Research Workshop of the
Israel Science Foundation

Proceedings of the 2nd Workshop on

Model Checking and Automated Planning (MOCHAP-15)

Edited By:

Sergiy Bogomolov, Daniele Magazzeni, Martin Wehrle

Jerusalem, Israel 7/6/2015

Organizing Committee

Sergiy Bogomolov
IST Austria, Austria

Daniele Magazzeni
King's College London, UK

Martin Wehrle
University of Basel, Switzerland

Program Committee

Foreword

There has been a lot of work on the exchanges between the two research areas of model checking and automated planning. From a high level perspective, model checking and planning problems are related in the sense that plans (found by a planning system) correspond to error traces (found by a model checker), and vice versa. The two paradigms of “planning via model checking” and “directed model checking” are now widely used in different planning and verification domains.

The purpose of the workshop on Model Checking and Automated Planning (MOCHAP) is to continue to promote a cross-fertilisation between research on planning and verification, incrementing the synergy between the two areas.

After the successful first edition of the workshop, held at ICAPS 2014, this year MOCHAP-15 featured again a very rich program. Topics include planning in nondeterministic domains, planning with LTL, symmetry and partial order reduction, diagnosis and guided search on hybrid systems. Furthermore, two notable researchers have accepted our invitation to complete the program: Paolo Traverso, with a talk on "*20 Years of Planning via Model Checking: From Theory to Practice*", and Doron Peled, with a talk on "*Commutativity based search*".

We thank the members of the Program Committee for their dedicated effort in ensuring the quality of the papers presented at MOCHAP-15. We thank the invited speakers and all the authors for presenting their work and for contributing to a successful event.

Sergiy Bogomolov, Daniele Magazzeni, Martin Wehrle
MOCHAP-15 Chairs

Table of Contents

20 Years of Planning via Model Checking: From Theory to Practice

Paolo Traverso
FBK-ICT irst, Centre for Information Technology
I-38123 Trento - Italy

Abstract

Planning via Model Checking is nowadays a well-known technique. Techniques based on model checking have been successfully applied for dealing with different kinds of planning problems. In classical planning domains, LTL based model checking has been used to guide the search towards the goal. Classical planners based on BDDs have participated to the international planning competition since its early editions. Techniques based on both explicit state and symbolic model checking have been used to address the problem of planning under uncertainty, including planning with full observability in non-deterministic domains (FOND), planning with partial observability (POND), conformant planning, and planning in non-deterministic domains with temporally extended LTL and CTL goals. Techniques for interleaving planning via model checking and partial plan execution have been explored. Model checking techniques have also been used for planning with preferences, and planning in asynchronous domains. Recently, planning via model checking has been successfully applied to hybrid systems.

In my talk, I will review some of the different approaches in planning via model checking. I will then discuss some applications in different domains, e.g., applications for safety critical systems, web services and business processes, the dynamic management of harbor facilities, and planning for services in the field of smart cities and communities. I will discuss some lessons learned and some future challenges for the practical application of planning via model checking.

Stubborn Sets for Fully Observable Nondeterministic Planning

Dominik Winterer and **Robert Mattmüller**
University of Freiburg, Germany
{wintered, mattmuel}@informatik.uni-freiburg.de

Martin Wehrle
University of Basel, Switzerland
martin.wehrle@unibas.ch

Abstract

The stubborn set method is a state-space reduction technique, originally introduced in model checking and then transfered to classical planning. It was shown that stubborn sets significantly improve the performance of optimal deterministic planners by considering only a subset of applicable operators in a state. Fully observable nondeterministic planning (FOND) extends the formalism of classical planning by nondeterministic operators. We show that stubborn sets are also beneficial for FOND problems. We introduce nondeterministic stubborn sets, stubborn sets which preserve strong cyclic plans. We follow two approaches: Fast Incremental Planning with stubborn sets from classical planning and LAO* search with nondeterministic stubborn sets. Our experiments show that both approaches increase coverage and decrease node generations when compared to their respective baselines.

Introduction

Classical planning is the problem of finding a sequence of actions leading from a specified initial state to some goal state. Whereas in classical planning outcomes of actions are uniquely determined, fully observable nondeterministic planning (FOND) permits actions whose outcomes are uncertain. Such nondeterministic actions can be used to model, e.g., the failure of an agent's action. While this is often addressed by re-planning, strong cyclic plans—trial-and-error strategies—empower the agent to solve failure situations without re-planning.

Recently, research in classical planning has shifted towards techniques orthogonal to heuristics such as partial order reduction which has been transfered from computer aided verification (Valmari 1989; Godefroid 1995) to optimal deterministic planning (Alkhazraji et al. 2012). Further research aimed at improving the efficiency of stubborn set computation and determining a generalized definition of stubborn sets (Wehrle and Helmert 2014). We address the stubborn set method combined with two algorithms, Fast Incremental Planning (FIP) and LAO*.

Fast Incremental Planning is an algorithm for strong cyclic planning which solves FOND problems within multiple runs of an underlying classical planner (Kuter et al. 2008). Planner for Relevant Policies (PRP) combines this idea with a regression search to generalize the policy and substantially outperforms FIP (Muise, McIlraith, and Beck 2012). Our first step towards estimating the potential of stubborn sets for FOND planning is to use FIP with an underlying classical planner in combination with stubborn sets from classical planning. However, the main drawback of such determinization approaches is that they may find poor solutions, e.g., strong cyclic plans with high expected costs.

LAO* (Hansen and Zilberstein 2001), originally proposed to solve MDPs, is an algorithm for strong cyclic planning, which finds strong cyclic solutions in the nondeterministic state space. Using an admissible heuristic estimator, it finds strong cyclic plans of minimal expected costs. It has been shown that combining LAO* with pattern database heuristics (Mattmüller et al. 2010) is a successful approach to solving FOND problems. Our contribution is a stubborn set formalism for nondeterministic state spaces, that preserves strong cyclic plans. We evaluated both approaches, FIP with stubborn sets from classical planning and LAO* with our new formalism. Our results show that both approaches increase coverage and reduce node generations when compared to their respective baselines without stubborn sets.

Preliminaries

We use an SAS^+ based notation (Bäckström and Nebel 1993) to model fully observable nondeterministic planning problems. States of the world are described by a finite set of state variables $\mathcal{V}$. Every variable $v \in \mathcal{V}$ has an associated finite domain $\mathcal{D}_v$ and an extended domain $\mathcal{D}_v^+ = \mathcal{D}_v \uplus \{\perp\}$ where $\perp$ defines the undefined value. A *partial state* is a function s with $s(v) \in \mathcal{D}_v^+$ for all $v \in \mathcal{V}$. We write $vars(s)$ for the set of all v with $s(v) \neq \perp$. A partial state is a *state* if $vars(s) = \mathcal{V}$.

Definition 1 (nondeterministic planning task). *A nondeterministic planning task is a 4-tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$, where $\mathcal{V}$ is a finite set of finite-domain variables, $\mathcal{O}$ is a finite set of nondeterministic operators, s_0 is a state called the initial state and s_* is a partial state called the goal. Each nondeterministic operator $o = \langle Pre \mid Eff \rangle$ has a partial state Pre called precondition, a finite set of partial states Eff and an associated non-negative number $cost(o)$ called its cost.*

An operator o is *applicable* in a state s if Pre is satisfied in s. The application of a single effect $\mathit{eff} \in \mathit{Eff}$ in s yields the state $app(\mathit{eff}, s)$ that results from updating the values of s with the ones of eff. The application of o to a state s yields the set of states $o(s) := \{app(\mathit{eff}, s) \mid \mathit{eff} \in \mathit{Eff}\}$. The set of applicable operators in a state s is denoted by $app(s)$. Sometimes we want to refer to a particular outcome of an operator $o = \langle \mathit{Pre} \mid \{\mathit{eff}_1, \cdots, \mathit{eff}_k\}\rangle$. The *determinization* of nondeterministic operator o is $o^{[1]}, \cdots, o^{[k]}$ with every outcome $o^{[i]} = \langle \mathit{Pre} \mid \{\mathit{eff}_i\}\rangle$. The *all-outcomes determinization* of planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_*\rangle$ is $\Pi_{det} = \langle \mathcal{V}, \mathcal{O}_{det}, s_0, s_*\rangle$ where $\mathcal{O}_{det}$ is the set of all operator outcomes of $\mathcal{O}$.

An operator is deterministic if $|\mathit{Eff}| = 1$. It is nondeterministic if $|\mathit{Eff}| \geq 2$. We say a planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_*\rangle$ is deterministic if all of its operators are deterministic. We refer to all variables in the precondition of an operator o as $prevars(o) = vars(\mathit{Pre})$ and to all variables in its effects as $\mathit{effvars} = \bigcup_i vars(\mathit{eff}_i)$.

A solution to a FOND planning task Π with set of states S is a *policy* $\pi : S \to \mathcal{O} \cup \{\bot\}$, which maps a state to an appropriate action or is undefined, e.g. $\pi(s) = \bot$. Policy π is *weak* if it defines at least one path from the initial state to a goal state following it. It is *closed* if following it either leads to a goal state or to a state where the policy is defined. Policy π is *proper* if from every state visited following it there exits a path to a goal state following it. Policy π is *acyclic* if it does not revisit already visited states.

Definition 2 (weak plan, strong cyclic plan, strong plan)**.** *Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_*\rangle$ be a planning task.*

- *A policy for Π is called a weak plan for Π if it is weak.*
- *A policy for Π is called a strong cyclic plan for Π if it is closed and proper.*
- *A policy for Π is called a strong plan for Π if it is closed proper and acyclic.*

A weak plan is a sequence of actions which leads to the goal if all nondeterministic operator outcomes were deterministic. It corresponds to a plan in classical planning. A strong plan guarantees that after a maximum number of steps a goal state is reached. Strong cyclic planning relaxes that property requiring that the goal is reached within a finite sequence of actions. We want to emphasize that the nondeterminism in FOND planning is not necessarily the same as in model checking with nondeterministic models since unlike strong cyclic plans, counterexamples in model checking are linear sequences.

Deterministic Stubborn Sets

The first step towards stubborn sets is the definition of operator interference. We follow the definition of Wehrle and Helmert (2014).

Definition 3 (interference of deterministic operators)**.** *Let o_1 and o_2 be operators of a deterministic planning task Π and let s be a state of Π. Operators o_1 and o_2 interfere in s if they are both applicable in s, and*

- *o_1 disables o_2 in s, i.e., $o_2 \notin app(o_1(s))$, or*
- *o_2 disables o_1 in s, i.e., $o_1 \notin app(o_2(s))$, or*
- *o_1 and o_2 conflict in s, i.e., $s_{12} = o_1(o_2(s))$ and $s_{21} = o_2(o_1(s))$ are both defined and differ: $s_{12} \neq s_{21}$.*

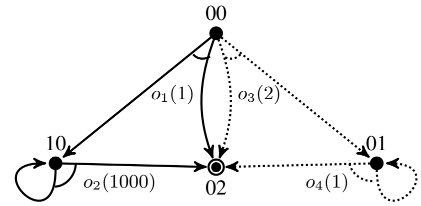


Figure 1: Solid: expensive strong cyclic solution. Dotted: cheap strong cyclic solution. Determinization-based algorithms might not find the cheap solution.

We approximate deterministic operator interference, by considering it globally for any state s. According to this syntactic notion of interference, two deterministic operators o_1 and o_2 *interfere* if the effect of o_1 violates the precondition of o_2 (or vice versa) or if o_1 and o_2 have a common variable in their effects which they set to different values. Furthermore, we consider that operators which are never jointly applicable cannot interfere. This is done by checking whether the preconditions of two operators o_1 and o_2 are mutually exclusive (Wehrle and Helmert 2014). For stubborn sets we need two more definitions. A *disjunctive action landmark* (DAL) in state s is a set of operators such that all operator sequences leading from s to a goal state contain some operator in the set. A *necessary enabling set* (NES) for operator o in state s is a set of operators such that all operator sequences that lead from s to some goal state and include o contain some operator in the NES before the first occurrence of o. Both sets can be computed by selecting a variable v whose value differs from either the goal or the precondition of the operator to enable. Then, we add each operator which achieves the desired value of v. As both sets are not uniquely determined, the pruning power and size of stubborn sets depends on their choices (Wehrle and Helmert 2014).

Definition 4 (deterministic strong stubborn set)**.** *Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_*\rangle$ be a deterministic planning task and s a state. A set $T_s \subseteq \mathcal{O}$ is a deterministic strong stubborn set (DSSS) in s if the following conditions hold:*

1. *T_s contains a disjunctive action landmark in s.*
2. *For all operators $o \in T_s$ with $o \notin app(s)$, T_s contains a necessary enabling set for o in s.*
3. *For all operators $o \in T_s$ with $o \in app(s)$, T_s contains all operators that interfere with o in s.*

We use FIP combined with an underlying classical planner using deterministic stubborn sets. Solving FOND problems with classical planners can lead to costly strong cyclic plans. Although optimality does not play the major role in FOND planning, the possibility of finding arbitrarily bad solutions is undesirable. We show that exactly this might happen.

Example 1. Consider a nondeterministic planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_*\rangle$ with variables $\mathcal{V} = \{v_1, v_2\}$ and the following operators:

- $o_1 = \langle v_1 = 0 \mid \{v_1 := 1\}, \{v_2 := 2\}\rangle$
- $o_2 = \langle v_1 = 1, v_2 = 0 \mid \{\top\}, \{v_1 := 0, v_2 := 2\}\rangle$
- $o_3 = \langle v_1 = 0 \mid \{v_2 := 2\}, \{v_2 := 1\}\rangle$
- $o_4 = \langle v_1 = 0, v_2 = 1 \mid \{\top\}, \{v_2 := 2\}\rangle$

As cost function we have $cost : \{o_1 \mapsto 1, o_2 \mapsto 1000, o_3 \mapsto 2, o_4 \mapsto 1\}$, the initial state is $s_0 = \{v_1 \mapsto 0, v_2 \mapsto 0\}$ and the goal $s_* = \{v_2 \mapsto 2\}$. Assume we perform a run of the FIP algorithm and its first weak plan would be $o_1^{[2]}$ inducing the fail-state $o_1^{[1]}(s_0) = 10$. In a subsequent weak plan search, the algorithm considers both outcomes of o_2 and adds them to the policy. This yields a clearly non-optimal strong cyclic plan, whereas the optimal solution consists of o_3 and o_4 (Figure 1). Applying PRP to this example gives the same solution, since regressing $o_1^{[2]}$ is ineffective.

Nondeterministic Stubborn Sets

Reducing FOND problems to multiple classical planning problems sometimes leads to poor strong cyclic solutions since the individual runs of classical planners only guarantee good weak plans which are not always part of a good strong cyclic plans. To overcome this, it can be beneficial to plan in the nondeterministic state space e.g., with LAO* search (Hansen and Zilberstein 2001) which finds strong cyclic plans with minimum expected costs under certain assumptions. Planning in the nondeterministic state space needs new definitions of stubborn sets and operator interference since the former do not consider nondeterministic operators.

For a given nondeterministic planning problem Π, a straightforward approach would be to directly apply the original definition of strong stubborn sets on the all-outcome-determinization of Π, and additionally, to add for every outcome $o^{[i]}$ of a nondeterministic operator o every other outcome of o in order to respect o's nondeterministic nature. However, as the following example shows, such an approach is incomplete.

Example 2. Consider the following all-outcomes determinization $\Pi_{det} = \langle \mathcal{V}, \mathcal{O}_{det}, s_0, s_* \rangle$ of nondeterministic planning task Π with variables $\mathcal{V} = \{v_1, v_2\}$ and the following operators:

- $o_1^{[1]} = \langle v_1 = 0 \mid \{v_1 := 1\}\rangle$, $o_1^{[2]} = \langle v_1 = 0 \mid \{v_1 := 2\}\rangle$
- $o_2^{[1]} = \langle v_2 = 0 \mid \{v_2 := 1\}\rangle$, $o_2^{[2]} = \langle v_2 = 0 \mid \{v_2 := 2\}\rangle$
- $o_3^{[1]} = \langle v_2 = 0 \mid \{v_2 := 3\}\rangle$, $o_3^{[2]} = \langle v_2 = 0 \mid \{v_2 := 4\}\rangle$
- $o_{11} = \langle v_1 = 1, v_2 = 1 \mid \{v_2 := 5\}\rangle$
- $o_{12} = \langle v_1 = 1, v_2 = 2 \mid \{v_2 := 5\}\rangle$
- $o_{23} = \langle v_1 = 2, v_2 = 3 \mid \{v_2 := 5\}\rangle$
- $o_{24} = \langle v_1 = 2, v_2 = 4 \mid \{v_2 := 5\}\rangle$

The initial state is $s_0 = \{v_1 \mapsto 0, v_2 \mapsto 0\}$, and the goal is $s_* = \{v_2 \mapsto 5\}$. The set $\{o_{11}, o_{12}, o_{23}, o_{24}\}$ is a disjunctive action landmark in s_0 which we add to the candidate set T_{s_0}. As all operators in this set are inapplicable in s_0, we have to add a necessary enabling set for all of them. A valid choice for these necessary enabling sets is based on selecting the unsatisfied conditions $v_2 = 1$, $v_2 = 2$, $v_2 = 3$ and

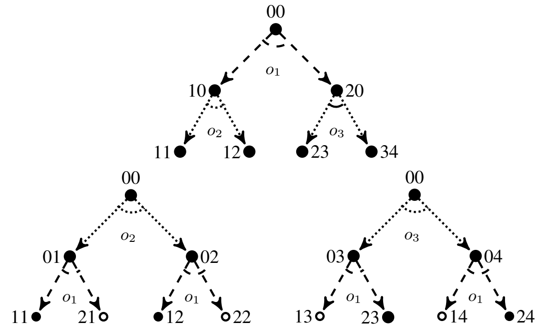


Figure 2: Postponing o_1 in s_0 only leads to policies with dead-ends. The straightforward instantiation is incomplete.

$v_2 = 4$ in the preconditions of o_{11}, o_{12}, o_{23}, o_{24}, respectively, and to add the determinized operators that set these conditions to true. These achieving operators correspond to all outcomes of o_2 and o_3, which are applicable in s_0 but non-interfering with any operator not in T_{s_0}. Hence, we finally get $T_{s_0} = \{o_{11}, o_{12}, o_{23}, o_{24}, o_2^{[1]}, o_2^{[2]}, o_3^{[1]}, o_3^{[2]}\}$.

However, T_{s_0} is insufficient for our purpose because every strong plan from s_0 has to start with o_1: Depending on the nondeterministic outcome of o_1 ($v_1 = 1$ or $v_1 = 2$), o_2 or o_3 can be applied to satisfy the precondition of an operator to reach the goal. In contrast, starting with o_2 and applying o_1 afterwards might lead to outcomes where no goal is reachable any more (e.g., $v_1 = 2$ and $v_2 = 2$). The analogous situation occurs when starting with o_3 and applying o_1 afterwards (Figure 2).

The core problem of our straightforward instantiation is that deterministic operator interference is an insufficient criterion for nondeterministic operators. Because changing the order of two non-interfering nondeterministic operators o and o' in a strong cyclic plan results in, e.g., outcomes of o' getting prefixes of weak plans started by o. While this is not an issue for all weak plans which contain outcomes of both operators, it is problematic to those weak plans which start with an outcome of o but do not contain an outcome of o'. A solution to this is to demand that such prefixes preserve the original weak plan which we address with the following property.

