Compilation of LTL goal formulas into PDDL

Stephen Cresswell and **Alexandra Coddington** [1]

Abstract. Temporally extended goals are used in planning to express safety and maintenance conditions. Linear temporal logic is the language often used to express temporally extended goals. We present a method for compiling LTL goal formulas into Planning Domain Definition Language (PDDL), which is handled by many AI planners. The compilation process first constructs a finite state machine representing all reachable progressions of the goal formula, then modifies the planning domain and problem definition so that the state of the FSM is tracked.

1 INTRODUCTION

In conventional formulations of planning problems, the goal condition is required to hold after the execution of the final action in the plan. It is also useful to be able to specify temporally extended goals, which requires more complex conditions to be satisfied by plans. Examples are safety conditions, e.g. the water must remain shut off while work is done on plumbing, and intermediate goals, e.g. a package must be moved to $loc2$ via $loc3$. In [2], these conditions are expressed using linear temporal logic (LTL) and used with a forward chaining planner. The same mechanism is also used to express control knowledge [3], resulting in a distinguished performance in the hand-coded track of the 3^{rd} International Planning Competition [11].

In this paper, we consider the question of whether extended goals written in LTL may be compiled directly into the representation of a planning domain and problem in the PDDL language [14]. This would allow such goals to be handled by many existing planners. An LTL formula expresses properties of an infinite sequence of future states and can be used to talk about properties of plans. LTL includes the temporal modalities *sometimes*, *always* and *until*. It is easy to encode simple cases of LTL goal conditions directly into PDDL, e.g. by the use of dummy operators. However, this opportunistic approach does not extend to LTL formulas in which the temporal modalities are nested. A simple example from [7] is $G(p \Rightarrow Fq)$, meaning *whenever p is true, q will be true at some subsequent moment*. This demonstrates the need for a more systematic approach to the translation.

In the following, we describe a method for the compilation which proceeds in two stages. First, the LTL goal formula is used generate a finite state machine (FSM). Then the PDDL planning domain is modified so that planning operators make the appropriate update to the state of the FSM alongside the change to the world state.

2 COMPILATION FROM LTL TO FSM

A full explanation of LTL is given by Emerson [7]. Here we give only a brief description. Formulas of LTL are interpreted over models of the form $M = \langle w_0, w_1, ... \rangle$ where M is an infinite sequence of worlds. The temporal modalities U, G, F, X describe properties of sequences.

Emerson notation	Interpretation	Definition
$P\,U\,Q$	P until Q	Q is true in some world w (now or future), P is true in all worlds before w.
$G(P)$	always P	P is true in all worlds (now and future)
$F(P)$	eventually P	P is true in some world (now or future)
$X(P)$	next P	P is true is next world

We do not use standard conversions of LTL to Büchi automata (e.g. [8]), as we wish to directly generate a deterministic automaton, whilst filtering using available information on which actions are reachable and which states are consistent. Our algorithm is based on exhaustive application of the *Progress* algorithm of Bacchus and Kabanza [3]. Given a world state (i.e. an assignment of truth values to propositions) and an LTL goal formula, *Progress* generates a new goal formula which applies to the next world. If the number of propositions occurring in the goal formula is small, we can systematically generate reachable worlds, and use those worlds to generate all possible progressions of a goal formula.

This results in a FSM in which the states are goal formulas, and the edges are sets of world states. We additionally annotate the edges with the partially-instantiated planning operators which bring about the transition. This information is needed when encoding the FSM into PDDL.

As an example, we consider a problem which requires that first goal $at2$ is achieved, then $at1$ is achieved and maintained.

$$F(at2 \wedge F(G(at1)))$$

We then find that our algorithm generates the set of five formulas shown below, forming the FSM shown in Fig. 1.

$$\begin{aligned} false \quad & (1)\\ true \quad & (2)\\ F(at2 \wedge F(G(at1))) \quad & (3)\\ F(G(at1)) \vee F(at2 \wedge F(G(at1))) \quad & (4)\\ G(at1) \vee F(G(at1)) \vee F(at2 \wedge F(G(at1))) \quad & (5) \end{aligned}$$

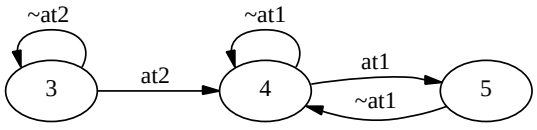

Figure 1. FSM for $F(at2 \wedge F(G(at1)))$

At the end of the plan, we can apply the assumption that there is no further change to world state, i.e. there follows an infinite sequence of identical world states. We can then reduce temporal modalities to truth values. In this example, this is the only way to reach $true$ or $false$.

[1] Department of Computer and Information Sciences, University of Strathclyde, Glasgow, United Kingdom. email:{sc, alex}@cis.strath.ac.uk

3 COMPILATION FROM FSM TO PDDL

We want to modify the planning operators so that they automatically track the current state in the FSM. The next FSM state depends on the current FSM state and the next world state, which is the world state *after* the effects of the operator have been applied. Unfortunately, an operator cannot perform tests on the next world state, so we have to test the combination of *transition* and current world state, where the transition depends on the operator and the choice of bindings. This is implemented by distinguishing each case by means of conditional effects. Below we consider how each feature of the FSM is represented in the translation into PDDL.

Edges in the FSM are represented as facts which are added to the initial state and are not changed by any operator. The edge fact includes symbols to represent the transition (e.g. `move-1-2`) and world state (e.g. `world0`), as well as start and end FSM states.

