U}^2$ it follows easily from the correctness of $\mathcal{U}$. The proof also shows that plan length is not affected. □

4 LIFTING THE TRANSLATION

The translation so far addresses ground operators only. In many applications operators can be represented more concisely as schemata that correspond to sets of ground operators. In the next section we give an example that illustrates how the translation can be lifted to schematic operators.

5 REPRESENTING CONTROL INFORMATION

We show how control information is encoded and how the translation of $\mathcal{U}$ works with the logistics domain and the control rules given by Bacchus and Kabanza [2]. The goals are introduced in the initial state as normal facts with the prefix $\mathcal{G}$. We also use facts $\mathcal{G}\mathrm{atcity}(p, l)$ that say that the goal location of package p is in the same city as location l. The variables below are typed as follows: t is a truck, p a package, a an airplane, and l a location. In the following formulae the variables are universally quantified unless indicated otherwise.

The axioms below respectively state the following. A package must be loaded in a truck if it is in a wrong city and not at an airport. A package must be loaded in a truck if it is in a wrong location of the right city. A package must be loaded in an airplane if it is in a wrong city.

$$C1\ \ \mathrm{at}(t,l) \wedge \mathrm{at}(p,l) \wedge \neg\mathcal{G}\mathrm{atcity}(p,l) \wedge \neg\mathrm{airport}(l) \rightarrow \mathrm{at}(t,l)\mathcal{U}\mathrm{in}(p,t)$$
$$C2\ \ \mathrm{at}(t,l) \wedge \mathrm{at}(p,l) \wedge \mathcal{G}\mathrm{atcity}(p,l) \wedge \neg\mathcal{G}\mathrm{at}(p,l) \rightarrow \mathrm{at}(t,l)\mathcal{U}\mathrm{in}(p,t)$$
$$C3\ \ \mathrm{at}(a,l) \wedge \mathrm{at}(p,l) \wedge \neg\mathcal{G}\mathrm{atcity}(p,l) \rightarrow \mathrm{at}(a,l)\mathcal{U}\neg\mathrm{at}(p,l)$$

A package at the goal location may not be moved anywhere. A package may not be loaded in a truck at the airport of a wrong city. A package may not be loaded in an airplane if it is in the right city.

$$C10\ \ \mathrm{at}(p,l) \wedge \mathcal{G}\mathrm{at}(p,l) \rightarrow \Box\mathrm{at}(p,l)$$
$$C11\ \ \mathrm{at}(p,l) \wedge \neg\mathcal{G}\mathrm{atcity}(p,l) \wedge \mathrm{airport}(l) \rightarrow \mathrm{at}(p,l)\mathcal{U}\exists a\mathrm{in}(p,a)$$
$$C12\ \ \mathrm{at}(p,l) \wedge \mathcal{G}\mathrm{atcity}(p,l) \rightarrow \Box\neg\mathrm{in}(p,a)$$

We have similar formulae that say when packages must be unloaded and must not be unloaded and that prevent unnecessary vehicle movement. They express the same rules as Bacchus and Kabanza's formulae [2]. There is no space to give them here. The formulae that prevent unnecessary vehicle movement are more complicated than the ones above. The same formulae were found problematic by Huang et al. [7].

All the logistics control rules can be formalized with the operator $\bigcirc$. The consequents of the implications in C1-C3 and C10-C12 are respectively replaced by $\bigcirc\mathrm{at}(t,l)$, $\bigcirc\mathrm{at}(t,l)$, $\bigcirc\mathrm{at}(a,l)$, $\bigcirc\mathrm{at}(p,l)$, $\bigcirc\neg\mathrm{in}(p,t)$, and $\bigcirc\neg\mathrm{in}(p,a)$.

Next we show how formulae C1-C3 and C10-C12 are incorporated into the operator for loading a package into a truck.

```
(:action load-truck
 :parameters (?o - obj ?t - truck ?l - location)
 :precondition (and (at ?o ?l) (at ?t ?l))
 :effect (and (in ?o ?t) (not (at ?o ?l))))
```

Many of the effects are redundant and can be eliminated by using general-purpose simplification rules: for example rules $c \twoheadrightarrow e$ in operators with preconditions p can be removed if $p \cup c \cup I$ is inconsistent (I is a set of invariants [13]). In the logistics example this happens to all effects from E_1.

From E_6 and C1, C2 and C3 we obtain the following.

```
(not (untilC1 ?o ?t ?l))
(not (untilC2 ?o ?t ?l))
(forall (?p - airplane) (not (untilC3 ?o ?p ?l)))
```

The first two effects allow the movement of trucks after packages have been loaded. The third effect is unnecessary.