Definition 5 (prefix-compatibility)**.** *Let Π be a planning task and $\Pi_{det} = \langle \mathcal{V}, \mathcal{O}_{det}, s_0, s_* \rangle$ its all-outcomes determinization. Two operators $o_1, o_2 \in \mathcal{O}_{det}$ are* prefix compatible *if for all operator sequences π_1 and π_2:*

- *$o_1\pi_1$ is a weak plan implies $o_2o_1\pi_1$ is also a weak plan and*
- *$o_2\pi_2$ is a weak plan implies $o_1o_2\pi_2$ is also a weak plan*

Intuitively, two operators o_1 and o_2 are prefix compatible if every weak plan starting with o_1 is preserved if we put o_2 to its front and vice versa. Equipped with prefix compatibility, we can formulate the definition of a stubborn set for the nondeterministic state space which has two additional rules compared to the DSSS definition.

Definition 6 (nondeterministic strong stubborn set)**.** *Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ be a nondeterministic planning task,*

$\Pi_{det} = \langle \mathcal{V}, \mathcal{O}_{det}, s_0, s_* \rangle$ *its all-outcomes determinization and* s *a state. A set* $T_s \subseteq \mathcal{O}_{det}$*, is a nondeterministic strong stubborn set (NSSS) in* s *if the following conditions hold:*

1. T_s *contains a disjunctive action landmark in* s *for* Π_{det}.
2. *For all operators* $o \in T_s$ *with* $o \notin app(s)$, T_s *contains a necessary enabling set for* o *in* s *for* Π_{det}.
3. *For all operators* $o \in T_s$ *with* $o \in app(s)$, T_s *contains all operators that interfere with* o *in* s *for* Π_{det}.
4. *For every outcome* $o^{[i]} \in T_s$ *of nondeterministic operator* o, T_s *contains all operators that are not prefix compatible with* o.
5. *For every outcome* $o^{[i]} \in T_s$ *of nondeterministic operator* o, T_s *contains all other outcomes of* o.

Proposition 1. *Nondeterministic strong stubborn sets preserve completeness for strong cyclic planning.*

Proof. At first we show completeness for strong planning then we show it for strong cyclic planning. Let π be a strong plan from state s that induces weak plans π_i and π_j, such that there is state $\tilde{s}$ with $\pi_i(\tilde{s}) = o^{[i]}$ and $\pi_j(\tilde{s}) = o^{[j]}$. Weak plans π_i and π_j have the following structure: $\pi_i = \alpha o^{[i]} \beta_i$ and $\pi_j = \alpha o^{[j]} \beta_j$ where α is a common operator sequence without outcomes of nondeterministic operators, and β_i, β_j contain also outcomes of nondeterministic operators respectively. State $\tilde{s}$ is the branching point of π_i and π_j. Let k_i be the smallest index such that operator $o_{k_i} \in \pi_i$ is contained in the nondeterministic stubborn set T_s, similarly for k_j and π_j. We distinguish the following three cases.

(1) $o_{k_i} \in \alpha = o_1 \cdots o_n$. Clearly $o_{k_i} = o_{k_j}$. o_{k_i} is applicable since otherwise a necessary enabling set has to be contained in T_s and at least one operator has to be applied before o_{k_i}, contradicting the choice of k_i. Since k_i is the smallest index such that $o_{k_i} \in T_s$, o_{k_i} does not interfere with any operator of smaller index because otherwise an operator applied before o_{k_i} must be contained in T_s. Also, this contradicts the choice of k_i. Thus we can replace α by $o_{k_i} o_1 \cdots o_{k_i-1} o_{k_i+1} \cdots o_n$.

(2) $o_{k_i} = o^{[i]}$. $o_{k_j} \notin \alpha$ since otherwise $o_{k_i} \in \alpha$. Also, o_{k_j} cannot be in β_j because by the definition of the NSSS $o^{[j]} \in T_s$. It follows that $o_{k_j} = o^{[j]}$. Like in case (1) o_{k_i} is applicable and does not interfere with operators of smaller index for π_i, the same holds for o_{k_j} and π_j. Thus we can move the nondeterministic operator o to the front resulting in $o^{[i]} \alpha \beta_i$ and $o^{[j]} \alpha \beta_j$.

(3) $o_{k_i} \in \beta_i = o_{n+2} \cdots o_{n+m_i}$. o_{k_j} cannot be in α since otherwise $o_{k_i} \in \alpha$. Also, $o_{k_j} \neq o^{[j]}$ since otherwise by definition of the NSSS, $o^{[i]}$ would be included in T_s, contradicting the choice of o_{k_i}. Therefore $o_{k_j} \in \beta_j$. Let $s_1 \cdots s_{n-1} \tilde{s} s_{n+1} \cdots s_{n+m_i}$ be the states visited by π_i. We know as in case (1) that o_{k_i} does not interfere with operators of smaller index. Inductively it follows that o_{k_i} is applicable in $\tilde{s}$. Also we know that o_{k_i} and o are prefix compatible since otherwise $o \in T_s$. This means that $o^{[i]} o_{n+2} \cdots o_{n+m_i}$ is a weak plan from $o_k(\tilde{s}) = o_k(o_n \cdots (o_1(s_0)))$. From the non-interference of o_k with operators of smaller index we get $o_k(o_n \cdots (o_1(s_0))) = o_n(o_{n-1} \cdots (o_1(o_k(s_0))))$. Hence we can move o_k to the front of π_i and π_j. If o_k is an outcome of a nondeterministic operator o' then it has a sibling o'^{l} which is the smallest index of weak plan π_l. This case is covered by case (2) with π_l and π_i.

Since a strong plan is a strong cyclic plan without cycles we just have to consider the effect of cycles. NSSS is state dependent but not path dependent, therefore revisiting some state s does not affect T_s, concluding the proof. □

Nondeterministic stubborn sets are in general not optimality preserving for strong cyclic planning since prefix compatibility leads to operators being added in front of other ones which can lead to solutions with higher expected costs.

Approximating Prefix Compatibility

The exact notion of prefix compatibility is intractable to compute because we would have to consider all weak plans. Therefore we outline how to find a sufficient criterion for prefix-compatibility. We define $Dis(o)$ as the set of operator-variable pairs $(o', v) \in \mathcal{O}_{det} \times \mathcal{V}$ such that o disables o' on variable v in any state. Further we define $Neg(o)$ as the set of goal variables with which o conflicts, i.e., $\mathit{eff}(o)[v] \neq s_*[v]$ for goal-related variables v on which o has an effect. If $Dis(o_1) = Dis(o_2)$ and $Neg(o_1) = Neg(o_2)$ then o_1 and o_2 are prefix compatible. The idea behind this is: if two operators o_1 and o_2 disable the same set of operators on the same variables, then every deterministic operator sequence starting with o_1 remains applicable if we append o_2 to its front. Also weak plans are preserved since o_1 and o_2 do not violate different goal variables.

In some cases, we can weaken this syntactic notion of prefix compatibility. Consider two non-interfering operators $o_1 = \langle Pre \mid \{\mathit{eff}'_1\} \rangle$ and $o_2 = \langle Pre \mid \{\mathit{eff}_1, \cdots, \mathit{eff}_n\} \rangle$. If $\sigma = \{s \mapsto o_1, o_1(s) \mapsto o_2\}$ is a subsequence of a strong cyclic plan from state s then exchanging the order of o_1 and o_2 gives an equivalent subsequence since they induce the same set of states, i.e., $o_1(o_2^{[i]}(s))) = o_2^{[i]}(o_1(s))$ for all $i \leq n$. Therefore if two such operators do not interfere, it suffices to check $Dis(o_1) \subseteq Dis(o_2)$ and $Neg(o_1) \subseteq Neg(o_2)$.

Sometimes nondeterministic operators contain only one nontrivial effect, i.e. an operator $o = \langle Pre \mid \{\mathit{eff}_1\}, \{\top\} \rangle$. For every weak plan $o^{[1]}\pi$ from state s, it exists a finite sequence $\sigma = o^{[2]} \cdots o^{[2]}$, repeated applications of o's trivial effect, such that $\sigma o^{[1]} \pi$ is also a weak plan from s. Thus, every operator being a prefix of $o^{[1]}\pi$ preserving the weak plan, does also preserve $\sigma o^{[1]} \pi$. Such operators are therefore trivially prefix compatible to any other deterministic operator.

Efficient Computation

As nondeterministic stubborn sets leave open how the disjunctive action landmark and the necessary enabling sets were chosen, the pruning power of stubborn sets depends highly on these design choices. We outlined how prefix compatibility can be syntactically addressed. However, for applicable nondeterministic operators with more than one nontrivial effect, in the stubborn set we have to add both the interfering and the non-prefix compatible operators. This leads to many operators being added to the stubborn set. It is therefore reasonable to avoid applicable nondeterministic operators with more than one nontrivial effect from being added to the stubborn set. Let $nontrivial$ be the set of operators with more than one nontrivial effect. Our intention is to exclude applicable operators of $nontrivial$ from being added to the stubborn set. We address this by computing a weight whenever we have to add a DAL or NES to the stubborn set. We calculate a weight for each DAL or NES and chose the DAL or NES with lowest weight according to:

$$weight(o, s, T_s) = \begin{cases} \infty, & \text{if } o \in app(s) \wedge o \in nontrivial \\ K, & \text{if } o \in app(s) \wedge o \notin nontrivial \\ 1, & \text{otherwise} \end{cases}$$

where o is an operator not in stubborn set T_s and K a nonzero natural number. Our *exclude* strategy is an extension of a strategy presented by Laarman et al. (2013) which penalizes applicable operators not in the current candidate stubborn set. A coarser strategy towards prefix compatibility for nontrivial operators is to simply assume that an applicable nontrivial operator is not prefix compatible to all other operators. On par with the exclude strategy this is feasible since it avoids the costly computation of the disabling relation.

A Tighter Envelope

Active operators (Chen and Yao 2009; Wehrle et al. 2013) approximate the set of operators which can be part of any weak plan from some state using domain transitions graphs (DTGs). From a more general point of view, Wehrle et al. (2013) denote subsets which preserve at least one weak plan from some state as an *envelope*. Combining a tight envelope with stubborn sets may not only exclude operators which are not in envelope E from the stubborn set but also prevent cascades from being added to the stubborn set. Of course, the active operators can also be used for strong cyclic planning since strong cyclic plans consist of multiple weak plans. We additionally exploit the structure of strong cyclic plans and obtain a tighter envelope.

A *part-of-a-plan operator* $o \in \mathcal{O}$ in s is a deterministic operator that is contained in some weak plan starting from s. This notion is intractable to compute so we have to find a sufficient criterion for it.

Definition 7 (active operator). *Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ be a deterministic planning task. An active operator $o \in \mathcal{O}$ in a state s is an operator that satisfies the following conditions:*

1. *For every variable $v \in prevars(o)$, there is a path in $DTG(v)$ from $s[v]$ to $pre(o)[v]$, and also from $pre(o)[v]$ to the goal value $s[v]$ if v is goal-related.*
2. *For all $v \in effvars(o) \cap vars(s_*)$ there is a path in $DTG(v)$ from $eff_1(o)[v]$ to $s_*[v]$.*

Intuitively, the definition states that an operator is active if it is part of some weak plan from the corresponding abstracted state in every singleton abstraction of Π. Thus, a part-of-a-plan operator is always active but not vice versa.

A *nondeterministic part-of-a-plan operator* $o \in \mathcal{O}$ in s is an operator that is contained in some strong plan π starting from s. Like for the part-of-a-plan operators this is intractable to compute.

Proposition 2. *Let o be an applicable nondeterministic operator and $o^{[i]}$ one of its outcomes. If $o^{[i]}$ is inactive in state s then o cannot be part of any strong cyclic plan from s.*

Proof. We show this by contradiction. If there were a strong cyclic plan π from s, such that $\pi(\tilde{s}) = o$ for some state $\tilde{s}$ but $o^{[i]}$ is no part-of-a-plan operator in s. Let further $\sigma = o_1 \cdots o_n$ be a deterministic operator sequence applicable in s that leads to $\tilde{s}$. Since $o^{[i]}$ is no part-of-a-plan operator in s, no weak plan from s does contain $o^{[i]}$. Therefore $\sigma o^{[i]} \pi'$ for an arbitrary operator sequence π' cannot be a weak plan from s. This implies that π' is no weak plan from $o^{[i]}(\tilde{s})$. Thus π is not proper since $o(\tilde{s})$ is reachable following π but there is no goal state reachable from $o(\tilde{s})$. This contradicts π being a strong plan. It follows that if any outcome of a nondeterministic operator o is not active in s, then o cannot be part of a strong cyclic plan from s. □

We denote our new envelope by *nondeterministic active envelope*. It can be used for both the DSSS and NSSS.

Experimental Evaluation

We focused our experimental evaluation on the following two configurations:

1. FIP combined with DSSSs
2. LAO* combined with NSSSs

We further investigated the impact of different envelopes: *full, active, nondeterministic active* (Table 1 and Table 2). Also, we varied the approximation of prefix compatibility for the NSSS approach (Table 3). We differentiate between the approach which assumes that every nontrivial operator is not prefix compatible with all other operators (*no prefix*) and the approach where prefix compatibility is syntactically approximated (*syntactic*). For the DSSS, disjunctive action landmarks and necessary enabling sets were computed using the *laarman* strategy. For the NSSS we used the *exclude* strategy. The interference relation for both the DSSSs and NSSSs is entirely precomputed which is also true for the achievers, the NSSSs need the additional precomputation of the disabling relation. For the underlying classical planner of FIP, we used greedy best first search. As heuristic estimator, we chose the FF heuristic (Hoffmann and Nebel 2001) for all approaches.

We evaluated both stubborn set approaches on all FOND domains of the IPC-2008 and variations of these. Furthermore we added two domains from probabilistic planning to our benchmark set.[1] All experiments were conducted on a server equipped with AMD Opteron 2.3 GHz CPUs. We set

[1] *First-Responders-new* consists of larger instances of the First-Responders domain. *Forest-new* is taken by Muise, McIlraith, and Beck (2012). *Tidyup* is the *Mobile Manipulation* domain of Hertle et al. (2014) adapted for FOND planning. *Earth-Observation* was introduced by Aldinger and Löhr (2013).

	Coverage				Node Generations			
Domain	FIP	DSSS	DACT	NACT	FIP	DSSS	DACT	NACT
FR(75)	**74**	–1	–1	–1	2917345	45.71%	45.71%	**45.65%**
FR-NEW(91)	**75**	–1	–1	–1	9849763	65.3%	65.3%	**65.29%**
FOREST-NEW(90)	16	**+4**	**+4**	**+4**	16463519	**10.94%**	**10.94%**	**10.94%**
FOREST(90)	13	**+4**	**+4**	**+4**	28857372	**18.96%**	**18.96%**	**18.96%**
EARTH(40)	**35**	–2	–2	–2	**5592159**	**100.0%**	**100.0%**	**100.0%**
TIDYUP(10)	5	±0	**+5**	**+5**	2307584	99.94%	0.12%	**0.1%**
TTIREWORLD(40)	**3**	**±0**	**±0**	**±0**	**24805**	**100.0%**	**100.0%**	**100.0%**
BW(30)	**25**	–1	–1	–1	**191627**	**100.0%**	**100.0%**	**100.0%**
FAULTS(55)	**55**	**±0**	**±0**	**±0**	**160599**	**100.0%**	**100.0%**	**100.0%**
Overall	301	+3	**+8**	**+8**	66364773	35.13%	31.66%	**31.65%**

Table 1: Comparison of plain FIP with FIP using DSSS and different envelopes: full, active (DACT), nondeterministic active (NACT). Nodes of DACT, NACT in % of plain FIP.

	Coverage				Node Generations			
Domain	LAO*	NSSS	DACT	NACT	LAO*	NSSS	DACT	NACT
FR(75)	57	+1	+1	**+2**	323275	85.86%	85.86%	**81.51%**
FR-NEW(91)	**19**	–2	–2	–2	**455521**	100.7%	100.7%	100.77%
FOREST-NEW(90)	3	**+3**	+2	**+3**	10989	**93.28%**	**93.28%**	**93.28%**
FOREST(90)	6	**+1**	**+1**	**+1**	**17861**	100.96%	100.96%	100.96%
EARTH(40)	**30**	**±0**	**±0**	**±0**	**80229**	**100.0%**	**100.0%**	**100.0%**
TIDYUP(10)	**9**	**±0**	**±0**	**±0**	40166	99.95%	72.47%	**59.68%**
TTIREWORLD(40)	**6**	**±0**	**±0**	**±0**	**27807**	**100.0%**	**100.0%**	**100.0%**
BW(30)	**21**	**±0**	**±0**	**±0**	**111845**	**100.0%**	**100.0%**	**100.0%**
FAULTS(55)	**54**	**±0**	**±0**	**±0**	**153582**	**100.0%**	**100.0%**	**100.0%**
Overall	205	+3	+2	**+4**	1221275	96.47%	95.57%	**94.02%**

Table 2: Comparison of plain LAO* with LAO* using NSSS and different envelopes: full, active (DACT), nondeterministic active (NACT). Nodes of DACT, NACT in % of plain LAO*.

a time limit of 30 minutes and a memory limit of 4GB for the Java Runtime Environment. We considered an instance as solved if the planner finds a solution within its time limit or proves that none exists.

Results

Our experiments yield the following insights:

- *DSSS approach*: FIP combined with DSSS significantly increases coverage and reduces node expansions. This is particularly pronounced in the domains *Forest* and *Forest-new*. DSSS with active envelope solves five additional problem instances of the *Tidyup* domain which has—in contrast to all other domains—applicable inactive operators permitting immense pruning (0.1% nodes generated). In all other domains the effect of active envelope and also of nondeterministic active envelope is negligible. In *First-Responders* and *First-Responders-new* the DSSS approach loses one instance because of heuristic tie-breaking. Also, it loses two instances in *Earth-Observation* and one in *Blocksworld*. The instances in *Earth-Observation* result from the vast number of generated states. In *Blocksworld* the DSSS approach fails to solve the hardest problem solved by the baseline which solves it close to the time limit (1564s out of 1800s).
- *NSSS approach*: LAO* search combined with NSSS does also clearly outperform its baseline in terms of coverage and node expansions. But in contrast to the DSSS approach, node generations are not as drastically reduced. Since the LAO* algorithm produces noticeably more overhead per node than the FIP algorithm, reducing a single node has greater impact for the LAO* algorithm than for FIP. The coverage increase is most evident in the domain *Forest-new* where three additional instances are solved and *First-Responders-new* with two additionally solved instances. The loss of two instances in *First-Responders-new* is caused by heuristic tie-breaks (100.7% node expansions). A blind search without stubborn set does not solve any problem in this domain, whereas in combination with NSSS, it solves the smallest instance. Furthermore, active envelope and nondeterministic active envelope are beneficial in terms of pruning power. We again see the empirical dominance of nondeterministic active envelope over active envelope supporting the theoretical results.
- *Prefix Compatibility*: In terms of pruning power, the approach without prefix compatibility for the nontrivial operators is dominated by the syntactic approximation of prefix compatibility. Except in the domain *First-Responders-new* more nodes are generated because of a single instance which is caused by heuristic tie-breaking (100.45% node expansions). The increased pruning power is reflected in better coverage for *First-Responders-new* and *Forest-new* where two hard additional instances are solved respectively.