The Current state of FSM is represented by a dynamic fact, e.g. `(fsm_state n3)`. In the initial state, the `fsm_state` points to the node which represents the initial goal formula. This must be updated appropriately when operators are applied.

Accepting states of FSM. The goal condition depends on which combinations of world state together with FSM state are reduced to *true* under the assumption that all future worlds are identical. In general, the goal condition consists of a disjunction of world state tests together with accepting FSM states.

Updating FSM state Operators are modified by adding conditional effects to update the FSM state. A conditional effect is included for each combination of transition and relevant conditions on world state. This allows the next FSM state to be determined using the edge facts.

4 RELATED WORK AND DISCUSSION

The use of LTL in planning has mostly either been in a framework of forward state-space search [3], or model checking [9], [13].

LTL is more widely used in the formal verification of programs. The approach [10] is similar to ours, which uses an automata directly to verify temporal properties of Java programs. Nondeterministic Büchi automata are derived using a variant of the algorithm described in [8], and then converted to a form which is similar to ours.

The planners described in [3],[6] use temporal formulas as domain-specific control rules, i.e. to aid the search for a successful plan by pruning out sequences of actions which cannot be part of an efficient plan. Together with such control rules, forward state-space search can be strong enough for fast planning. By using only forward search, the progressions of temporal formulas can be performed on-the-fly, and the large number of reachable formulas is not an obstruction. For this reason, control rules make unrestricted use of quantification and of recursively-defined predicates. In [2], metric time is also handled, by means of an extended version of LTL called MITL.

Our approach is neutral about the architecture of the planner, but imposes some restrictions on the formulas that can be handled. To use our regime to control forward state space search, it would be helpful to add extra preconditions to operators to prevent FSM transitions to *false*. Some authors [1] have considered deriving extra operator preconditions for control purposes, but these correspond only to fixed forms of LTL formula. Our compilation offers a more general approach.

A desirable property for the compilation is that it should preserve plan metrics. In most cases this is true, but the mechanism of updating the FSMs can force ordering between actions which otherwise could have been placed in parallel. This is particularly problematic when the *next* modality is used in the goal formula. In the presence of *next*, we have the unwelcome behaviour that every action updates the FSM. This means that any action can interfere with any other action, and this leads to a totally ordered plan.

5 CONCLUSION AND FUTURE WORK

We have described a systematic method for compiling propositional LTL goal formulas into PDDL. A full version of this paper [5] gives a more detailed account, additionally describing how a restricted form of quantification is handled. The compilation allows LTL formulas to be handled by a wide range of planners.

It is not surprising that a compilation scheme is inferior to direct modification of a planner to handle LTL. However, we believe that our intermediate representation, the FSM, would be useful for incorporating an LTL capability into existing planning architectures. We plan to integrate this representation with the Graphplan architecture [4]. Since the fact layers of a planning graph give an explicit representation of state, an FSM state could be associated with each fact layer. The *next* modality does not cause problems in this architecture, as the notion of a next state is compatible with the planning graph representation. Additionally, the LPGP [12] approach to managing temporal constraints in Graphplan would allow the planner to deal with MITL, the version of LTL extended by Bacchus and Kabanza to express annotations using metric time.

REFERENCES

[1] F. Bacchus and M. Ady, 'Precondition control'. Unpublished manuscript, 1999.

[2] F. Bacchus and F. Kabanza, 'Planning for temporally extended goals', *Annals of Mathematics and Artificial Intelligence*, (1996).

[3] F. Bacchus and F. Kabanza, 'Using temporal logics to express search control knowledge for planning', *Artificial Intelligence*, **116**, (2000).

[4] A. Blum and M. Furst, 'Fast planning through planning graph analysis', in *Proc. of IJCAI-95*, pp. 1636–1642. Morgan Kaufmann, (1995).

[5] S. Cresswell and A. Coddington. Planning with LTL goal formulas via compilation into PDDL, 2004. Available from http://planning.cis.strath.ac.uk/.

[6] P. Doherty and J. Kvarnstrom, 'TALplanner: An empirical investigation of a temporal logic-based forward chaining planner', in *Proc. 6th Int'l Workshop on the Temporal Representation and Reasoning, TIME '99*, (1999).

[7] E. Allen Emerson, 'Temporal and modal logic', in *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, ed., J. van Leeuwen, 995–1072, North-Holland, (1990).

[8] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper, 'Simple on-the-fly automatic verifi cation of linear temporal logic', in *Proc. 15th IFIP WG6.1 Int'l Symp. on Protocol Specification, Testing and Verification*, (1996).

[9] G. De Giacomo and M. Y. Vardi, 'Automata-theoretic approach to planning for temporally extended goals', in *ECP '99*, (1999).

[10] D. Giannakopoulou and K. Havelund, 'Automata-based verifi cation of temporal properties on running programs', in *16th IEEE International Conference on Software Engineering*, (2001).

[11] D. Long and M. Fox, 'The 3rd international planning competition: Results and analysis', *JAIR*, **20**, (2003).

[12] D. Long and M. Fox, 'Exploiting a graphplan framework in temporal planning', in *Int'l Conf. on Automated Planning and Scheduling, ICAPS 2003, Trento, Italy*, (2003).

[13] M. Cialdea Mayer, A. Orlandini, G. Balestreri, and C. Limongelli, 'A planner fully based on linear time logic', in *5th Int'l Conf. on AI Planning and Scheduling (AIPS-2000)*, pp. 347–354, (2000).

[14] D. McDermott et al., 'PDDL — the planning domain defi nition language', Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, (1998).