The effects from E_7 handle the situation in which κ has become true again before ϵ becomes true. Many of these effects can be eliminated. For example, if κ contains a static fact l (that is not affected by any operator), κ's becoming true cannot be caused by l, and the respective rule $l \twoheadrightarrow \neg\mathrm{U}^{\kappa}_{\phi,\epsilon}$ is therefore unnecessary. Similarly, if $\overline{\phi} \in \kappa$, $\overline{\phi}$ cannot become true before ϵ because this would violate $\phi \mathcal{U} \epsilon$ and is prevented by E_4, and $\overline{\phi} \twoheadrightarrow \neg\mathrm{U}^{\kappa}_{\phi,\epsilon}$ can therefore be removed.

From E_7 and C1, C2 and C3 we obtain the following.

```
(forall (?l2 - location)
  (when (not (at ?o ?l2))
     (not (untilC1 ?o ?t ?l2))))
(forall (?l2 - location)
  (when (not (at ?o ?l2))
     (not (untilC2 ?o ?t ?l2))))
(forall (?l2 - location ?p - airplane)
  (when (not (at ?o ?l2))
     (not (untilC3 ?o ?p ?l2))))
```

Also these effects are unnecessary, but we do not have a general-purpose rule for eliminating them.

From E_8 and C1-C3 and C10-C12 we obtain the following.

```
(forall (?t2 - truck)
  (when (in ?o ?t)
    (not (untilC1 ?o ?t ?l))))
(forall (?t2 - truck)
  (when (in ?o ?t)
    (not (untilC2 ?o ?t ?l))))
(forall (?p - airplane)
  (when (not (at ?o ?l))
    (not (untilC3 ?o ?p ?l))))
(when (false) (not (untilC10 ?o ?l)))
(when (exists (?p - airplane) (in ?o ?p))
  (not (untilC11 ?o ?l)))
(when (false) (not (untilC12 ?o ?l)))
```

From E_4 and C10 and C11 we obtain the following.

```
(when (and (untilC10 ?o ?l) (not (false))) (false))
(when (and (untilC11 ?o ?l)
           (not (exists (?p - airplane) (in ?o ?p)))
      (false))
```

These effects prevent loading if the package is in the goal location or waiting for an airplane.

In the lifted version of the translation universal quantification in the operators is often needed. For example, operators moving a truck potentially affect C1 and C2 for every package. Sometimes quantification can be avoided because of an invariant [13]. For example, *load-truck* makes the antecedent of the control formula that prevents unloading the package true. However, this formula prevents unloading at all locations that are not airports or destination locations. An invariant guarantees that the truck is at the current location only, and quantification over other locations is unnecessary.

6 EXPERIMENTS AND DISCUSSION

We have solved a number of logistics problems with TLPlan [2] and IPP [10] and both versions of the control formulae, with the operator $\mathcal{U}$ and with $\bigcirc$ only. These planners were chosen because their input languages can express disjunctive preconditions and conditional effects, and they do search in opposite directions. IPP is an implementation of the Graphplan algorithm [4]. The logistics problems seemed interesting because implementations of the Graphplan algorithm have not fared well in solving them, but control information very effectively rules out useless operator sequences.

The control formulae were discussed in Section 5. We replaced effects $l_1, \ldots, l_n \twoheadrightarrow \bot$ by preconditions $\overline{l_1} \vee \cdots \vee \overline{l_n}$ to allow IPP to use them in planning graph construction, and simplified the operators as discussed in Section 5. These transformations can easily be performed mechanically.

Table 1 gives runtimes of IPP and TLPlan and the formalization with the operator $\bigcirc$ on a Sun Ultra with a 360 MHz sparcv9 processor. We used two or three versions of each problem instance: without control information, with control formulae embedded in operators (/E), and interpreted with formula progression (/P). IPP does not interpret temporal logic formulae, so we could make a comparison between formula progression/regression and embedded control on TLPlan only. The /P runtimes are for the formalization of the problems in the TLPlan software distribution.

Table 1. Runtimes of two planners in seconds.

planner	log-a	log-b	log-c	log-d
TLPlan	$>$ 1 h	$>$ 1 h	$>$ 1 h	$>$ 1 h
TLPlan/E	0.17	0.13	0.17	0.52
TLPlan/P	0.45	0.33	0.54	1.94
IPP	1547.51	649.69	$>$ 22 h	$>$ 22 h
IPP/E	15.04	5.63	1010.55	30638.03

Without control TLPlan searches blindly and the chances of quickly reaching a goal state in a search tree with a depth of dozens of nodes are extremely small. TLPlan with embedded control is almost three times faster than TLPlan with control formulae interpreted with formula progression. An explanation for this difference is that formula progression and the use of defined predicates causes an overhead on top of the underlying forward-chaining planning algorithm. With embedded control this overhead disappears.