	Coverage		Node Generations	
Domain	no prefix	syntactic	no prefix	syntactic
FR(75)	**59**	–2	264437	**96.27%**
FR-NEW(91)	17	**+1**	**535598**	100.45%
FOREST-NEW(90)	6	**+1**	21088	**56.02%**
FOREST(90)	**7**	–1	18033	**72.36%**

Table 3: Comparison of no prefix compatibility for nontrivial operators with syntactic approximation. We grouped the domains where coverage and node generations are equal. Nodes of syntactic in % of no prefix approach.

Conclusion

We demonstrated that the stubborn set approach is also beneficial for FOND planning. While this was expectable for FIP and deterministic strong stubborn sets, we needed a new formalism for LAO* search, which does not reduce FOND to classical planning. We provided a novel notion of stubborn sets and proved that it is completeness preserving.

For the future, we want to focus on how prefix compatibility can be better approximated and evaluate how the degree of a nondeterministic operator affects the results of our approach. As a first step, we noticed that two operators cannot be prefix compatible if they have mutex preconditions. Also, it would be interesting to combine PRP with stubborn sets.

Acknowledgments

This work was partly supported by the DFG as part of the SFB/TR 14 AVACS and by the Swiss National Science Foundation (SNSF) as part of the project "Automated Reformulation and Pruning in Factored State Spaces (ARAP)".

References

Aldinger, J., and Löhr, J. 2013. Planning for agile earth observation satellites. In *Proceedings of the Workshop on Planning in Continuous Domains (PCD) at ICAPS 2013*.

Alkhazraji, Y.; Wehrle, M.; Mattmüller, R.; and Helmert, M. 2012. A stubborn set algorithm for optimal planning. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, 891–892.

Bäckström, C., and Nebel, B. 1993. Complexity results for sas+ planning. *Computational Intelligence* 11:625–655.

Chen, Y., and Yao, G. 2009. Completeness and optimality preserving reduction for planning. In *Proceedings of the 21st International Joint Conference on Artifical Intelligence (IJCAI 2009)*, 1659–1664.

Godefroid, P. 1995. Partial-order methods for the verification of concurrent systems: An approach to the state-explosion problem.

Hansen, E. A., and Zilberstein, S. 2001. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence* 129(1–2).

Hertle, A.; Dornhege, C.; Keller, T.; Mattmüller, R.; Ortlieb, M.; and Nebel, B. 2014. An experimental comparison of classical, FOND and probabilistic planning. In *Proceedings of the 37th Annual German Conference on AI (KI 2014)*, 297–308.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research (JAIR)* 14:2001.

Kuter, U.; Nau, D.; Reisner, E.; and Goldman, R. P. 2008. Using classical planners to solve nondeterministic planning problems. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS 2008)*, 190–197.

Laarman, A.; Pater, E.; van de Pol, J.; and Weber, M. 2013. Guard-based partial-order reduction. In *Proceedings of the 21st International Symposium on Model Checking of Software (SPIN 2014)*, 227–245.

Mattmüller, R.; Ortlieb, M.; Helmert, M.; and Bercher, P. 2010. Pattern database heuristics for fully observable nondeterministic planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS 2010)*, 105–112.

Muise, C.; McIlraith, S. A.; and Beck, J. C. 2012. Improved non-deterministic planning by exploiting state relevance. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 172–180.

Valmari, A. 1989. Stubborn sets for reduced state space generation. In *Proceedings of the Tenth International Conference on Application and Theory of Petri Nets (APN 1989)*, 1–22.

Wehrle, M., and Helmert. 2014. Efficient stubborn sets: Generalized algorithms and selection strategies. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 323–331.

Wehrle, M.; Helmert, M.; Alkhazraji, Y.; and Mattmüller, R. 2013. The relative pruning power of strong stubborn sets and expansion core. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS 2013)*, 251–259.

Counterexample-Guided Abstraction Refinement for POND Planning

Jonas Thiem and **Robert Mattmüller** and **Manuela Ortlieb**
University of Freiburg, Germany
{thiemj,mattmuel,ortlieb}@informatik.uni-freiburg.de

Abstract

Counterexample-guided abstraction refinement (CEGAR) allows to gradually refine a problem until the required detail for a solution is reached. We propose the use of CEGAR to demonstrate unsolvability of partially observable nondeterministic planning tasks while avoiding search through the entire state space.

Partially observable tasks are ubiquitous in planning and robotics (Oliehoek 2010, p. 3). Sometimes, it is important to show unsolvability of such a task fast. Examples include algorithms minimizing necessary sensors, where unsolvability proofs are needed to show that a certain sensor cannot be left out (Mattmüller, Ortlieb, and Wacker 2014), and algorithms for strong and strong cyclic planning (Cimatti et al. 2003) that first try to find a strong plan, and if strong plan non-existence has been established, resort to finding a strong cyclic plan instead. The Counterexample-Guided Abstraction Refinement (CEGAR) technique originating from model checking (Clarke et al. 2003) can be used to speed up unsolvability proofs and has recently been used for classical planning (Seipp 2012), and, in a setting closely related to ours, in the context of games with incomplete information (Dimitrova and Finkbeiner 2008). CEGAR works as follows: It starts with a small initial abstraction of the planning task and searches for an abstract plan. If no such plan exists, it makes use of the fact that abstractions induce over-approximations of reachability and concludes that no concrete plan can exist, either. Otherwise, CEGAR tries to concretize the abstract plan found. Either, the solution is concretizable. Then CEGAR terminates. Otherwise, the solution is *spurious* and the abstraction must be refined. In our setting, instead of requiring abstractions where every single transition is preserved, preserving goal reachability is sufficient. A central question is how to define abstractions (guaranteeing over-approximations). A straightforward way for POND planning is to define an abstract belief state $\mathcal{B}$ as a set of concrete belief states B, where each such B consists of the set of world states s considered possible in B. Then the abstract initial state, goal states, and transitions can be defined easily. E.g., an action precondition φ is satisfied in $\mathcal{B}$ iff it is satisfied in *some* concrete belief state B represented by $\mathcal{B}$ (to ensure over-approximation), and φ is satisfied in B iff it is satisfied in *all* states $s \in B$ (to account for the uncertainty of the belief B). Unfortunately, representing such a set of sets $\mathcal{B}$ compactly is hard. On the other hand, representing a set B of states s compactly is, although exponential in the worst case, often feasible using binary decision diagrams (BDDs) (Bryant 1986). Therefore, in this work we approximate abstract belief states $\mathcal{B}$ by BDD-encoded sets of world states B. Furthermore, as abstractions we use simple projections to *patterns* P, i.e., sets of variables (Culberson and Schaeffer 1996). This raises several questions: (a) How to define and compute an (approximate) abstraction to a pattern P efficiently, (b) how to ensure that goal reachability is preserved, and (c) how to refine an abstraction if necessary. For (a), we use a simple *syntactic* projection to P similar to the one used for PDB heuristics in classical planning. However, when we use sets B of world states as abstract states and thus let the layers "belief" and "abstraction" collapse into one, we introduce an error that violates the over-approximation. We amend this as follows: We only allow variables in P that can always and unconditionally be observed, or that are known initially and can never become unknown. In addition, we forbid observations of variables outside of the pattern. This guarantees that we only ever produce singleton abstract belief states. Since we forbid some observations, we have no longer an over-approximation, but it can be proven that the goal reachability, possibly along longer paths, is preserved with the chosen restrictions. Regarding (c), we refine an abstraction by collecting all actions in the abstract policy whose precondition is violated in the concrete task and add the violated precondition variables to the refined pattern. If this violates the restriction of patterns, we immediately move to a pattern consisting of all variables in the planning task, i.e., to the identity abstraction. We implemented this variant of the CEGAR algorithm on top of the MYND planner (Mattmüller et al. 2010). For the benchmarked unsolvable problems, CEGAR leads to an increase of 20 to 50 percent in successfully handled unsolvable problems. As a downside, solvable problems suffer a slowdown which gradually widened in our benchmarks. For future work, we plan to investigate the performance of CEGAR as part of the two motivating scenarios, sensor minimization and strong/strong-cyclic planning. We also plan to investigate alternative abstractions such as the doubly exponential one mentioned above.

Acknowledgments

This work was partly supported by the DFG as part of the SFB/TR 14 AVACS.

References

Bryant, R. E. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers* 35(8):677–691.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence* 147(1–2):35–84.

Clarke, E.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2003. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM (JACM)* 50(5):752–794.

Culberson, J. C., and Schaeffer, J. 1996. Searching with pattern databases. In *Advances in Artificial Intelligence*, 402–416. Springer-Verlag.

Dimitrova, R., and Finkbeiner, B. 2008. Abstraction refinement for games with incomplete information. In *FSTTCS*, 175–186. Citeseer.

Mattmüller, R.; Ortlieb, M.; Helmert, M.; and Bercher, P. 2010. Pattern database heuristics for fully observable nondeterministic planning. In Brafman, R.; Geffner, H.; Hoffmann, J.; and Kautz, H., eds., *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS 2010)*, 105–112.

Mattmüller, R.; Ortlieb, M.; and Wacker, E. 2014. Minimizing necessary observations for nondeterministic planning. In *KI 2014: Advances in Artificial Intelligence*. Springer. 309–320.

Oliehoek, F. 2010. *Value-based planning for teams of agents in stochastic partially observable environments*. Amsterdam University Press.

Seipp, J. 2012. *Counterexample-guided abstraction refinement for classical planning*. Master's thesis, Albert-Ludwigs-Universität Freiburg.

Compiling Away LTL Planning Goals in Polynomial Time

Jorge Torres
Department of Computer Science
Pontificia Universidad Católica de Chile
Santiago, Chile

Jorge A. Baier
Department of Computer Science
Pontificia Universidad Católica de Chile
Santiago, Chile

Abstract

Linear temporal logic (LTL) is an expressive language that allows specifying temporally extended goals and preferences. A general approach to dealing with general LTL properties in planning is by "compiling them away"; i.e., in a pre-processing phase, all LTL formulas are converted into simple, non-temporal formulas that can be evaluated in a planning state. This is accomplished by first generating a finite-state automaton for the formula, and then by introducing new fluents that are used to capture all possible runs of the automaton. Unfortunately, current translation approaches are worst-case exponential on the size of the LTL formula. In this paper, we present a polynomial approach to compiling away LTL goals. Our method relies on the exploitation of alternating automata. Since alternating automata are different from non-deterministic automata, our translation technique does not capture all possible runs in a planning state and thus is very different from previous approaches. We prove that our translation is sound and complete, and evaluate it empirically showing that it has strengths and weaknesses. Specifically, we find classes of formulas in which it seems to outperform significantly the current state of the art.

Introduction

Linear Temporal Logic (LTL) (Pnueli 1977) is a compelling language for the specification of goals in AI planning, because it allows defining constraints on state trajectories which are more expressive than simple final-state goals, such as "*deliver priority packages before non-priority ones*", or "*while moving from the office to the kitchen, make sure door D becomes closed some time after it is opened*". It was first proposed as the goal specification language of TLPlan system (Bacchus and Kabanza 1998). Currently, a limited but compelling subset of LTL has been incorporated into PDDL3 (Gerevini et al. 2009) for specifying hard and soft goals.

While there are some systems that natively support the PDDL3 subset of LTL [e.g., Coles and Coles, 2011], when planning for general LTL goals, there are two salient approaches: goal progression (Bacchus and Kabanza 1998) and compilation approaches (Rintanen 2000; Cresswell and Coddington 2004; Edelkamp, Jabbar, and Naizih 2006; Baier and McIlraith 2006). Goal progression has been shown to be extremely effective when the goal formula encodes some domain-specific control knowledge that prunes large portions of the search space (Bacchus and Kabanza 2000). In the absence of such expert knowledge, however, compilation approaches are more effective at planning for LTL goals since they produce an equivalent classical planning problem, which can then be fed into optimized off-the-shelf planners.

State-of-the-art compilation approaches to planning for LTL goals exploit the relationship between LTL and finite-state automata (FSA) (Edelkamp 2006; Baier and McIlraith 2006). As a result, the size of the output is worst-case *exponential* in the size of the LTL goal. Since deciding plan existence for both LTL and classical goals is PSPACE-complete (Bylander 1994; De Giacomo and Vardi 1999), none of these approaches is optimal with respect to computational complexity, since they rely on a potentially exponential compilation. From a practical perspective, this worst case is also problematic since the size of a planning instance has a direct influence on planning runtime.

In this paper, we present a novel approach to compile away general LTL goals into classical goals that runs in polynomial time on the size of the input that is thus optimal with respect to computational complexity. Like existing FSA approaches, our compilation exploits a relation between LTL and automata, but instead of FSA, we exploit alternating automata (AA), a generalization of FSA that does not seem to be efficiently compilable with techniques used in previous approaches. Specifically, our compilation handles each non deterministic choice of the AA with a specific action, hence leaving non-deterministic choices to be decided at planning time. This differs substantially from both Edelkamp's and Baier and McIlraith's approaches, which represent all runs of the automaton simultaneously in a single planning state.

We propose variants of our method that lead to performance improvements of planning systems utilizing relaxed-plan heuristics. Finally, we evaluate our compilation empirically, comparing it against Baier and McIlraith's—who below we refer to as B&M. We conclude that our translation has strengths and weaknesses: it outperforms B&M's for classes of formulas that require very large FSA, while B&M's seems stronger for shallower, simpler formulas.

In the rest of the paper, we outline the required background, we describe our AA construction for finite LTL logic, and then show the details of our compilation approach. We continue describing the details of our empirical evaluation. We finish with conclusions. Refer to Torres and Baier (2015) for a slightly extended version of this paper.

Preliminaries

The following sections describe the background necessary for the rest of the paper.

Propositional Logic Preliminaries

Given a set of propositions F, the set of *literals* of F, $Lit(F)$, is defined as $Lit(F) = F \cup \{\neg p \mid p \in F\}$. The complement of a literal ℓ is denoted by $\overline{\ell}$, and is defined as $\neg p$ if $\ell = p$ and as p if $\ell = \neg p$, for some $p \in F$. $\overline{L}$ denotes $\{\overline{\ell} \mid \ell \in L\}$.

Given a Boolean value function $\pi : P \to \{\mathsf{false}, \mathsf{true}\}$, and a Boolean formula φ over P, $\pi \models \varphi$ denotes that π satisfies φ, and we assume it defined in the standard way. To simplify notation, we use $s \models \varphi$, for a set s of propositions, to abbreviate $\pi_s \models \varphi$, where $\pi_s = \{p \to \mathsf{true} \mid p \in s\} \cup \{p \to \mathsf{false} \mid p \in F \setminus s\}$. In addition, we say that $s \models R$, when R is a set of Boolean formulas, iff $s \models r$, for every $r \in R$.

Deterministic Classical Planning

Deterministic classical planning attempts to model decision making of an agent in a deterministic world. We use a standard planning language that allows so-called negative preconditions and conditional effects. A *planning problem* is a tuple $\langle F, O, I, G\rangle$, where F is a set of propositions, O is a set of action operators, $I \subseteq F$ defines an initial state, and $G \subseteq Lit(F)$ defines a goal condition.

Each action operator a is associated with the pair $(prec(a), \mathit{eff}(a))$, where $prec(a) \subseteq Lit(F)$ is the *precondition* of a and $\mathit{eff}(a)$ is a set of *conditional effects*, each of the form $C \to \ell$, where $C \subseteq Lit(F)$ is a *condition* and literal ℓ is the effect. Sometimes we write ℓ as a shorthand for the unconditional effect $\{\} \to \ell$.

We say that an action a is *applicable* in a planning state s iff $s \models prec(a)$. We denote by $\rho(s, a)$ the state that results from applying a in s. Formally,

$$\begin{aligned}\rho(s,a) =& (s \setminus \{p \mid C \to \neg p \in \mathit{eff}(a), s \models C\}) \cup \\ & \{p \mid C \to p \in \mathit{eff}(a), s \models C\}\end{aligned}$$

if $s \in F$ and a is applicable in s; otherwise, $\delta(a, s)$ is undefined. If α is a sequence of actions and a is an action, we define $\rho(s, \alpha a)$ as $\rho(\delta(s, \alpha), a)$ if $\rho(s, \alpha)$ is defined. Furthermore, if α is the empty sequence, then $\rho(s, \alpha) = s$.

An action sequence α is *applicable* in a state s iff $\rho(s, \alpha)$ is defined. If an action sequence $\alpha = a_1 a_2 \ldots a_n$ is applicable in s, it induces an execution trace $\sigma = s_1 \ldots s_{n+1}$ in s, where $s_i = \rho(I, a_1 \ldots a_{i-1})$, for every $i \in \{1, \ldots, n+1\}$.

An action sequence is a *plan* for problem $\langle F, O, I, G\rangle$ if α is applicable in I and $\rho(I, \alpha) \models G$.

Alternating Automata

Alternating automata (AA) are a natural generalization of non-deterministic finite-state automata (NFA). At a definitional level, the difference between an NFA and an AA is the transition function. For example, if A is an NFA with transition function δ, and we have that $\delta(q, a) = \{p, r\}$, then this intuitively means that A may end up in state p or in state r as a result of reading symbol a when A was previously in state q. With an AA, transitions are defined as formulas. For example, if δ' is the transition function for an AA A', then $\delta'(q, a) = p \vee r$ means, as before, that A' ends up in p or r after reading an a in state q. Nevertheless, formulas provide more expressive power. For example $\delta'(q, b) = (s \wedge t) \vee r$ can be intuitively understood as A' will end up in both s and t or (only) in r after reading a b in state q. In this model, only *positive Boolean formulas* are allowed for defining δ.

Definition 1 (Positive Boolean Formula) *The set of positive formulas over a set of propositions $\mathcal{P}$—denoted by $\mathcal{B}^+(\mathcal{P})$—is the set of all Boolean formulas over $\mathcal{P}$ and constants $\bot$ and $\top$ that do not use the connective "$\neg$".*

The formal definition for AA that we use henceforth follows.

Definition 2 (Alternating Automata) *An alternating automata (AA) over words is a tuple $A = (Q, \Sigma, \delta, \mathcal{I}, \mathcal{F})$, where Q is a finite set of states, Σ, the alphabet, is a finite set of symbols, $\delta : Q \times \Sigma \to \mathcal{B}^+(Q)$ is the transition function, $\mathcal{I} \subseteq Q$ are the initial states, and $\mathcal{F} \subseteq Q$ is a set of final states.*

As suggested above, any NFA is also an AA. Indeed, given an NFA with transition function δ, we can generate an equivalent AA with transition function δ' by simply defining $\delta'(q, a) = \bigvee_{p \in P} p$, when $\delta(q, a) = P$. We observe that this means $\delta'(q, a) = \bot$ when P is empty.

As with NFAs, an AA accepts a word w whenever there exists a *run* of the AA over w that satisfies a certain property. Here is the most important (computational) difference between AAs and NFAs: a run of an AA is a sequence of *sets* of states rather than a sequence of states. Before defining runs formally, for notational convenience, we extend δ for any subset T of Q as $\delta(T, a) = \bigwedge_{q \in T} \delta(q, a)$ if $T \neq \emptyset$ and $\delta(T, a) = \top$ if $T = \emptyset$.

Definition 3 (Run of an AA over a Finite String) *A run of an AA $A = (Q, \Sigma, \delta, \mathcal{I}, \mathcal{F})$ over word $x_1 x_2 \ldots x_n$ is a sequence $Q_0 Q_1 \ldots Q_n$ of subsets of Q, where $Q_0 = \mathcal{I}$, and $Q_i \models \delta(Q_{i-1}, x_i)$, for every $i \in \{1, \ldots, n\}$.*

Definition 4 *A word w is accepted by an AA A iff there is a run $Q_0 \ldots Q_n$ of A over w such that $Q_n \subseteq F$.*

For example, if the definition of an AA A is such that $\delta'(q, b) = (s \wedge t) \vee r$, and $\mathcal{I} = \{q\}$, then both $\{q\}\{s, t\}$ and $\{q\}\{r\}$ are runs of A over word b.

Finite LTL

The focus of this paper is planning with LTL interpreted over *finite* state sequences (Baier and McIlraith 2006; De Giacomo and Vardi 2013). At the syntax level, the finite LTL we use in this paper is almost identical to regular LTL, except for the addition of a "weak next" modality ($\bullet$). The definition follows.

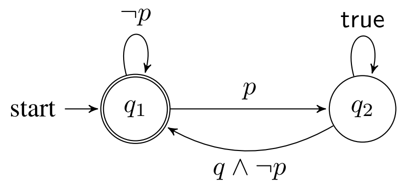


Figure 1: An NFA for formula $\square(p \rightarrow \bigcirc\Diamond q)$ that expresses the fact that every time p becomes true in a state, then q has to be true in the state after or in the future. The input to the automaton is a (finite) sequence $s_0 \ldots s_n$ of planning states.

Definition 5 (Finite LTL formulas) *The set of finite LTL formulas over a set of propositions $\mathcal{P}$, fLTL($\mathcal{P}$), is inductively defined as follows:*

- *p is in fLTL($\mathcal{P}$), for every $p \in \mathcal{P}$.*
- *If φ and ψ are in fLTL($\mathcal{P}$) then so are $\neg\varphi$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $\bigcirc\varphi$, $\bullet\varphi$, $(\varphi \,\mathsf{U}\, \psi)$, and $(\varphi \,\mathsf{R}\, \psi)$.*

The truth value of a finite LTL formula is evaluated over a finite sequence of states. Below we assume that those states are actually planning states.

Definition 6 *Given a sequence of states $\sigma = s_0 \ldots s_n$ and a formula $\varphi \in$ fLTL($\mathcal{P}$), we say that σ satisfies φ, denoted as $\sigma \models \varphi$, iff it holds that $\sigma, 0 \models \varphi$, where, for every $i \in \{0, \ldots, n\}$:*

1. *$\sigma, i \models p$ iff $s_i \models p$, when $p \in \mathcal{P}$.*
2. *$\sigma, i \models \neg\varphi$ iff $\sigma, i \not\models \varphi$*
3. *$\sigma, i \models \psi \wedge \chi$ iff $\sigma, i \models \psi$ and $\sigma, i \models \chi$*
4. *$\sigma, i \models \psi \vee \chi$ iff $\sigma, i \models \psi$ or $\sigma, i \models \chi$*
5. *$\sigma, i \models \bigcirc\psi$ iff $i < n$ and $\sigma, (i+1) \models \psi$*
6. *$\sigma, i \models \bullet\psi$ iff $i = n$ or $\sigma, (i+1) \models \psi$*
7. *$\sigma, i \models \psi \,\mathsf{U}\, \chi$ iff there exists $k \in \{i, ..., n\}$ such that $\sigma, k \models \chi$ and for each $j \in \{i, ..., k-1\}$, it holds that $\sigma, j \models \psi$*
8. *$\sigma, i \models \psi \,\mathsf{R}\, \chi$ iff for each $k \in \{i, ..., n\}$ it holds that $\sigma, k \models \chi$ or there exists a $j \in \{i, ..., k-1\}$ such that $\sigma, j \models \psi$*

We sometimes use the macros $\mathsf{true} \stackrel{\text{def}}{=} p \vee \neg p$, $\mathsf{false} \stackrel{\text{def}}{=} \neg\mathsf{true}$, and $\varphi \rightarrow \psi$ as $\neg\varphi \vee \psi$. Additionally, $\Diamond\varphi$, pronounced as "eventually φ" is defined as $\mathsf{true} \,\mathsf{U}\, \varphi$, and $\square\varphi$, pronounced as "always φ" is defined as $\neg\Diamond\neg\varphi$.

Deterministic Planning with LTL goals

A planning problem with a finite LTL goal is a tuple $P = \langle F, O, I, G \rangle$, where F, O, and I are defined as in classical planning problems, but where G is a formula in $fLTL(F)$. An action sequence α is a plan for P if α is applicable in I, and the execution trace σ induced by the execution of α in I is such that $\sigma \models G$.

There are two approaches to compiling away LTL via non-deterministic finite-state automata (Edelkamp, Jabbar, and Naizih 2006; Baier and McIlraith 2006). B&M's approach compiles away LTL formulas exploiting the fact that for every finite LTL formula φ it is possible to build an NFA that accepts the finite models of φ. To illustrate this, Figure 1 shows an NFA for $\square(p \rightarrow \bigcirc\Diamond q)$. B&M represent the NFA within the planning domain using one fluent per automaton state. In the example of Figure 1, this means that the new planning problem contains fluents E_{q_1} and E_{q_2}. The translation is such that if α is a sequence of actions that induces the execution trace $\sigma = s_1 \ldots s_n$, then E_q is true in s_n iff there is some run of the automaton over σ that ends in state q. B&M's translation has the following property.

Theorem 1 (Follows from (Baier 2010)) *Let P be a classical planning problem, φ be a finite LTL formula, and P' be the instance that results from applying the B&M translation to P. Moreover, let α be a sequence of actions applicable in the initial state of P, and let σ and σ' be, respectively, the sequences of (planning) states induced by the execution of α in P and P'. Finally, let A_φ be the NFA for φ. Then the following are equivalent statements.*

1. *There exists a run ρ of A_φ ending in q.*
2. *E_q is true in the last state of σ'.*

As a corollary of the previous theorem, one obtains that satisfaction of finite LTL formulas can be determined by checking whether or not the disjunction $\bigvee_{f \in \mathcal{F}} E_f$ holds, where $\mathcal{F}$ denotes the set of final states of A_φ.

Unfortunately, B&M's translation is worst-case exponential (Baier 2010); for example, an NFA for $\wedge_{i=1}^{n} \Diamond p_i$ has 2^n states. Baier (2010) proposes a formula-partitioning technique that allows the method to generate more compact translations for certain formulas. The method, however, is not applicable to any formula.

Edelkamp's approach is similar to B&M's: it builds a Büchi automaton (BA), whose states are represented via fluents, compactly representing all runs of the automaton in a single planning state. The main difference is that the state of the automaton is updated via specific actions—a process that they call *synchronized update*. We modify this idea in the compilation we give below; however, our compilation is significantly different since it does not represent all runs of the automaton in the same planning state. It is important to remark that the use of BA interpreted as NFA does not yield a correct translation for general LTL goals, although it is correct for the PDDL3 subset of LTL (De Giacomo, Masellis, and Montali 2014).

Alternating Automata and Finite LTL

A central part of our approach is the generation of an AA from an LTL formula. To do this we modify Muller, Saoudi, and Schupp's AA (1988) for *infinite* LTL formulas. Our AA is equivalent to a recent proposal by De Giacomo, Masellis, and Montali (2014). The main difference between our construction and De Giacomo, Masellis, and Montali's is that we do not assume a distinguished proposition becomes true only in the final state. On the other hand, we require a special state (q_F) that indicates the sequence should finish. The use of such a state is the main difference between our AA for finite LTL and Muller, Saoudi, and Schupp's AA for infinite LTL.

We require the LTL input formula to be written in negation normal form (NNF); i.e., a form in which negations can be applied only to atomic formula. This transformation can be done in linear time (Gerth et al. 1995).

Let φ be in *fLTL(F)* and $sub(\varphi)$ be the set of the subformulas of φ, including φ. We define $A_\varphi = (Q, 2^F, \delta, q_\varphi, \{q_F\})$, where $Q = \{q_\alpha \mid \alpha \in sub(\varphi)\} \cup \{q_F\}$ and:

$$
\begin{array}{rcl}
\delta(q_\ell, s) & = & \begin{cases} \top, & \text{if } \ell \in Lit(F) \text{ and } s \models \ell \\ \bot, & \text{if } \ell \in Lit(F) \text{ and } s \not\models \ell \end{cases} \\
\delta(q_F, s) & = & \bot \\
\delta(q_{\alpha\vee\beta}, s) & = & \delta(q_\alpha, s) \vee \delta(q_\beta, s) \\
\delta(q_{\alpha\wedge\beta}, s) & = & \delta(q_\alpha, s) \wedge \delta(q_\beta, s) \\
\delta(q_{\bigcirc\alpha}) & = & q_\alpha \\
\delta(q_{\bullet\alpha}) & = & q_F \vee q_\alpha \\
\delta(q_{\alpha \mathsf{U} \beta}, s) & = & \delta(q_\beta, s) \vee (\delta(q_\alpha, s) \wedge q_{\alpha \mathsf{U} \beta}) \\
\delta(q_{\alpha \mathsf{R} \beta}, s) & = & \delta(q_\beta, s) \wedge (q_F \vee \delta(q_\alpha, s) \vee q_{\alpha \mathsf{R} \beta})
\end{array}
$$

Theorem 2 *Given an LTL formula φ and a finite sequence of states σ, A_φ accepts σ iff $\sigma \models \varphi$.*

Proof sketch: Suppose that $\sigma = x_1 x_2 \dots x_n \in \sigma^*$. Then it can be proven by induction on the construction of φ the following lemma: $\sigma, i \models \varphi$ if and only if exists a sequence $r = Q_{i-1} Q_i \dots Q_n$, such that: $Q_{i-1} = \{\varphi\}$, $Q_n \subseteq \{q_F\}$, for each subset Q_j in the sequence r it holds that $Q_j \subseteq sub(\varphi) \cup \{q_F\}$ and for each $j \in \{i, i+1, \dots, n\}$ it holds that $Q_j \models \delta(Q_{j-1}, x_j)$. ■

Compiling Away finite LTL Properties

Now we propose an approach to compiling away finite LTL properties using the AA construction described above.

First, we argue that the idea underlying both Edelkamp's and B&M's translations would *not* yield an efficient translation if applied to AA. Recall in both approaches if $E_{q_1}, \dots, E_{q_n}$ are true in a planning state s, then there are n runs of the automaton, each of which ends in $q_1, \dots, q_n$ (Theorem 1). In other words, the planning state keeps track of *all* of the runs of the automaton. To apply the same principle to AA, we would need to introduce one fluent for each *subset* of states of the AA, therefore generating a number of fluents exponential on the size of the original formula. This is because runs of AA are sequences of *sets* of states, so we would require states of the form E_R, where R is a set of states.

To produce an efficient translation, we renounce the idea of representing all runs of the automaton in a single planning state. Our translation will then only keep track of a *single* run.

Translating LTL via LTL Synchronization

Our compilation approach takes as input an LTL planning problem P and produces a new planning problem P', which is is like P but contains additional fluents and actions. Like previous compilations, A_G is represented in P' with additional fluents, one for each state of the automaton for G. Like in Edelkamp's compilation P' contains specific actions—below referred to as *synchronization actions*—whose only purpose is to update the truth values of those additional fluents. A plan for P' alternates one action from the original problem P with a number of synchronization actions. Unlike any other previous compilation, P' does not represent all possible runs of the automaton in a single planning state.

Synchronization actions update the state of the automaton following the definition of the δ function. The most notable characteristic that distinguishes synchronization from the Edelkamp' s translation is that non-determinism inherent to the AA is modeled using alternative actions, each of which represents the different non-deterministic options of the AA. As such if there are n possible non-deterministic choices, via the applications of synchronization actions there will be n reachable planning states, each representing a single run.

Given a planning problem $P = \langle F, O, I, G \rangle$, our translation generates a problem P' in which there is one (new) fluent q for each state q of the AA A_G. The compilation is such that the following property holds: if $\alpha = a_1 a_2 \dots a_n$ is applicable in the initial state of P, then there exists a set $\mathcal{A}_\alpha$ of action sequences of the form $\alpha_0 a_1 \alpha_1 a_2 \alpha_2 \dots a_n \alpha_n$, where each α_i is a sequence of synchronization actions whose sole objective is to update the fluents representing A_G's state.

Our theoretical result below says that our compilation can represent all runs, but only one run at a time. Specifically, each of the sequences of $\mathcal{A}_\alpha$ corresponds to some run of A_G over the state sequence induced by α over P. Moreover, if $\alpha' \in \mathcal{A}_\alpha$, E_q is true in the state resulting from performing sequence α' in P' iff q is *contained* in the last element of a run that corresponds to α'.

We we are ready to define P'. Assume the AA for G has the form $A_G = (Q, \Sigma, \delta, q_0, \{q_f\})$.

Fluents P' has the same fluents as P plus fluents for the representation of the states of the automaton (Q), flags for controlling the different modes ($\mathbf{copy}$, $\mathbf{sync}$, $\mathbf{world}$), and a special fluent $\mathbf{ok}$, which becomes false if the goal has been falsified. Finally, it includes the set $Q^S = \{q^S \mid q \in Q\}$ which are "copies" of the automata fluents, which we describe in detail below. Formally, $F' = F \cup Q \cup Q^S \cup \{\mathbf{copy}, \mathbf{sync}, \mathbf{world}, \mathbf{ok}\}$.

The set of operators O' is the union of the sets O_w and O_s.

World Mode Set O_w contains the same actions in O, but preconditions are modified to allow execution only in "world mode". Effects, on the other hand are modified to allow the execution of the *copy* action, which initiates the synchronization phase, and which is described below. Formally, $O_w = \{a' \mid a \in O\}$, and for all a' in O_w:

$$
\begin{aligned}
prec(a') &= prec(a) \cup \{\mathbf{ok}, \mathbf{world}\}, \\
\mathit{eff}(a') &= \mathit{eff}(a) \cup \{\mathbf{copy}, \neg\mathbf{world}\}.
\end{aligned}
$$

Synchronization Mode The synchronization mode can be divided in three consecutive phases. In the first phase, we execute the *copy* action which in the successor states adds a copy q^S for each fluent q that is currently true, deleting q. Intuitively, during synchronization, each q^S defines the state of the automaton prior to synchronization. The precondition

Sync Action	Precondition	Effect
$trans(q^S_\ell)$	$\{\mathbf{sync}, \mathbf{ok}, q^S_\ell, \ell\}$	$\{\neg q^S_\ell\}$
$trans(q^S_F)$	$\{\mathbf{sync}, \mathbf{ok}, q^S_F\}$	$\{\neg q^S_F, \neg\mathbf{ok}\}$
$trans(q^S_{\alpha\wedge\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\alpha\wedge\beta}\}$	$\{q^S_\alpha, q^S_\beta, \neg q^S_{\alpha\wedge\beta}\}$
$trans_1(q^S_{\alpha\vee\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\alpha\vee\beta}\}$	$\{q^S_\alpha, \neg q^S_{\alpha\vee\beta}\}$
$trans_2(q^S_{\alpha\vee\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\alpha\vee\beta}\}$	$\{q^S_\beta, \neg q^S_{\alpha\vee\beta}\}$
$trans(q^S_{\bigcirc\alpha})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\bigcirc\alpha}\}$	$\{q_\alpha, \neg q^S_{\bigcirc\alpha}\}$
$trans_1(q^S_{\bullet\alpha})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\bullet\alpha}\}$	$\{q_F, \neg q^S_{\bullet\alpha}\}$
$trans_2(q^S_{\bullet\alpha})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\bullet\alpha}\}$	$\{q_\alpha, \neg q^S_{\bullet\alpha}\}$
$trans_1(q^S_{\alpha\,\mathsf{U}\,\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\alpha\,\mathsf{U}\,\beta}\}$	$\{q^S_\beta, \neg q^S_{\alpha\,\mathsf{U}\,\beta}\}$
$trans_2(q^S_{\alpha\,\mathsf{U}\,\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\alpha\,\mathsf{U}\,\beta}\}$	$\{q^S_\alpha, q_{\alpha\,\mathsf{U}\,\beta}, \neg q^S_{\alpha\,\mathsf{U}\,\beta}\}$
$trans_1(q^S_{\alpha\,\mathsf{R}\,\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\alpha\,\mathsf{R}\,\beta}\}$	$\{q^S_\beta, q_F, \neg q^S_{\alpha\,\mathsf{R}\,\beta}\}$
$trans_2(q^S_{\alpha\,\mathsf{R}\,\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\alpha\,\mathsf{R}\,\beta}\}$	$\{q^S_\beta, q^S_\alpha, \neg q^S_{\alpha\,\mathsf{R}\,\beta}\}$
$trans_3(q^S_{\alpha\,\mathsf{R}\,\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\alpha\,\mathsf{R}\,\beta}\}$	$\{q^S_\beta, q_{\alpha\,\mathsf{R}\,\beta}, \neg q^S_{\alpha\,\mathsf{R}\,\beta}\}$

Table 1: The synchronization actions for LTL goal G in NNF. Above ℓ, $\alpha\,\mathsf{R}\,\beta$, $\alpha\,\mathsf{U}\,\beta$, and $\bigcirc\alpha$ are assumed to be in the set of subformulas of G. In addition, ℓ is assumed to be a literal.

of *copy* is simply $\{\mathbf{copy}, \mathbf{ok}\}$, while its effect is defined by:

$$\mathit{eff}(\mathit{copy}) = \{q \to q^S, q \to \neg q \mid q \in Q\} \cup \{\mathbf{sync}, \neg\mathbf{copy}\}$$

As soon as the **sync** fluent becomes true, the second phase of synchronization begins. Here the only executable actions are those that update the state of the automaton, which are defined in Table 1. Note that one of the actions deletes the **ok** fluent. This can happen, for example while synchronizing a formula that actually expresses the fact that the action sequence has to conclude now.

When no more synchronization actions are possible, we enter the third phase of synchronization. Here only action $world$ is executable; its only objective is to reestablish world mode. The precondition of $world$ is $\{\mathbf{sync}, \mathbf{ok}\} \cup \overline{Q^S}$, and its effect is $\{\mathbf{world}, \neg\mathbf{sync}\}$.

The set O_s is defined as the one containing actions *copy*, $world$, and all actions defined in Table 1.

New Initial State The initial state of the original problem P intuitively needs to be "processed" by $\mathcal{A}_G$ before starting to plan. Therefore, we define I' as $I \cup \{q_G, \mathbf{copy}, \mathbf{ok}\}$.

New Goal Finally, the goal of the problem is to reach a state in which no state fluent in Q is true, except for q_f, which may be true. Therefore, we define $G' = \{\mathbf{world}, \mathbf{ok}\} \cup \overline{(Q \setminus \{q_F\})}$.

Properties

There are two important properties that can be proven about our translation. First, our translation is correct.