With TLPlan and the formalization with the operator $\mathcal{U}$ the runtimes are slightly higher (0.21, 0.14, 0.21, and 0.62 seconds) because of a higher number of preconditions and effects. Surprisingly, the corresponding IPP runtimes are higher than with no control at all (none of the runs terminated in 2 hours.) Also on the formalization with $\bigcirc$ IPP runtimes are high and it performs a lot of search (10^5 operator applications for logistics-b, down from 10^7 without control) even though the formulae almost uniquely tell what actions to take. Search cannot be completely avoided because an airplane may arrive to a goal city from several locations: the rules allow flights A-B-C and B-A-C if both A and B contain packages to be transported to C. However, as TLPlan traverses search trees with only 50 to 80 nodes, the difference is not explained by this.

A better explanation for IPP's performance is the handling of disjunctive subgoals that are obtained from effects $l_1, \ldots, l_n \twoheadrightarrow \bot$. IPP reduces disjunctive subgoals to non-disjunctive subgoals: first all minimal non-disjunctive subgoals that have at least one literal from each disjunction are produced, and then separately for each minimal non-disjunctive subgoal all sets of operators that produce it are tried out. There are examples in which the number of possible sets of operators is small (constant) but the number of alternative non-disjunctive subgoals is high (exponential), and removing disjunctivity before operator selection – which is not necessary – therefore increases the number of branches exponentially.[3] The logistics control formulae with $\mathcal{U}$ produce many more disjunctive subgoals than the formulae with $\bigcirc$, and this would explain the IPP runtimes.

7 RELATED WORK

Bacchus and Kabanza [2] pointed out the possibility of efficiently interpreting temporal logic formulae for pruning the search tree in a forward-chaining planner. The formulae are interpreted starting from the initial state. Assuming that the formulae are true at the current time point, the truth and falsity of several other formulae can be inferred for the next time point. These formulae restrict the possible choices of the next operators. This process of formula progression is not theorem-proving but a form of model-checking, and is computationally inexpensive. Our encoding of temporal logic formulae can be seen as an implicit way of doing progression. An important difference is that our encoding works with all planning algorithms, also ones that are not based on forward-chaining. The encoding can be automated fully and it is fairly efficient.

Huang et al. [7] extend a planning algorithm with temporal logic control. Many formulae can be handled during a preprocessing phase by not producing some of the ground instances of the operators. Other formulae require adding new clauses in the encoding of the problem in the propositional logic.

Baioletti et al. [3] consider a generalized notion of goals that are expressed as formulae with modal operators $\Box$ and $\Diamond$ and others. The formulae are encoded into plan operators like in our work. Baioletti et al. do not use operators $\bigcirc$ or $\mathcal{U}$ that seem to be necessary for formalizing control information.

Many forms of planning can be viewed as specialized problems of program synthesis – classical planning for example is synthesis of sequential non-branching programs from atomic programs that consist of assignment statements only – and hence the current work can be compared to work in that area. Temporal logic has been proposed as a framework for program synthesis [12], and it would be in principle possible to do planning completely in the temporal logic framework. Our approach is complementary: in this paper we are addressing approaches to planning that are not essentially based on reasoning in temporal logics, and therefore reducing temporal logic to the basic planning framework is beneficial.

The current work also has connections to specializing and transforming programs [8]. Program specialization typically attempts to

[3] For $(a_1 \vee b_1) \wedge \cdots \wedge (a_n \vee b_n)$ and operators respectively with effects $a_1 \wedge \cdots \wedge a_n$ and $b_1 \wedge \cdots \wedge b_n$ the increase is from 3 to 2^n.

make a given program more efficient by taking into account information concerning the possible input values. Program specialization has been performed for example by folding/unfolding transformations [5, 14, 11]. In our case, specialization is not based on restrictions on input values but on formulae that characterize the structure of the desired programs. Also, we do not start from a program that should be specialized. Instead, we transform a given program synthesis problem to a different one that can hopefully be solved more efficiently and from which a solution to the original problem can be easily extracted.

8 CONCLUSION

We have shown how to encode a class of temporal logic control formulae into plan operators so that explicit handling of formulae in planning algorithms is avoided. The class of formulae and the translation are interesting: control rules can easily be represented, plan size is not affected, and the sizes of operators increase relatively little. Our experiments show that our technique speeds up a backward-chaining planner substantially and can be competitive with formula progression as proposed by Bacchus and Kabanza [2].