Theorem 3 (Correctness) *Let $P = \langle F, O, I, G\rangle$ be a planning problem with an LTL goal and $P' = \langle F', O', I', G'\rangle$ be the translated instance. Then P has a plan $a_1 a_2 \ldots a_n$ iff P' has a plan $\alpha_0 a_1 \alpha_1 a_2 \alpha_2 \ldots a_n \alpha_n$, in which for each $i \in \{0, \ldots, n\}$, α_i is a sequence of actions in O_s.*

Proof sketch: We show each sequence of actions α_i simulates the behavior of the automata, i.e., whenever t is a planning state whose next action must be *copy* and $q_\beta \in t$, then $\rho(t, \alpha_i)$ satisfies $\delta(q_\beta, t)$.
For this, let's define t^S as the subset of all the automata fluents Q^S that are added during the execution of the sequence of actions α_i. We will prove the following lemma by induction on the construction of φ: If $q^S_\varphi \in t^S$, then $\rho(t, \alpha_i) \models \delta(q_\varphi, t)$:
Observe that if $q^S_\varphi \in t^S$, then there must be an action *trans*(q^S_φ) that was executed in α_i. This is because $\rho(t, \alpha_i) \cap Q^S = \emptyset$ and only *trans*(q^S_φ) can delete q^S_φ from the current state. The second observation is: If some action *trans* adds q^S_α, then $q^S_\alpha \in t^S$. This is by definition of t^S. If the action adds q_ψ, then $q_\psi \in \rho(t, \alpha_i)$, because the only action that deletes fluents in Q is *copy*.

- $\varphi = \ell$. Assume ℓ is positive literal. Then there is a planning state s in which $trans(q^S_\ell)$ was executed. Since the precondition requires $\ell \in s$ and ℓ can only be added by an action from O_w, then $\ell \in t$. By definition, $\delta(q_\ell, t) = \top$, and it is clear that $\rho(t, \alpha_i) \models \delta(q_\varphi, t)$. The argument is analogous for a negative literal ℓ.

We will not consider the case for q_F. It is never desirable to synchronize that state, because the special fluent **ok** is removed, leading to a dead end. Now, assume that $q^S_\varphi \in t^S$ implies $\rho(t, \alpha_i) \models \delta(q_\varphi, t)$ for every φ with less than m operators. The proof sketch for each case can be verified by the reader as follows:

- For each φ, it is clear that a version of $trans(q^S_\varphi)$ is executed due to the first observation.
- If q_ψ is added by *trans*, then $q_\psi \in \rho(t, \alpha_i)$ due to the second observation. This implies that $\rho(t, \alpha_i) \models q_\psi$.
- If q^S_α is added by *trans*, then $q^S_\alpha \in t^S$. By induction hypothesis, $\rho(t, \alpha_i) \models \delta(q_\alpha, t)$, because α is a strict subformula of φ and has less than m operators.
- Finally, using entailment (for positive boolean formulae) and the definition of the transitions for the alternating automata A_φ, it can be verified that $\rho(t, \alpha_i) \models \delta(q_\varphi, t)$.
- The argument is similar for the other versions of *trans*.

To conclude our theorem, note that if t is a planning state, $q_\beta \in t$ and the next action to execute is *copy*, then $q^S_\beta \in t^S$. Using the lemma, this implies $\rho(t, \alpha_i) \models \delta(q_\varphi, t)$. ■

Second, the size of the plan for P' is linear on the size of the plan for P.

Theorem 4 (Bounded synchronizations) *If T is a reachable planning state from I' and $T \cap Q^S \neq \emptyset$, then there is a sequence of trans actions σ such that $\delta(T, copy\cdot\sigma) \cap Q^S = \emptyset$ and $|\sigma| \in \mathcal{O}(|G|)$.*

Proof: Note that T is a state in *world* mode getting ready to go into *synchronization* mode after the *copy* action has been executed. The main idea of the proof is to choose the order of the subformulae to be synchronized, where the first one corresponds to the largest subformula of the current state, the second one corresponds to the second largest subformula and so on. Note that when an action $trans(q^S_\alpha)$ is executed, it always happens that at most two fluents q^S_β and q^S_γ are added, and the formulae β and γ are strict subformulae of α. This means that a subformula will never get synchronized twice in a single synchronization phase σ. Since the number of subformulae is linear on $|G|$, this means that the length of σ must be $O(|G|)$. ■

Towards More Efficient Translations

The translation we have presented above can be modified slightly for obtaining improved performance. The following are modifications that we have considered.

An Order for Synchronization Actions Consider the goal formula is $\alpha \wedge \beta$ and that currently both q_α and q_β are true. The planner has two equivalent ways of completing the synchronization: by executing first $trans(q_\alpha)$ and then $trans(q_\beta)$, or by inverting this sequence. By enforcing an order between these synchronizations, we can reduce the branching factor at synchronization phase. Such an order is simple to enforce by modifying preconditions and effects of synchronization actions so that states are synchronized following a topological order of the parse tree of G.

Positive Goals The goal condition of the translated instance requires being in and every $q \in Q$ to be false. On the other hand, action *copy*, which has to be performed after each world action, has precisely the effect of making every $q \in Q$ false. This may significantly hurt performance if search relies on heuristics that relax negative effects of actions, like the FF heuristic (Hoffmann and Nebel 2001), which is key to the performance of state-of-the-art planning systems (Richter and Helmert 2009). To improve heuristic guidance, we define a new fluent q^D, for each $q \in Q$, with the intuitive meaning that q^D becomes true when $trans(q)$ cannot be executed in the future. For every action $trans(q_\alpha^S)$ that does not add q_α, we include the conditional effect $\{q_\beta \mid \beta \in super(\alpha)\} \to q_\alpha^D$, where $super(\alpha)$ is the set of subformulas of G that are proper superformulas of α. Using a function f that takes a $fLTL(F)$ formula and generates a propositional formula, the new goal $f(G)$ can be recursively written as follows:

- If $\varphi = p$ and $p \in Lit(F)$, then $f(p) = q_p^D$.
- If $\varphi = \alpha \wedge \beta$, then $f(\varphi) = q_\varphi^D \wedge f(\alpha) \wedge f(\beta)$
- If $\varphi = \alpha \vee \beta$, then $f(\varphi) = q_\varphi^D \wedge (f(\alpha) \vee f(\beta))$
- If $\varphi = \bigcirc\beta$ or $\varphi = \bullet\beta$, then $f(\varphi) = q_\varphi^D \wedge f(\beta)$
- If $\varphi = \alpha \star \beta$, where $\star \in \{\mathsf{U}, \mathsf{R}\}$, then $f(\varphi) = q_\varphi^D \wedge f(\beta)$

Empirical Evaluation

The objective of our evaluation was to compare our approach with existing translation approaches, over a range of general LTL goals, to understand when it is convenient to use one or other approach. We chose to compare to B&M's rather than Edelkamp's because the former seems to yield better performance (Baier, Bacchus, and McIlraith 2009). We do not compare against other existing systems that handle PDDL3 natively, such as LPRPG-P (Coles and Coles 2011), because efficient translations for the (restricted) subset of LTL of PDDL3 into NFA are known (Gerevini et al. 2009).

We considered both LAMA (Richter, Helmert, and Westphal 2008) and $\text{FF}_\mathcal{X}$ (Thiébaux, Hoffmann, and Nebel 2005), because both are modern planners supporting derived predicates (required by B&M). We observed that LAMA's preprocessing time where high, sometimes exceeding planning time by 1 to 2 orders of magnitude, and thus decided to report results we obtained with $\text{FF}_\mathcal{X}$. We used an 800MHz-CPU machine running Linux. Processes were limited to 1 GB of RAM and 15 min. runtime.

There are no planning benchmarks with general LTL goals, so we chose two of the domains (rovers and openstacks) of the 2006 International Planning Competition, which included LTL preferences (but not goals), and generated our own problems, with some of our goals inspired by the preferences. In addition, we considered the blocksworld.

Our translator was implemented in SWI-Prolog. It takes a domain and a problem in PDDL with an LTL goal as input and generates PDDL domain and problem files. It also receives an additional parameter specifying the translation mode which can be any of the following: *simple*, *OSA*, *PG*, and *OSA+PG*, where *simple* is the translation of Section , and *OSA*, *PG* are the optimizations described in Section . *OSA+PG* is the combination of *OSA* and *PG*.

Table 2 shows a representative selection of the results we obtained. It shows translation time (TT), plan length (PL), planning time (PT), the number of planning states that were evaluated before the goal was reached (PS). Times are displayed in seconds. For our translators we also include the length of the plan without synchronization actions (WPL). NR means the planner/translator did not return a plan.

We observe mixed results. B&M yields superior results on some problems: for example p01 and p03 of *openstacks*, which are of the form $\bigwedge_{i=1}^n \Diamond p_i$, where $n = 3$ and $n = 5$ for p01 and p03, respectively. The performance gap is probably due to the fact that (1) the B&M problem requires fewer actions in the plan and (2) B&M's output for these goals is quite compact on the size of the formula. On the other hand, there are other goal formulas in which our approach outperforms B&M. For example, p09 and p11 in openstacks, which are of the form: $\Diamond \bigwedge_{i=1}^n \Diamond p_i$, where $n = 5$ and $n = 4$ for p09 and p11, respectively. Even though the outer $\Diamond$ can be removed yielding an equivalent formula, B&M generates an output exponential in n, which results in higher translation time and eventually in the the planner running out of memory.

By observing the rest of the data, we conclude that B&M returns an output that is significantly larger than our approaches for the following classes of formulas: $\alpha \,\mathsf{U}(\bigwedge_{i=1}^n \Diamond\beta_i)$, $\alpha \,\mathsf{U}(\bigwedge_{i=1}^n \beta_i \,\mathsf{U}\, \gamma_i)$, $(\bigvee_{i=1}^n \Box\alpha_i) \,\mathsf{U}\, \beta$, and $(\bigvee_{i=1}^n \alpha_i \,\mathsf{R}\, \beta_i) \,\mathsf{U}\, \beta$, with $n \geq 4$, yielding finally an "NR". Being polynomial, our translation handles these formulas reasonably well: low translation times, and a compact output. In many cases, this allows the planner to return a solution.

The use of positive goals has an important influence in performance possibly because the heuristic is more accurate, leading to fewer expansions. OSA, on the other hand, seems to negatively affect planning performance in $\text{FF}_\mathcal{X}$. The reason is the following: $\text{FF}_\mathcal{X}$ will frequently choose the wrong synchronization action and therefore its enforced hill climbing algorithm will often fail. This behavior may not be observed in planners that use complete search algorithms.

Openstacks Domain

	B&M's translator				Non-PG + Non-OSA					Non-PG + OSA					PG + Non-OSA					PG + OSA				
	TT	PL	PT	PS	TT	PL	WPL	PT	PS	TT	PL	WPL	PT	PS	TT	PL	WPL	PT	PS	TT	PL	WPL	PT	PS
p01	0.246	23	0.02	34	0.455	142	21	5.41	196938	0.474	265	21	8.78	179157	0.462	166	23	0.05	1412	0.483	274	22	0.21	3100
p02	0.377	23	0.10	34	0.459	117	21	9.08	294097	0.481	287	21	10.57	202873	0.469	167	23	0.05	1413	0.491	297	22	0.23	3217
p03	0.268	25	0.02	36	0.477	199	23	78.71	1364638	0.503	433	23	48.62	834783	0.504	212	24	0.22	5905	0.536	448	24	1.01	12172
p04	22.390	0	NR	NR	0.478	133	23	149.88	1958261	0.518	457	23	53.27	876816	0.513	213	24	0.22	5906	0.548	473	24	1.05	12292
p05	0.256	24	0.10	214	0.466	166	21	20.15	533753	0.491	331	21	18.44	341998	0.484	197	22	0.41	10439	0.509	342	22	0.48	6498
p06	0.266	24	0.13	224	0.474	215	22	138.11	1892750	0.510	415	22	35.84	635715	0.506	231	22	1.53	35117	0.542	410	22	1.07	13364
p07	2.423	2	96.98	3	0.474	21	2	0.41	8907	0.504	49	2	0.95	14569	0.495	19	2	0.00	86	0.529	44	2	0.04	498
p08	4.567	2	0.00	3	0.477	20	2	2.58	51965	0.510	52	2	0.67	10967	0.505	32	2	0.07	1634	0.535	48	2	0.07	1002
p09	21.947	0	NR	NR	0.479	133	23	150.20	1958261	0.515	457	23	53.05	876816	0.515	213	24	0.22	5906	0.548	473	24	1.06	12292
p10	0.380	23	0.10	34	0.457	117	21	9.14	294097	0.480	287	21	10.62	202873	0.474	167	23	0.05	1413	0.499	297	22	0.22	3217
p11	1.599	0	NR	NR	0.472	125	22	31.93	755539	0.498	369	22	23.41	418240	0.489	192	24	0.10	2763	0.517	382	23	0.48	6176
p12	0.373	23	0.10	34	0.463	136	21	6.44	222245	0.486	309	21	10.86	203461	0.475	167	23	0.05	1413	0.501	319	22	0.25	3421
p13	1.594	0	NR	NR	0.470	156	22	22.49	592081	0.504	392	22	24.04	417585	0.496	192	24	0.10	2763	0.527	405	23	0.52	6573
p14	21.852	0	NR	NR	0.482	179	23	103.30	1573433	0.523	481	23	54.07	872446	0.525	213	24	0.23	5906	0.564	497	24	1.17	13074
Rovers Domain																								
p01	0.242	4	0.00	5	0.460	25	4	0.03	1757	0.471	31	4	0.04	1353	0.462	22	4	0.00	64	0.469	28	4	0.00	62
p02	0.255	7	0.00	10	0.468	45	7	0.55	25490	0.487	73	7	1.84	44002	0.475	42	7	0.01	316	0.489	69	7	0.01	181
p03	0.270	10	0.01	15	0.481	68	10	7.67	264568	0.501	133	10	41.78	522432	0.490	66	10	0.02	452	0.505	129	10	0.03	425
p04	0.407	10	0.05	16	0.481	62	10	68.39	1173227	0.507	144	10	46.04	544034	0.498	67	10	0.02	453	0.517	140	10	0.02	448
p05	1.639	0	NR	NR	0.494	0	0	NR	NR	0.539	1	0	NR	NR	0.517	106	15	0.05	1324	0.545	252	15	0.07	1201
p06	0.299	3	0.01	4	0.499	19	2	0.04	1522	0.528	49	2	0.04	847	0.525	17	3	0.00	43	0.548	47	3	0.02	301
p07	0.301	3	0.01	4	0.501	19	2	0.03	1290	0.537	49	2	0.05	962	0.522	26	5	0.01	148	0.554	78	5	0.20	3494
p08	NR	0	NR	NR	0.505	22	2	0.36	12158	0.542	52	2	0.17	3381	0.532	30	5	0.08	2291	0.567	83	5	0.40	6896
p09	2.010	2	0.00	3	0.504	20	2	0.07	2394	0.539	52	2	0.10	1769	0.537	27	5	0.02	200	0.567	83	5	0.25	4180
Blocksworld Domain																								
p01	0.234	2	0.00	3	0.452	24	2	0.11	3451	0.481	49	2	0.07	1612	0.470	21	2	0.00	34	0.502	43	2	0.00	106
p02	0.243	2	0.01	3	0.467	28	2	2.66	48436	0.508	61	2	1.08	14118	0.513	25	2	0.00	53	0.550	55	2	0.02	172
p03	0.251	2	0.01	3	0.486	32	2	84.40	637178	0.535	73	2	17.99	104398	0.566	29	2	0.01	81	0.609	67	2	0.04	256
p04	1.627	2	0.02	3	0.448	22	2	0.12	4115	0.472	46	2	0.04	792	0.464	19	2	0.00	32	0.490	41	2	0.01	146
p05	22.220	0	NR	NR	0.458	25	2	1.94	45113	0.492	55	2	0.31	4692	0.486	22	2	0.00	50	0.523	50	2	0.01	228
p06	471.574	0	NR	NR	0.471	28	2	38.99	474906	0.514	64	2	2.52	24761	0.522	25	2	0.01	77	0.559	59	2	0.04	330
p07	9.801	1	0.00	2	0.446	11	1	0.00	88	0.473	31	1	0.01	122	0.464	8	1	0.00	12	0.494	25	1	0.00	85
p08	423.327	1	0.00	2	0.463	11	1	0.01	172	0.494	37	1	0.02	248	0.490	8	1	0.00	16	0.523	31	1	0.01	147
p09	NR	0	NR	NR	0.473	11	1	0.02	285	0.519	43	1	0.05	492	0.527	8	1	0.00	22	0.566	37	1	0.02	269
p10	0.383	2	0.58	4	0.443	24	2	0.09	3607	0.470	46	2	0.01	357	0.465	22	2	0.00	101	0.489	41	2	0.01	97
p11	1.809	0	NR	NR	0.456	27	2	1.80	44666	0.490	55	2	0.06	1067	0.487	25	2	0.00	194	0.519	50	2	0.02	162
p12	25.655	0	NR	NR	0.470	30	2	48.30	610049	0.509	64	2	0.31	3325	0.520	28	2	0.04	514	0.558	59	2	0.04	287
p13	3.813	0	NR	NR	0.457	25	2	0.16	5680	0.489	52	2	0.09	1700	0.483	22	2	0.00	35	0.517	46	2	0.00	150
p14	182.181	0	NR	NR	0.476	29	2	6.16	122233	0.517	64	2	1.15	14576	0.532	26	2	0.01	54	0.567	58	2	0.03	257
p15	NR	0	NR	NR	0.495	0	0	NR	NR	0.547	76	2	19.13	106729	0.584	30	2	0.01	82	0.638	70	2	0.06	396

Table 2: Experimental results for a variety of LTL planning tasks.

Conclusions

We proposed polynomial-time translations of LTL into final-state goals, which, unlike existing translations are optimal with respect to computational complexity. The main difference between our approach and state-of-the-art NFA-based translations is that we use AA, and represent a single run of the AA in the planning state. We conclude from our experimental data that it seems more convenient to use an our AA translation precisely when the output generated by the NFA-based translation is exponentially large in the size of the formula. Otherwise, it seems that NFA-based translations are more efficient because they do not require synchronization actions, which require longer plans, and possibly higher planning times. Obviously, a combination of both translation approaches into one single translator should be possible. Investigating such a combination is left for future work.

References

Bacchus, F., and Kabanza, F. 1998. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence* 22(1-2):5–27.

Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence* 116(1-2):123–191.

Baier, J. A., and McIlraith, S. A. 2006. Planning with first-order temporally extended goals using heuristic search. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI)*, 788–795.

Baier, J. A.; Bacchus, F.; and McIlraith, S. A. 2009. A heuristic search approach to planning with temporally extended preferences. *Artificial Intelligence* 173(5-6):593–618.

Baier, J. A. 2010. *Effective Search Techniques for Non-Classical Planning via Reformulation*. Ph.D. in Computer Science, University of Toronto.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artificial Intelligence* 69(1-2):165–204.

Coles, A. J., and Coles, A. 2011. LPRPG-P: relaxed plan heuristics for planning with preferences.

Cresswell, S., and Coddington, A. M. 2004. Compilation of LTL goal formulas into PDDL. In de Mántaras, R. L., and Saitta, L., eds., *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)*, 985–986. Valencia, Spain: IOS Press.

De Giacomo, G., and Vardi, M. Y. 1999. Automata-theoretic approach to planning for temporally extended goals. In Biundo, S., and Fox, M., eds., *ECP*, volume 1809 of *LNCS*, 226–238. Durham, UK: Springer.

De Giacomo, G., and Vardi, M. Y. 2013. Linear temporal logic and linear dynamic logic on finite traces. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*.

De Giacomo, G.; Masellis, R. D.; and Montali, M. 2014. Reasoning on LTL on finite traces: Insensitivity to infiniteness. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI)*, 1027–1033.

Edelkamp, S.; Jabbar, S.; and Naizih, M. 2006. Large-scale optimal PDDL3 planning with MIPS-XXL. In *5th International Planning Competition Booklet (IPC-2006)*, 28–30.

Edelkamp, S. 2006. On the compilation of plan constraints and preferences. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS)*.

Gerevini, A.; Haslum, P.; Long, D.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence* 173(5-6):619–668.

Gerth, R.; Peled, D.; Vardi, M. Y.; and Wolper, P. 1995. Simple on-the-fly automatic verification of linear temporal logic. In *Proceedings of the 15th International Symposium on Protocol Specification, Testing and Verification (PSTV)*, 3–18.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Muller, D. E.; Saoudi, A.; and Schupp, P. E. 1988. Weak Alternating Automata Give a Simple Explanation of Why Most Temporal and Dynamic Logics are Decidable in Exponential Time. In *Proceedings of the 3rd Annual Symposium on Logic in Computer Science (LICS)*, 422–427.

Pnueli, A. 1977. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science (FOCS)*, 46–57.

Richter, S., and Helmert, M. 2009. Preferred operators and deferred evaluation in satisficing planning. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI)*, 975–982.

Rintanen, J. 2000. Incorporation of temporal logic control into plan operators. In Horn, W., ed., *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI)*, 526–530. Berlin, Germany: IOS Press.

Thiébaux, S.; Hoffmann, J.; and Nebel, B. 2005. In defense of PDDL axioms. *Artificial Intelligence* 168(1-2):38–69.

Torres, J., and Baier, J. A. 2015. Polynomial-time reformulations of ltl temporally extended goals into final-state goals. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*.

A Unifying Framework for Planning with LTL and Regular Expressions

Eleni Triantafillou
Department of Computer Science
University of Toronto
Toronto, Canada

Jorge A. Baier
Depto. de Ciencia de la Computación
Pontificia Universidad Católica de Chile
Santiago, Chile

Sheila A. McIlraith
Department of Computer Science
University of Toronto
Toronto, Canada

Abstract

Temporally extended goals are critical to the specification of many real-world planning problems. Such goals are typically specified in the subset of Linear Temporal Logic (LTL) found in the Planning Domain Description Language, PDDL3.0. In this paper, we propose LTL-RE, a high-level language that supports the specification of a wide variety of temporal goals, not only using LTL but also using regular expressions. LTL-RE derives its formal foundation from finite Linear Dynamic Logic ($\mathsf{LDL_f}$), and its expressive power is no less than that of regular expressions. LTL-RE augments $\mathsf{LDL_f}$ with planning-friendly syntax including LTL and typical programming language constructs. It is also designed for use with AI automated planning transition systems, supporting both state-, action-, and path-oriented temporal goal specification. Building on recent work focused on LTL, we propose a translation of LTL-RE into Alternating Automata, which are then embedded directly in domain descriptions for use with classical planners. We evaluate the behavior of our translator and the resultant planning problems, with comparison to alternative LTL translators.

1 Introduction

Most real-world planning problems involve complex goals that are temporally extended, necessitate the optimization of preferences or other quality measures, require adherence to safety constraints and directives, and/or may require or benefit from following a prescribed high-level script that specifies *how* the task is to be realized. By way of illustration, consider a logistics company that transports packages. Package delivery may be governed by the following types of (possibly inconsistent) goals:

- *Always ship frozen food in a refrigerated truck.*
- *Prefer to deliver priority packages before regular packages.*
- *If the customer is a preferred customer, then always apply a 15% discount to the final bill.*
- *Prefer to deliver domestic packages within 24 hours of receipt.*
- *While a truck is at a location and not full, load all packages bound for a different destination on the truck; drive to the next destination; unload all packages to be delivered to this destination.*

While some forms of non-classical goal[1] specification were initially realized via special-purpose planners such as the Hierarchical Task Network (HTN) planner SHOP2 (e.g., (Erol, Hendler, and Nau 1994)) or TLPLAN, the pioneering planning system that accepts Linear Temporal Logic (LTL) pruning rules (Bacchus and Kabanza 1998), more recent efforts have focused on incorporating such non-classical goals, which include both temporally extended goals and preferences, into state-of-the-art domain independent planners (e.g., (Rintanen 2000; Doherty and Kvarnström 2001; Cresswell and Coddington 2004; Edelkamp 2006; Baier and McIlraith 2006; Benton, Kambhampati, and Do 2006; Baier, Fritz, and McIlraith 2007; Coles and Coles 2011; Lago, Pistore, and Traverso 2002)). Such systems have been used in service of a diversity of planning and non-planning applications from genomic rearrangement (Uras and Erdem 2010) and program test generation (Razavi, Farzan, and McIlraith 2014) to story generation (Haslum 2012), automated diagnosis (Grastien et al. 2007; Sohrabi, Baier, and McIlraith 2010), and verification (Albarghouthi, Baier, and McIlraith 2009; Patrizi et al. 2011). In recognition of the planning community's need for non-classical planning objectives, PDDL3.0 (Gerevini et al. 2009) was designed to capture a useful subset of LTL constraints which can either be cast as hard constraints on the plan or aggregated in a weighted sum to construct an objective function of soft constraints for optimization in the context of plan generation.

The provision of planning systems that accept temporally extended goals is at the heart of synergies between the AI Automated Planning community and the Model Checking community, where similar types of constraints are used to specify safety and liveness properties for software and hardware verification, and to specify target behavior for the synthesis of software and controllers. Historically the model checking community has specified such properties in a temporal logic such as LTL, or one of its branching time counterparts – CTL or CTL*. Such temporal logics are very good at specifying *state-centric* properties but they don't provide a natural vehicle for specifying *action-centric* properties – procedural properties involving the actions of a domain.

In a series of well-received lectures between 2011 and 2013, Moshe Vardi advocated convincingly for both the benefits of LTL, but also for its limitations in the context of industry-driven verification tasks (e.g., (Vardi 2012)). In response, Vardi advocated for Linear Dynamic Logic (LDL), a temporal logic that combines LTL and Regular Expressions (REs) in a manner that avoids the exponential blowup that typically plagues REs in such a context. Subsequently, De Giacomo and Vardi (2013), proposed $\mathsf{LDL_f}$, which defines

[1] A classical planning goal is limited to a conjunction of properties that must hold in the final state.

LDL over finite traces, citing automated planning among the applications for the logic.

While the AI planning community has increasingly studied state-centric path constraints in the form of temporally extended goals (TEGs), there has been far less examination of temporally extended goals that take the form of REs. An exception to this is the work of Baier, Fritz, and McIlraith (2007; 2008) which supports planning with action-centric procedural control/goals in a Golog-like language that captures the syntax of REs. A second exception is the work by Shaparau, Pistore, and Traverso (2008) which, building on previous work on the EAGLE goal language (Lago, Pistore, and Traverso 2002), also provides a form of REs for temporally extended goal specification.

In this paper we propose a goal specification language, LTL-RE, that supports both LTL and REs. However, unlike previous work noted above, it has its formal underpinnings in one uniform language – $\mathsf{LDL_f}$ – capturing the semantics of $\mathsf{LDL_f}$ while at the same time augmenting $\mathsf{LDL_f}$ with further syntax that we believe is more compelling to an end user charged with specifying goals or constraints for plan generation. LTL- RE is able to specify goals with respect to planning domain actions as well as state properties in the form of REs, using compelling programming constructs such as "if then else" and "while" loops. We define the syntax and semantics of LTL- RE and examine how to plan for LTL-RE goals using state-of-the-art domain independent planners via a reformulation into finite state automata. Unlike previous reformulation approaches that exploited Nondeterministic Finite State Automata (NFAs) (e.g., (Baier and McIlraith 2006)), we exploit an approach based on Alternating Automata following Torres and Baier (2015) that avoids the worst-case exponential blow-up inherent to NFAs. This workshop paper represents a work in progress. We present our algorithm and report on experimental results to date, contrasting the efficacy of our reformulation to LTL-specific LDL reformulations based on NFAs and Alternating Automata.

2 Preliminaries

In this section we recount how transition systems are compactly described using a planning language and review the syntax and semantics of LTL, LDL and their finite trace counterparts, $\mathsf{LTL_f}$ and $\mathsf{LDL_f}$.

2.1 A Planning-Language Transition System

The objective of this paper is to show how to plan for a rich goal language based on LTL and REs uniformly captured in $\mathsf{LDL_f}$. We assume that the "world" for which we want to build our plan is described by a deterministic transition system compactly described by an initial state and a set of actions. In AI automated planning such a transition system is typically specified using the Planning Domain Description Language (PDDL) of which there are several variants of differing expressivity (McDermott 1998).

Formally, a transition system is given by a tuple $(P, \mathcal{A}, \mathcal{I})$, where P is a set of propositions, which we use to describe a state, $\mathcal{A}$ is a set of actions, and $\mathcal{I} \subseteq P$ is the *initial state*. For every action $a \in \mathcal{A}$, $prec(a)$ and $\mathit{eff}(a)$ denote, respectively, the preconditions and effects of a. $prec(a)$ is a set of fluent literals over P and $\mathit{eff}(a)$ is a set of elements of the form $C \to L$, where C is a set of literals over P and L is a literal over P. When $C \to L$ is an effect of a, and a is applied on a state s in which C holds, then L must hold in the state that results from applying a in s.

An action a is applicable in a state $s \subseteq 2^P$ if $s \models prec(a)$. If a is applicable in s, then partial function $\delta : 2^{2^P} \times A$ is defined such that:

$$\delta(s, a) = s \backslash \{p \mid C \to \neg p \in \mathit{eff}(a)\} \cup \{p \mid C \to p \in \mathit{eff}(a)\}$$

If a is not applicable in s, then $\delta(a, s)$ is undefined.

A sequence of actions $a_0 a_1 \dots a_{n-1}$ is applicable in s_0 if $\delta(s_i, a_i)$ is defined for each $i \in \{0, \dots, n-1\}$. A state trace $s_0 s_1 \dots s_{n+1}$ is *induced* by the execution of $\alpha = a_0 a_1 \dots a_n$ in a state s iff (1) α is applicable in s, (2) $s = s_0$, and (3) $\delta(s_i, a_i) = s_{i+1}$, for every $i \in \{0, \dots, n-1\}$.

2.2 From Propositional Dynamic Logic to LDL

Propositional Dynamic Logic (PDL) was introduced by Fischer and Ladner (1979) to describe interesting properties of programs, such as correctness and termination. In PDL, terms are actions and propositions, and modal operators are exploited to directly reference regular programs within the language. This allows, for instance, the use of a test operator which results in the blocking of the program if the property that is being tested is false. It also supports nondeterministic choice of actions, sequencing of actions in a program, and the repetition of a program for a nondeterministic number of iterations. Within PDL, it is also possible to define programming constructs such as "if then else" and "while do".

LDL is an extension of PDL, which carries over the rich expressive properties of PDL but interpretted with respect to linear traces, just as LTL is used in planning and model checking to express interesting state-centric properties of linear traces (e.g., TEGs). LDL is a logic that is expressively equivalent to Monadic Second Order Logic (MSO), and is strictly more powerful than first order logic (FOL), or equivalently, LTL. In their 2013 paper, De Giacomo and Vardi argue that $\mathsf{LDL_f}$ witnesses the marriage of the best properties of REs on finite traces ($\mathsf{RE_f}$) and $\mathsf{LTL_f}$, namely the rich expressivity of RE with the declarative convenience $\mathsf{LTL_f}$ (De Giacomo and Vardi 2013). Since, however, in our opinion $\mathsf{LDL_f}$ is still not a very intuitive specification mechanism, we augment it with syntax to allow for a more intuitive expression of temporal and dynamic properties and clarify its use in the context of a transition system expressed via a planning language, such as PDDL.

2.3 $\mathsf{LTL_f}$ and $\mathsf{LDL_f}$

$\underline{\mathsf{LTL_f}}$: LTL is a modal temporal logic, first proposed for verification (Pnueli 1977). It supports the expression of rich path properties using modalities that include always ($\Box$), eventually ($\Diamond$), until (U), and next ($\bigcirc$). These temporal modalities can be arbitrarily nested over well-formed formulae defined over standard logical constructs such at $\neg$, $\vee$, $\wedge$, etc. $\mathsf{LTL_f}$ is a finite variant of LTL that has been used extensively for the specification of TEGs in automated planning. Below we review the semantics of $\mathsf{LTL_f}$.

Given a finite trace π over an alphabet 2^P, and an instant, i of the trace, the $\mathsf{LTL_f}$ operators are defined below.

- $\pi, i \models \bigcirc\varphi$ iff $i < last \wedge \pi, i+1 \models \varphi$
- $\pi, i \models \varphi_1 \,\mathsf{U}\, \varphi_2$ iff for some j such that $i \leq j \leq last$, we have that $\pi, j \models \varphi_2$ and for all $k, i \leq k < j$, we have that $\pi, k \models \varphi_2$

The operators $\Box$ and $\Diamond$ can be defined in terms of the above modal operators. Intuitively, $\Box\phi$ denotes that formula ϕ holds in every state of the trace from the current instant forward, while $\Diamond\phi$ denotes that ϕ will hold at some instant in the subtrace from the current instant forward. More formally,

- $\pi, i \models \Diamond\varphi$ iff $\pi, i \models true \,\mathsf{U}\, \varphi$
- $\pi, i \models \Box\varphi$ iff $\pi, i \models \neg\Diamond\neg\varphi$

$\underline{\mathsf{LDL_f}}$: $\mathsf{LDL_f}$ (De Giacomo and Vardi 2013) features the properties of PDL, but formulae are evaluated over finite linear traces. The syntax of $\mathsf{LDL_f}$ is defined as follows:

$\varphi ::= A \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle\rho\rangle\varphi$

$\rho ::= \phi \mid \varphi? \mid \rho_1 + \rho_2 \mid \rho_1;\rho_2 \mid \rho^*$

where A denotes atomic propositions, ϕ denotes a propositional formula over atomic propositions, ρ denotes path expressions, which are regular expressions over propositional formulas ϕ, together with the test construct ?. Finally, φ denotes $\mathsf{LDL_f}$ formulas formed by applying Boolean connectives, combined with the modal operator $\langle\rho\rangle\varphi$.

The modal operator $\langle\rho\rangle\varphi$ evaluates to $true$ in a state if there exists a trace, starting from the current state, which satisfies ρ and ends in a state which satisfies φ. Its dual operator, $[\rho]\varphi$, which can be defined as $\neg\langle\rho\rangle\neg\varphi$, is $true$ in a state if all traces starting from that state which satisfy ρ end in a state that satisfies φ.

For a given finite trace π over an alphabet 2^P, and an instant, i, of the trace, $i \in \{0, \ldots, last\}$, we inductively define what it means for an $\mathsf{LDL_f}$ formula φ to be $true$, i.e $\pi, i \models \varphi$:

- $\pi, i \models A$, for $A \in P$ iff $A \in \pi(i)$
- $\pi, i \models \neg\varphi$ iff $\pi, i \not\models \varphi$
- $\pi, i \models \varphi \wedge \varphi'$ iff $\pi, i \models \varphi$ and $\pi, i \models \varphi'$
- $\pi, i \models \langle\rho\rangle\varphi$ iff for some j such that $i \leq j \leq last$, we have that $(i, j) \in R(\rho, \pi)$ and $\pi, j \models \varphi$

where the relation $R(\rho, s)$ is defined inductively as follows:

- $R(\phi, s) = \{(i, i+1) \mid \pi(i) \models \phi\}$ (ϕ propositional)
- $R(\phi?, s) = \{(i, i) \mid \pi, i \models \phi\}$
- $R(\rho_1 + \rho_2, s) = R(\rho_1, s) \cup R(\rho_2, s)$
- $R(\rho_1; \rho_2, s) = \{(i, j) \mid exists\ k$ such that $(i, k) \in R(\rho_1, s)$ and $(k, j) \in R(\rho_2, s)$
- $R(\rho^*, s) = \{(i, i) \cup (i, j) \mid exists\ k$ such that $(i, k) \in R(\rho, s)$ and $(k, j) \in R(\rho^*, s)\}$

3 A Goal Language over LTL & REs

3.1 Overview

In this paper we propose Linear Temporal Logic with Regular Expressions for finite traces (LTL-RE)[2], a high-level language for the specification of temporally extended goals that are evaluated over finite traces. This language functions as a unifying framework, by supporting syntax from LTL and LDL, programming constructs "**if-then-else**" and "**while**", and a modality "**final**", for expressing properties that must hold in the last state of a finite trace. It also supports direct reference to planning language actions from within LTL-RE through the use of a special predicate, "**occ**" which ranges over the ground actions in a planning problem description, $\mathcal{A}$.

LTL-RE is as expressive as $\mathsf{LDL_f}$, which is equivalent to Monadic Second Order Logic. This is strictly more expressive than $\mathsf{LTL_f}$. This fact allows the definition of LTL operators, and the constructs "if then else", "while" and "final" in terms of the syntax of $\mathsf{LDL_f}$, as we demonstrate in a following section.

3.2 The Syntax of LTL-RE

Given a transition system $(P, \mathcal{A}, \mathcal{I})$, as defined in Section 2.1, the syntax of LTL-RE is given by the following grammar:

$$\begin{aligned}
\phi ::=& p : p \in P \mid \mathbf{occ}(a) : a \in \mathcal{A} \mid \neg\phi \mid \phi_1 \wedge \phi_2 \\
\varphi ::=& \phi \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathbf{final}\ \phi \mid \langle\rho\rangle\varphi \mid \varphi_1 \,\mathsf{U}\, \varphi_2 \mid \\
& \bigcirc\varphi \mid \Box\varphi \mid \Diamond\varphi \\
\rho ::=& \phi \mid \varphi? \mid \rho_1 + \rho_2 \mid \rho_1;\rho_2 \mid \rho^* \mid \\
& \textbf{if-then-else}(\varphi, \rho_1, \rho_2) \mid \mathbf{while}(\varphi, \rho)
\end{aligned}$$

With this syntax, we can express typical LTL goals such as "Always have your phone and eventually be at home."

$$\Box have(Phone) \wedge \Diamond at(Home)$$

but also TEGs that take the form of regular expressions, such as "If it's night time then take a taxi home, else take the subway."

$$\textbf{if-then-else}(night, \mathbf{occ}(taxi(Home)), \mathbf{occ}(subway(Home)))$$

3.3 Semantics

LTL-RE is interpreted over a pair $\pi = (\sigma, \alpha)$, where α is a sequence of actions, and σ is the sequence of states induced by the execution of α in a certain state.

We say that $\pi \models \varphi$, where φ is an LTL-RE formula, $\pi = (\sigma, \alpha)$, $\sigma = s_0 \ldots s_n$, and $\alpha = a_0 \ldots a_{n-1}$ iff $\pi, 0 \models \varphi$. Now we assume we include the same definitions in $\mathsf{LDL_f}$'s semantics that were listed in the previous section, taking into account that $\pi(i)$ now refers the the i-th state, i.e., s_i. In addition we add the following rule for the **occ** operator:

- $\pi, i \models \mathbf{occ}(a)$ iff $i < n$ and $a_i = a$

[2]not to be confused with RELTL (e.g., (Eisner and Fisman 2007)).

This operator is what makes it possible to directly refer to actions within the language, and not merely through their effects on the state properties. Specifically, **occ**(a) is true in the current state, if a is the next planning action to be executed.

Now we define the semantics for the "**if-then-else**" and "**while**" programming constructs. Following (Fischer and Ladner 1979), we can express these constructs in terms of standard LDL operators.