The translation has syntactic restrictions but they are not violated for example by the formulae proposed by Bacchus and Kabanza (assuming a transformation to CNF.) A difference that may affect the formalization of control rules is the lack of defined predicates. Extending the translation to defined predicates is possible, but we have not pursued this question further in this work. Because of the complete independence of our approach of any planning algorithm, it is directly applicable to many kinds of algorithms. It may be possible and desirable to extend the translation to cover more general classes of formulae. It is not clear in what extent this is possible without sacrificing the good properties the current translation has. By inspecting the translation it would seem possible to allow disjunctions of literals in the subformulae of $\mathcal{U}$, $\mathcal{U}^2$ and $\bigcirc$. Furthermore, now there is an asymmetry between the source and target languages the source operators may not have conditional effects. Removing this asymmetry would be desirable.

In addition to the problem of using temporal logic control formulae, there is also the problem of automatically verifying that control formulae preserve the existence of (shortest) plans, and more generally, the problem of synthesizing control formulae automatically. A restricted but widely applicable type of control formulae is obtained by recognizing symmetries in planning problems [9]. For control formulae in general, already the problem of verifying whether a control formula preserves existence of plans is computationally very complex, PSPACE-hard in the propositional case, but it is likely that there are useful incomplete polynomial time algorithms – like for invariant synthesis [13], another PSPACE-hard problem – that may increase the efficiency and applicability of automated planning substantially.

REFERENCES

[1] C. Anderson, D. Smith, and D. Weld, `Conditional effects in Graphplan', in *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, eds., Reid Simmons, Manuela Veloso, and Stephen Smith, pp. 44–53. AAAI Press, (1998).

[2] Fahiem Bacchus and Froduald Kabanza, `Using temporal logics to express search control knowledge for planning', *Artificial Intelligence*, **116**(1–2), 123–191, (2000).

[3] Marco Baioletti, Stefano Marcugini, and Alfredo Milani, `Encoding planning constraints into partial order planning domains', in *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR '98)*, eds., A. G. Cohn, L. K. Schubert, and S. C. Shapiro, pp. 608–616, Trento, Italy, (1998). Morgan Kaufmann Publishers.

[4] Avrim L. Blum and Merrick L. Furst, `Fast planning through planning graph analysis', *Artificial Intelligence*, **90**(1-2), 281–300, (1997).

[5] R. M. Burstall and John Darlington, `A transformation system for developing recursive programs', *Journal of the ACM*, **24**(1), 44–67, (January 1977).

[6] E. Allen Emerson, `Temporal and modal logic', in *Handbook of Theoretical Computer Science*, ed., J. Van Leeuwen, volume B, 995–1072, Elsevier Science Publishers, (1990).

[7] Yi-Chen Huang, Bart Selman, and Henry Kautz, `Control knowledge in planning: benefits and tradeoffs', in *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99) and the Eleventh Conference on Innovative Applications of Artificial Intelligence (IAAI-99)*, pp. 511–517, Orlando, Florida, (1999). AAAI Press.

[8] N. D. Jones, C. K. Gomard, and P. Sestoft, *Partial Evaluation and Automatic Program Generation*, Prentice Hall, 1993.

[9] David Joslin and Amitabha Roy, `Exploiting symmetry in lifted CSPs', in *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97) and 9th Innovative Applications of Artificial Intelligence Conference (IAAI-97)*, pp. 197–202, Menlo Park, (July 1997). AAAI Press.

[10] Jana Koehler, Bernhard Nebel, Jörg Hoffmann, and Yannis Dimopoulos, `Extending planning graphs to an ADL subset', in *Proceedings of the Fourth European Conference on Planning (ECP'97)*, pp. 273–285. Springer-Verlag, (1997).

[11] Alberto Pettorossi and Maurizio Proietti, `Program specialization via algorithmic unfold/fold transformations', *ACM Computing Surveys*, **30**(3es), (September 1998).

[12] Amir Pnueli and Roni Rosner, `On the synthesis of an asynchronous reactive module', in *Automata, Languages and Programming, 16th International Colloquium*, eds., Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simona Ronchi Della Rocca, volume 372 of *Lecture Notes in Computer Science*, pp. 652–671, Stresa, Italy, (July 1989). Springer-Verlag.

[13] Jussi Rintanen, `An iterative algorithm for synthesizing invariants', in *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000) and the Twelfth Conference on Innovative Applications of Artificial Intelligence (IAAI-2000)*. AAAI Press, (2000).

[14] Hisao Tamaki and Taisuke Sato, `Unfold/fold transformation of logic programs', in *Proceedings of the Second International Logic Programming Conference*, ed., Sten-Åke Tärnlund, pp. 127–138, Uppsala, Sweden, (1984). Uppsala University.