- $\pi, i \models$ **if-then-else**$(\psi, \varphi_1, \varphi_2)$ if $\pi, i \models \psi?; \varphi_1 + \neg\psi?; \varphi_2$
- $\pi, i \models$ **while**(ψ, φ) if $\pi, i \models (\psi?; \varphi)^*; \neg\psi$

We also define the semantics for the LTL operators $\bigcirc$, $\Diamond$, $\Box$ and U can be rewritten using the syntax of $\mathsf{LDL_f}$ while preserving their semantics as defined in the previous section. This is shown in (De Giacomo et al. 2014).

- $\pi, i \models \bigcirc\varphi$ iff $\pi, i \models \langle true \rangle \varphi \wedge i < last$
- $\pi, i \models \Diamond\varphi$ iff $\pi, i \models \langle true^* \rangle \varphi$
- $\pi, i \models \Box\varphi$ iff $\pi, i \models [true^*]\varphi$
- $\pi, i \models \psi \,\mathsf{U}\, \varphi$ iff $\pi, i \models \langle (\psi?; true)^* \rangle \varphi$

Finally, we define the semantics for the modality **final**:

- $\pi, i \models$ **final** φ iff $\pi, i \models \varphi \wedge i = last$

3.4 Planning for an LTL-RE Goal

We end this section by defining what it means to plan for an LTL-RE goal.

Definition 3.1. *Let $R = (P, \mathcal{A}, \mathcal{I})$ be a transition system and φ be an* LTL-RE *formula. Then the sequence of action α is a plan for φ over R iff α is applicable in $\mathcal{I}$ and generates a state trace σ over $\mathcal{I}$ such that $(\sigma, \alpha) \models \varphi$.*

4 Planning for LTL-RE Goals with Standard Planners

In this section we show how we can plan for LTL-RE goals using state-of-the-art planners. To this end, we use a two-step approach that follows (Torres and Baier 2015). In the first stage, we build an alternating automaton for the LTL-RE formula. Then, we show how this automaton can be used to compile the temporal goal into a non-temporal (final-state) goal. We do this by exploiting the fact that the dynamics of an alternating automaton can be encoded efficiently in a new transition system that is built from the original planning domain transition system.

4.1 Alternating Automata on Words

An Alternating Automaton on Words (AA) on the alphabet 2^P is a tuple A defined as: $A = (2^P, Q, q_0, \delta, F)$, where Q is a finite nonempty set of states, q_0 is the initial state, F is a set of accepting states, and δ is a transition function $\delta : Q \times 2^P \to B^+(Q)$, where $B^+(Q)$ is a set of positive Boolean formulas whose atoms are states of Q.

A *run* of an AA $A = (2^P, Q, q_0, \delta, F)$ over word $w = b_1 \dots b_n$ is a sequence of subsets of Q, $Q_0 Q_1 \dots Q_n$, such that $Q_0 = \{q_0\}$, and $Q_{i+1} \models \delta(q, b_i)$, for every $q \in Q_i$, and every $i \in \{0, \dots, n-1\}$. An AA accepts a word w if it has a run ending in a subset of F.

4.2 From $\mathsf{LDL_f}$ to AA

Following (Fischer and Ladner 1979; De Giacomo and Vardi 2013), the Fisher-Ladner Closure of a $\mathsf{LDL_f}$ formula φ is a set CL_φ of $\mathsf{LDL_f}$ formulas, recursively defined as follows:

$\varphi \in CL_\varphi$
$\neg\psi \in CL_\varphi$ if $\psi \in CL_\varphi$ and ψ not of the form $\neg\psi'$
$\varphi_1 \wedge \varphi_2 \in CL_\varphi$ implies $\varphi_1, \varphi_2 \in CL_\varphi$
$\langle\rho\rangle\varphi \in CL_\varphi$ implies $\varphi \in CL_\varphi$
$\langle\phi\rangle\varphi \in CL_\varphi$ implies $\phi \in CL_\varphi$ (ϕ propositional)
$\langle\psi?\rangle\varphi \in CL_\varphi$ implies $\psi \in CL_\varphi$
$\langle\rho_1; \rho_2\rangle\varphi \in CL_\varphi$ implies $\langle\rho_1\rangle\langle\rho_2\rangle\varphi \in CL_\varphi$
$\langle\rho_1 + \rho_2\rangle\varphi \in CL_\varphi$ implies $\langle\rho_1\rangle\varphi, \langle\rho_2\rangle\varphi \in CL_\varphi$
$\langle\rho^*\rangle\varphi \in CL_\varphi$ implies $\langle\rho\rangle\langle\rho^*\rangle\varphi \in CL_\varphi$

De Giacomo and Vardi define an AA which accepts all and only the traces that satisfy a given $\mathsf{LDL_f}$ formula. Specifically, given a set of propositions P, and a $\mathsf{LDL_f}$ formula φ which is in Negation Normal Form (NNF), the automaton for φ that is defined in (De Giacomo and Vardi 2013) is given by: $A'_\varphi = (2^P, CL_\varphi, \varphi, \delta', \{\})$, where 2^P is the alphabet, CL_φ, denoting the Fisher-Ladner Closure of the formula φ is the state set, and δ' is their transition function.

We define an automaton A_φ for a set of propositions P and a $\mathsf{LDL_f}$ formula in NNF φ as follows: $A_\varphi = (2^P, CL_\varphi \cup \{q_F\}, \varphi, \delta, \{q_F\})$, where q_F is a special automaton state in which the AA transitions when the trace has ended. Also, δ is the transition function, which differs from the aforementioned δ' only in the transition for $[\phi]\varphi$, as we elaborate on later. For an interpretation Π, and assuming that A stands for a propositional formula, we define δ below.

$\delta(A, \Pi) = true$ if $A \in \Pi$
$\delta(A, \Pi) = false$ if $A \notin \Pi$
$\delta(q_F, \Pi) = false$
$\delta(\varphi_1 \wedge \varphi_2, \Pi) = \delta(\varphi_1) \wedge \delta(\varphi_1)$
$\delta(\varphi_1 \vee \varphi_2, \Pi) = \delta(\varphi_1) \vee \delta(\varphi_1)$

$$\delta(\langle\phi\rangle\varphi, \Pi) = \begin{cases} \varphi \text{ if } \Pi \models \phi & (\phi \text{ propositional}) \\ false \text{ if } \Pi \not\models \phi \end{cases}$$

$\delta(\langle\psi?\rangle\varphi, \Pi) = \delta(\psi, \Pi) \wedge \delta(\varphi, \Pi)$
$\delta(\langle\rho_1 + \rho_2\rangle\varphi, \Pi) = \delta(\langle\rho_1\rangle\varphi, \Pi) \vee \delta(\langle\rho_2\rangle\varphi, \Pi)$
$\delta(\langle\rho_1; \rho_2\rangle\varphi, \Pi) = \delta(\langle\rho_1\rangle\langle\rho_2\rangle\varphi, \Pi)$

$$\delta(\langle\rho^*\rangle\varphi, \Pi) = \begin{cases} \delta(\varphi, \Pi) \text{ if } \rho \text{ is test-only} \\ \delta(\varphi, \Pi) \vee \delta(\langle\rho\rangle\langle\rho^*\rangle\varphi, \Pi) \text{ o/w} \end{cases}$$

$$\delta([\phi]\varphi, \Pi) = \begin{cases} \varphi \vee q_F \text{ if } \Pi \models \phi & (\phi \text{ propositional}) \\ true \text{ if } \Pi \not\models \phi \end{cases}$$

$\delta([\psi?]\varphi, \Pi) = \delta(nnf(\neg\psi), \Pi) \vee \delta(\varphi, \Pi)$
$\delta([\rho_1 + \rho_2]\varphi, \Pi) = \delta([\rho_1]\varphi, \Pi) \wedge \delta([\rho_2]\varphi, \Pi)$
$\delta([\rho_1; \rho_2]\varphi, \Pi) = \delta([\rho_1][\rho_2]\varphi, \Pi)$

$$\delta([\rho^*]\varphi, \Pi) = \begin{cases} \delta(\varphi, \Pi) \text{ if } \rho \text{ is test-only} \\ \delta(\varphi, \Pi) \wedge \delta([\rho][\rho^*]\varphi, \Pi) \text{ o/w} \end{cases}$$

It is important to note that a $\mathsf{LDL_f}$ formula can be rewritten to an equivalent $\mathsf{LDL_f}$ formula which is in NNF in linear time. Further, that the state set of the AA for a $\mathsf{LDL_f}$ formula φ, namely CL_φ is linear in the size of φ.

Theorem 4.1. *Let φ be an $\mathsf{LDL_f}$ formula and A_φ the AA defined above. Then, for any interpretation π, $\pi \models \varphi$ iff A_φ accepts π.*

Proof. The correctness of the theorem stems from Theorem 17 in (De Giacomo and Vardi 2013). Our only modification to the AA presented in that paper is in the transition $\delta([\phi]\varphi, \Pi)$: In the first of the two cases for this transition (ie when $\Pi \models \phi$), we add the disjunction with q_f. To see why this is necessary, consider the case where Π contains a single state, and in that state ϕ is true and φ is false. Then, since $\Pi \models \phi$, we are in the first of the two cases of this transition, so we transition to a state in which $\varphi \vee q_f$, ie $\mathsf{false} \vee q_f$, is true. Had we not included q_f in this disjunction, the automaton would not accept this trace, which is an incorrect behavior. By allowing q_f as an option for this transition, we provide the automaton with the choice to end the trace and accept it, as it should. $\square$

4.3 Building an AA for LTL-RE

LTL- RE augments $\mathsf{LDL_f}$ with LTL, programming constructs, and a final modality, in order to make goal specification easier. These constructs can all be defined in terms of native $\mathsf{LDL_f}$. The most significant extension of LTL-RE over $\mathsf{LDL_f}$, from the perspective of translation, is the addition of the **occ** predicate that enables a goal to reference the occurrence of a particular ground action.

Let $(P, \mathcal{A}, \mathcal{I})$ be a transition system, α be a sequence of action applicable in $\mathcal{I}$, and σ be the state trace that is induced by the execution of α in $\mathcal{I}$. Then we define a word $\beta = b_0 b_1 \ldots b_n$ in which $b_i = s_i \cup \{a_i\}$, for $i \in \{0, \ldots, n-1\}$, and $b_n = s_n$.

Given an LTL-RE formula φ, we first replace all program constructs ("if-then-else" and "while"), as well as any LTL constructs by the corresponding $\mathsf{LDL_f}$ equivalent given by the semantics defined in the previous section (i.e., we replace any occurrence of $\Box\varphi$ by $[true^*]\varphi$, and so forth). Then, as with $\mathsf{LDL_f}$ we put the resulting formula in negation normal form. Let φ' be the resulting formula. The AA for the resulting formula is like $A_{\varphi'}$ described above for $\mathsf{LDL_f}$, but includes the following additional definition for δ:

$$\delta(\mathbf{occ}(a), \Pi) = occ(a)$$

where $occ(a)$ is the special fluent from the set Occ, which is described in a following paragraph. By making $occ(a)$ true, we ensure that the ground action a must be the next action of the plan in order for this trace to be accepting. The automaton for φ that results from applying these steps is denoted as A_φ.

Theorem 4.2. *Let A_φ, σ be a state trace induced by the execution of action sequence α, and β be defined as above. Then $(\sigma, \alpha) \models \varphi$ iff A_φ accepts β.*

4.4 Compiling away LTL-RE goals

Now we describe a method to compile away LTL-RE goals by representing the AA within a new (output) transition system, constructed from the original planning transition system. Our method is based on the one proposed in (Torres and Baier 2015), which in their case compiled away LTL planning goals expressed as an AA. Although Torres and Baier only deal with LTL rather than LDL, the main technical difference between their method and ours is the treatment of action-centric constraints and in particular the special handling of the **occ** predicate in order to support REs over actions.

Given a transition system $T = (P, \mathcal{A}, \mathcal{I})$, and an LTL-RE goal φ, our method generates a new transition system $T' = (P', \mathcal{A}', \mathcal{I}')$. A plan for T and goal φ can be obtained by finding a sequence of action that reaches a final state where a distinguished property in t' holds. Satisfaction of this property corresponds to successful transitioning through the automaton A_φ. Thus, the problem of planning for a temporal goal is reduced to the problem of finding a classical plan for which this distinguished property holds. Such a plan can be obtained using optimized off-the-shelf classical planners.

Following (Torres and Baier 2015), in T' there is one (new) fluent q for each state q of A_φ. If $\alpha = a_1 a_2 \ldots a_n$ is applicable in the initial state of T, then there will exist a corresponding *set of action sequences* (denoted $\mathcal{A}_\alpha$) of the form $\alpha_0 a_1 \alpha_1 a_2 \alpha_2 \ldots a_n \alpha_n$, where each α_i is a sequence of so-called "synchronization actions" which did not appear in P and whose objective is to update the state of A_φ.

Also in keeping with Torres and Baier, all actions in T also appear in T'. Execution in T' can be understood as having two "modes". In the so-called *world mode*, actions from the original transition system T can be executed. In the so-called *synchronization mode* actions that update the state of the automaton can be executed. The set of propositions P' contains additional propositions representing the state of the AA for φ plus additional flags that are used to switch appropriately between modes. Synchronization actions update the state of the automaton following the definition of the δ function.

Fluents P' has the same fluents as P plus fluents that represent the states of the automaton (Q), flags for controlling the different modes ($\mathbf{copy}$, $\mathbf{sync}$, $\mathbf{world}$), and a special fluent $\mathbf{ok}$, which becomes false if the goal has been falsified. Finally, it includes the set $Q^S = \{q^S \mid q \in Q\}$ which are "copies" of the automata fluents (described in detail below), and Occ which contains a fluent $occ(a')$ for each a such that $occ(a)$ is a subformula of the original goal formula φ. Formally, the set of fluents $F' = F \cup Q \cup Q^S \cup \{\mathbf{copy}, \mathbf{sync}, \mathbf{world}, \mathbf{ok}\} \cup Occ$.

The set of planning operators O' is the union of the sets O_w and O_s for the world-mode and synchronization-mode actions, as follows.

World Mode Operators The set O_w contains the same actions in $\mathcal{A}$, but preconditions are modified to allow execution only in "world mode". Effects, on the other hand, are modified to allow the execution of the *copy* action, which initiates the synchronization phase, and which is described below. Formally, $O_w = \{a' \mid a \in \mathcal{A}\}$, and for all a' in O_w:

$$prec(a') = prec(a) \cup \{\mathbf{ok}, \mathbf{world}\} \cup notOcc(a'),$$
$$eff(a') = eff(a) \cup \{\mathbf{copy}, \neg\mathbf{world}\},$$

where $notOcc(a') = \{\neg occ(a) \mid a \neq a' \text{ and } occ(a) \in Occ\}$. I.e., the preconditions for executing a' are all the original preconditions for action a, that the flags indicate the planner is in world mode and the goal has not been falsified,

Sync Action	Precondition	Effect
$tr(q^S_\ell)$	$\{\mathbf{sync}, \mathbf{ok}, q^S_\ell, \ell\}$	$\{\neg q^S_\ell\}$
$tr(q^S_F)$	$\{\mathbf{sync}, \mathbf{ok}, q^S_F\}$	$\{\neg q^S_F, \neg\mathbf{ok}\}$
$tr(q^S_{\alpha\wedge\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\alpha\wedge\beta}\}$	$\{q^S_\alpha, q^S_\beta, \neg q^S_{\alpha\wedge\beta}\}$
$tr_1(q^S_{\alpha\vee\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\alpha\vee\beta}\}$	$\{q^S_\alpha, \neg q^S_{\alpha\vee\beta}\}$
$tr_2(q^S_{\alpha\vee\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\alpha\vee\beta}\}$	$\{q^S_\beta, \neg q^S_{\alpha\vee\beta}\}$
$tr(q^S_{\langle\alpha?\rangle\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\langle\alpha?\rangle\beta}\}$	$\{q^S_\alpha, q^S_\beta, \neg q^S_{\langle\alpha?\rangle\beta}\}$
$tr_1(q^S_{\langle\alpha_1+\alpha_2\rangle\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\langle\alpha_1+\alpha_2\rangle\beta}\}$	$\{q^S_{\langle\alpha_1\rangle\beta}, \neg q^S_{\langle\alpha_1+\alpha_2\rangle\beta}\}$
$tr_2(q^S_{\langle\alpha_1+\alpha_2\rangle\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\langle\alpha_1+\alpha_2\rangle\beta}\}$	$\{q^S_{\langle\alpha_2\rangle\beta} \neg q^S_{\langle\alpha_1+\alpha_2\rangle\beta}\}$
$tr(q^S_{\langle\alpha_1;\alpha_2\rangle\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\langle\alpha_1;\alpha_2\rangle\beta}\}$	$\{q^S_{\langle\alpha_1\rangle\langle\alpha_2\rangle\beta} \neg q^S_{\langle\alpha_1;\alpha_2\rangle\beta}\}$
α is "test-only":		
$tr(q^S_{\langle\alpha^*\rangle\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\langle\alpha^*\rangle\beta}\}$	$\{q^S_\beta, \neg q^S_{\langle\alpha^*\rangle\beta}\}$
α isn't "test-only":		
$tr_1(q^S_{\langle\alpha^*\rangle\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\langle\alpha^*\rangle\beta}\}$	$\{q^S_\beta, \neg q^S_{\langle\alpha^*\rangle\beta}\}$
$tr_2(q^S_{\langle\alpha^*\rangle\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\langle\alpha^*\rangle\beta}\}$	$\{q^S_{\langle\alpha\rangle\langle\alpha^*\rangle\beta}, \neg q^S_{\langle\alpha^*\rangle\beta}\}$
$tr(q^S_{\langle\alpha\rangle\beta})$	$\{\mathbf{sync}, \mathbf{ok}, \alpha, q^S_{\langle\alpha\rangle\beta}\}$	$\{q_\beta, \neg q^S_{\langle\alpha\rangle\beta}\}$
$tr(q^S_{\langle\alpha\rangle\beta})$	$\{\mathbf{sync}, \mathbf{ok}, \neg\alpha, q^S_{\langle\alpha\rangle\beta}\}$	$\{\neg q^S_{\langle\alpha\rangle\beta}, \neg\mathbf{ok}\}$
$tr_1(q^S_{[\alpha?]\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{[\alpha?]\beta}\}$	$\{q^S_{nnf(\neg\alpha)}, \neg q^S_{[\alpha?]\beta}\}$
$tr_2(q^S_{[\alpha?]\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{[\alpha?]\beta}\}$	$\{q^S_\beta, \neg q^S_{[\alpha?]\beta}\}$
$tr(q^S_{[\alpha_1+\alpha_2]\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{[\alpha_1+\alpha_2]\beta}\}$	$\{q^S_{[\alpha_1]\beta}, q^S_{[\alpha_2]\beta}, \neg q^S_{[\alpha_1+\alpha_2]\beta}\}$
$tr(q^S_{[\alpha_1;\alpha_2]\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{[\alpha_1;\alpha_2]\beta}\}$	$\{q^S_{[\alpha_1][\alpha_2]\beta}, \neg q^S_{[\alpha_1;\alpha_2]\beta}\}$
α is "test-only":		
$tr(q^S_{[\alpha^*]\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{[\alpha^*]\beta}\}$	$\{q^S_\beta, \neg q^S_{[\alpha_1;\alpha_2]\beta}\}$
α isn't "test-only":		
$tr(q^S_{[\alpha^*]\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{[\alpha^*]\beta}\}$	$\{q^S_\beta, q^S_{[\alpha][\alpha^*]\beta}, \neg q^S_{[\alpha_1;\alpha_2]\beta}\}$
$tr_1(q^S_{[\alpha]\beta})$	$\{\mathbf{sync}, \mathbf{ok}, \alpha, q^S_{[\alpha]\beta}\}$	$\{q_\beta, \neg q^S_{[\alpha]\beta}\}$
$tr_2(q^S_{[\alpha]\beta})$	$\{\mathbf{sync}, \mathbf{ok}, \alpha, q^S_{[\alpha]\beta}\}$	$\{q_F, \neg q^S_{[\alpha]\beta}\}$
$tr(q^S_{\mathbf{occ}(a)})$	$\{\mathbf{sync}, \mathbf{ok}, q^S_{\mathbf{occ}(a)}\}$	$\{q_{occ(a)}, \neg q^S_{\mathbf{occ}(a)}\}$

Table 1: The synchronization actions generated for the translation of an LTL-RE goal φ in NNF. ℓ is assumed to be a literal, and a (used in the last line) is assumed to be a (ground) world action. The transition $tr(q^S_{\langle\alpha\rangle\beta})$ applies in the case where α is propositional (otherwise one of the earlier rules would be used). Similarly for $tr(q^S_{[\alpha]\beta})$. We say that α is test-only if it is a finite regular expression whose atoms are only tests $\psi?$.

and it's not the case that any of the other actions mentioned in the temporal goal φ (the set Occ) are occuring now.

Synchronization Mode Operators The set of synchronizing mode operators, O_s, contains the actions *copy*, $world$, and all actions defined in Table 1. Collectively these actions realize the bookkeeping associated with the transitioning of the AA A_φ as a result of the actions executed in so-called world mode.

Synchronization mode is divided into three consecutive parts. In the first part, we execute the *copy* action which in the successor states adds a copy q^S for each fluent q that is currently true, deleting q. Intuitively, during synchronization, each q^S defines the state of the automaton prior to synchronization. In addition, *copy* removes any propositions of the form $occ(a)$. The precondition of *copy* is $\{\mathbf{copy}, \mathbf{ok}\}$, while its effect is defined by:

$$\begin{aligned} \mathit{eff}(\mathit{copy}) = &\{q \to q^S, q \to \neg q \mid q \in Q\} \cup \\ &\{\mathbf{sync}, \neg\mathbf{copy}\} \cup \overline{Occ} \end{aligned}$$

As soon as the **sync** fluent becomes true, the second phase of synchronization begins. Here the only executable actions are those that update the state of the automaton, which are defined in Table 1. Note that one of the actions deletes the **ok** fluent. This can happen, for example while synchronizing a formula that actually expresses the fact that the action sequence has to conclude now.

When no more synchronization actions are possible—i.e., when there are no fluents of the form q^S—, we enter the third phase of synchronization. Here only action $world$ is executable; its only objective is to reestablish world mode. The precondition of $world$ is $\{\mathbf{sync}, \mathbf{ok}\} \cup \overline{Q^S}$, and its effect is $\{\mathbf{world}, \neg\mathbf{sync}\}$.

New Initial State The initial state of the original problem P intuitively needs to be "processed" by A_φ before starting to plan. Therefore, we define I' as $I \cup \{q_\varphi, \mathbf{copy}, \mathbf{ok}\}$.

New Goal Finally, the goal of the problem is to reach a state in which no state fluent in Q is true, except for q_f, which may be true. Therefore we define $G' = \{\mathbf{world}, \mathbf{ok}\} \cup \overline{Q}$.

5 Experimental Results

We have implemented the translator for LTL-RE. Three important questions to assess in an experimental evaluation are: 1) how large are the automata resulting from translation of the LTL-RE formulae, 2) how fast and space efficient is the translation, and 3) how effectively do the automata help guide search for a satisfying plan. Some of these are best evaluated on realistic benchmarks but such benchmarks exist only in limited ways and only for the LTL fragment of LTL-RE. As such, the comparative experimental analysis reported here is solely for the LTL fragment of LTL-RE.

For the purpose of experimentally evaluating our approach, we compare the performance of our LTL-RE translator both to an NFA-based LTL translator initially introduced in (Baier and McIlraith 2006), and to an AA-based LTL translator (Torres and Baier 2015), similar to ours. The NFA-based LTL translator is highly optimized to avoid, in most cases, the exponential blow up in the size of the automata characteristic of NFA-based representations of LTL. The AA translator exists in several versions including a naive version without engineering optimizations, and an optimized version. In order to fairly compare against our LTL-RE translator which currently lacks the implementation of analogous optimizations, we compared against the similarly unoptimized AA-based LTL translator. (The work reported here remains in progress, and optimization of our translator, analogous to those used in the NFA and AA-based LTL translators, constitutes ongoing work.) Even this comparison is not entirely appropriate. The LTL-RE translator is designed to translate all of $\mathsf{LDL_f}$ including regular expressions, LTL, and additional programming language constructors. One might expect that a special-purpose translator that

	NFA translator				AA-LTL					LTL-RE				
	TT	PL	PT	PS	TT	PL	WPL	PT	PS	TT	PL	WPL	PT	PS
p01	0.051	2	0.00	3	0.108	15	2	0.00	73	0.112	15	2	0.00	48
p02	0.044	3	0.00	4	0.093	22	3	0.00	139	0.110	22	3	0.00	96
p03	0.051	7	0.00	16	0.113	50	7	0.00	719	0.113	53	7	0.00	547
p04	0.058	10	0.00	27	0.112	75	10	0.01	3351	0.115	83	10	0.01	2959
p05	0.049	14	0.00	43	0.115	104	13	0.03	15575	0.139	121	13	0.04	16672
p06	0.303	14	0.00	43	0.117	99	13	0.04	16213	0.135	110	13	0.04	17153
p07	0.077	4	0.00	6	0.095	32	4	0.00	1555	0.115	32	4	0.00	1454
p08	3.568	7	0.00	11	0.116	55	6	0.04	20920	0.125	62	6	0.09	33262
p09	72.556	9	0.02	20	0.113	67	7	0.18	74360	0.133	78	7	0.57	156892
p10	72.614	9	0.02	20	0.119	68	7	0.22	89464	0.144	79	7	1.01	227686

Table 2: Results for domain *Blocksworld*, depicting translation time (TT), plan length (PL), total planning time (PT), number of planning states that were evaluated before the goal was reached (PS), and world plan length (WPL), which denotes the number of *planning* actions in PL; not including the actions that are responsible for the automaton synchronization.

	NFA translator				AA-LTL					LTL-RE				
	TT	PL	PT	PS	TT	PL	WPL	PT	PS	TT	PL	WPL	PT	PS
p01	0.057	7	0.00	10	0.109	41	7	0.04	14217	0.107	39	7	0.56	126511
p02	0.055	10	0.00	17	0.112	71	10	0.39	147017	0.115	81	10	18.35	1460496
p03	0.061	21	0.00	64	0.114	127	21	8.82	1010542	0.107	0	0	NR	NR
p04	0.060	27	0.02	121	0.112	0	0	NR	NR	0.123	0	0	NR	NR
p05	0.056	0	NR	NR	0.116	65	10	0.14	60253	0.124	71	10	3.69	574535
p06	0.062	14	0.00	35	0.114	78	13	0.52	148994	0.136	78	13	35.10	1642692
p07	0.058	21	0.00	61	0.115	113	21	0.83	221874	0.118	42	0	23.31	1006596
p08	0.045	20	0.00	70	0.085	111	20	0.98	226412	0.092	40	7	23.36	1006596
p09	0.058	10	0.00	14	0.118	73	10	10.92	1097329	0.089	106	10	11.57	1045964

Table 3: Results for domain *Logistics*, depicting translation time (TT), plan length (PL), total planning time (PT), number of planning states that were evaluated before the goal was reached (PS), and world plan length (WPL), which denotes the number of *planning* actions in PL; not including the actions that are responsible for the automaton synchronization.

	NFA translator				AA-LTL					LTL-RE				
	TT	PL	PT	PS	TT	PL	WPL	PT	PS	TT	PL	WPL	PT	PS
p01	0.057	5	0.00	7	0.107	31	5	0.01	5237	0.104	29	5	0.02	4455
p02	0.051	11	0.00	16	0.106	53	9	0.16	57597	0.109	52	9	0.21	61045
p03	0.050	15	0.00	28	0.110	74	13	2.65	436249	0.110	74	13	2.65	436249
p04	0.059	19	0.00	46	0.111	99	17	6.24	770784	0.128	96	17	20.71	1429295
p05	0.059	7	0.00	9	0.116	52	7	0.22	76082	0.118	53	7	0.22	54321
p06	0.051	12	0.00	20	0.123	76	11	2.89	485525	0.126	81	11	2.56	433910
p07	0.056	0	NR	NR	0.117	0	0	NR	NR	0.127	0	0	NR	NR
p08	0.053	5	0.00	9	0.097	39	5	0.01	4015	0.112	59	5	0.01	4702
p09	0.058	8	0.00	14	0.114	68	8	0.05	20804	0.139	120	8	0.21	45778
p10	0.065	0	NR	NR	0.120	0	0	NR	NR	0.138	0	0	NR	NR
p11	0.058	9	0.00	19	0.119	62	9	0.16	52978	0.125	75	9	0.12	27260
p12	0.060	11	0.00	22	0.116	89	11	5.84	747680	0.122	104	11	1.84	263967
p13	0.063	13	0.00	42	0.111	101	13	0.93	227407	0.132	149	13	2.77	334909
p14	0.063	15	0.00	46	0.121	0	0	NR	NR	0.139	0	0	NR	NR

Table 4: Results for domain *ZenoTravel*, depicting translation time (TT), plan length (PL), total planning time (PT), number of planning states that were evaluated before the goal was reached (PS), and world plan length (WPL), which denotes the number of *planning* actions in PL; not including the actions that are responsible for the automaton synchronization.

is tuned only to LTL would be more efficient than the LTL portion of a more general LTL-RE translator – at least before implementation of optimizations.

The experiments were performed on three domains from the International Planning Competition (IPC): *Blocksworld*, *Logistics* and *ZenoTravel*. Goals arose from those introduced in IPC 2002. All experiments were run on a 64-bit machine, with a CPU of 1600 MHz. Each experiment was limited to 15 minutes runtime and 1GB of memory. The results are illustrated in Tables 2, 3, and 4.

Tables 2, 3, and 4 illustrate the performance of three reformulations: the NFA-based one in the first column, the AA-based LTL reformulation in the second and our AA-based LTL-RE reformulation in the third. The headers of these tables include translation time (TT), plan length (PL), total planning time (PT), number of planning states that were evaluated before the goal was reached (PS), and finally, world plan length (WPL), which denotes the number of *planning* actions in PL; not including the actions that are responsible for the automaton synchronization.

A merit of an AA-based translation approach relative to an NFA-based approach is the avoidance of the exponential blowup in size that theoretically exists with the latter. Indeed, this was an original impetus for selection of an AA-based approach. Nevertheless, the NFA-based translator is so optimized that it did not exhibit this theoretical blow-up when originally developed and analyzed experimentally (Baier and McIlraith 2006). Consistent with this, we observe that the optimized NFA-based translator outperforms the two AA-based translators in many cases. However, there are problem instances that the NFA translator does less well on, marked by the drastic increase in total NFA translation time seen for example in *p07 - p10* of *Blocksworld*. These instances correspond to goal formulas of the form $\Diamond(\Diamond p_1 \wedge \Diamond p_2 \wedge ... \wedge \Diamond p_n)$, where $n = 3, 5, 6, 7$ for the instances *p07 - p10* respectively.

The comparison of the two AA-based translators is particularly interesting. The two methods generate plans of the same world plan length for all instances. The computational cost that is associated with the use of the more general framework of LTL-RE, however, is reflected in the number of states expanded before the goal is reached, as well as in the total planning time, and in the total plan length (which includes both "world" actions, and actions to synchronize the automaton).

The difference in the total length of the translations is evident in every domain, reflecting that the syntactic rewriting of LTL formulas using LDL syntax is not always the most compact way of representing them. The difference in the number of expanded states and the planning time, on the other hand, is exhibited most dramatically in the *Logistics* domain, shown in Table 3. Instance *p06*, in particular, exemplifies the gap in the planning times of the two last translators, while almost all instances of this domain showcase the difference in the number of states that these two translators generate, in favor of the AA-based LTL translator.

There are, however, instances in which our method outperforms the AA LTL translator. Some examples of this can be found in the *ZenoTravel* domain, in Table 4. In the case of *p12* for example, our translator takes significantly less time to generate the plan, and expands significantly less states while doing so.

We plan to further investigate the relationship between these two reformulations, and run more experiments to shed light on which formulas are better suited for each one. We also plan to implement an optimized version of the translator, in order to be able to fairly compare with more efficient reformulations of the AA-based LTL translation.

Our experiments did not examine the effectiveness of the translation of regular expressions and programming constructs. We note that Baier, Fritz, and McIlraith (2007; 2008) developed an automata-based translator for a Golog-like language that included regular expressions. Comparison with this translator would be interesting if suitable benchmarks could be found or constructed.

6 Discussion and Concluding Remarks

In this paper, we introduce LTL-RE: a high-level language for goal specification, which is rich enough to capture linear temporal formulas, as well as regular expressions. LTL-RE offers a convenient set of syntactic constructors, thus serving as a compelling vehicle for goal specification. A further contribution of our work is the implementation of a translation of LTL-RE goals into classical planning domains, making it feasible to plan for LTL-RE goals using state-of-the-art classical planners. We experimentally evaluate our approach by comparing the performance of this translator with an NFA-based translator and another AA-based translator, both specific to LTL formulas.

We are currently still running experiments, aiming to enhance our understanding of the differences in these reformulations. We also plan to experiment with Golog domains, to examine our translator's performance on goals which are equivalent to regular expressions. Finally, we are interested in equipping our implementation with optimizations similar to those in (Torres and Baier 2015).

Acknowledgements

We gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

Albarghouthi, A.; Baier, J.; and McIlraith, S. A. 2009. On the use of planning technology for verification. In *Proceedings of the ICAPS09 Workshop on Heuristics for Domain Independent Planning*.

Bacchus, F., and Kabanza, F. 1998. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence* 22(1-2):5–27.

Baier, J., and McIlraith, S. 2006. Planning with first-order temporally extended goals using heuristic search. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI06)*, 788–795.

Baier, J. A.; Fritz, C.; and McIlraith, S. A. 2007. Exploiting procedural domain control knowledge in state-of-the-art planners. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS)*, 26–33.

Benton, J.; Kambhampati, S.; and Do, M. B. 2006. YochanPS: PDDL3 simple preferences and partial satisfaction planning. In *5th International Planning Competition Booklet (IPC-2006)*, 54–57.

Coles, A., and Coles, A. 2011. LPRPG-P: relaxed plan heuristics for planning with preferences. In *Proceedings of the 21st International Conference on Automated Planning and Sched. (ICAPS)*.

Cresswell, S., and Coddington, A. M. 2004. Compilation of LTL goal formulas into PDDL. In de Mántaras, R. L., and Saitta, L., eds., *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)*, 985–986. Valencia, Spain: IOS Press.

De Giacomo, G., and Vardi, M. Y. 2013. Linear temporal logic and linear dynamic logic on finite traces. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*.

De Giacomo, G.; Masellis, R. D.; Grasso, M.; Maggi, F. M.; and Montali, M. 2014. Monitoring business metaconstraints based on LTL and LDL for finite traces. In *Proceedings of the 12th International Conference on Business Process Management BPM*, 1–17.

Doherty, P., and Kvarnström, J. 2001. Talplanner: A temporal logic-based planner. *AI Magazine* 22(3):95–102.

Edelkamp, S. 2006. Optimal symbolic PDDL3 planning with MIPS-BDD. In *5th International Planning Competition Booklet (IPC-2006)*, 31–33.

Eisner, C., and Fisman, D. 2007. *A practical introduction to PSL.* Springer Science & Business Media.

Erol, K.; Hendler, J.; and Nau, D. 1994. HTN planning: Complexity and expressivity. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI)*, volume 2, 1123–1128.

Fischer, M. J., and Ladner, R. E. 1979. Propositional dynamic logic of regular programs. *Journal of computer and system sciences* 18(2):194–211.

Fritz, C.; Baier, J. A.; and McIlraith, S. A. 2008. ConGolog, sin Trans: Compiling ConGolog into basic action theories for planning and beyond. In *Proceedings of the 11th International Conference on Knowledge Representation and Reasoning (KR)*, 600–610.

Gerevini, A.; Haslum, P.; Long, D.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence* 173(5-6):619–668.

Grastien, A.; Anbulagan; Rintanen, J.; and Kelareva, E. 2007. Diagnosis of discrete-event systems using satisfiability algorithms. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI)*, 305–310.

Haslum, P. 2012. Narrative planning: Compilations to classical planning. *Journal of Artificial Intelligence Research* 44:383–395.

Lago, U. D.; Pistore, M.; and Traverso, P. 2002. Planning with a language for extended goals. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI)*, 447–454.

McDermott, D. V. 1998. PDDL — The Planning Domain Definition Language. Technical Report TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

Patrizi, F.; Lipovetzky, N.; De Giacomo, G.; and Geffner, H. 2011. Computing infinite plans for LTL goals using a classical planner. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, 2003–2008.

Pnueli, A. 1977. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science (FOCS)*, 46–57.

Razavi, N.; Farzan, A.; and McIlraith, S. A. 2014. Generating effective tests for concurrent programs via AI automated planning techniques. *International Journal on Software Tools for Technology Transfer (STTT) (STTT)* 16(1):49–65.

Rintanen, J. 2000. Incorporation of temporal logic control into plan operators. In Horn, W., ed., *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI)*, 526–530. Berlin, Germany: IOS Press.

Shaparau, D.; Pistore, M.; and Traverso, P. 2008. Fusing procedural and declarative planning goals for nondeterministic domains. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI)*, 983–990.

Sohrabi, S.; Baier, J.; and McIlraith, S. 2010. Diagnosis as planning revisited. In *Proceedings of the 12th International Conference on Knowledge Representation and Reasoning (KR)*, 26–36.

Torres, J., and Baier, J. A. 2015. Polynomial-time reformulations of ltl temporally extended goals into final-state goals. In *Proceedings of the Workshop on Model-Checking and Automated Planning (MOCHAP) at ICAPS-2015*.

Uras, T., and Erdem, E. 2010. Genome rearrangement: A planning approach. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*.

Vardi, M. Y. 2012. The rise and fall of temporal logic. Keynote, 13th International Conference on Principles of Knowledge Representation and Reasoning.

Commutativity Based Search

Doron A. Peled
Department of Computer Science
Bar Ilan University

Abstract

The problem of state space search is fundamental to both concurrent system verification and planning. Often, the state space is huge, so optimizing the search may be crucial. We consider the problem of visiting all states in a graph where edges are generated by actions and the (reachable) states are not known in advance. Some of the actions may commute, i.e., they result in the same state for every order in which they are taken. We show different methods in which we use commutativity to achieve full coverage of the states while traversing considerably fewer edges.

On Combining Symmetry with Partial Order Reduction

Dragan Bošnački
Eindhoven University of Technology

We present some recent results (Bosnacki and Scheffers 2015) that combine two of the most successful state space reduction techiques in explicite state model checking: symmetry reduction and partial order reduction.

Partial order reduction (Godefroid 1996; Valmari 1996; Peled 1994) exploits the independence of the checked property from the execution order of the system actions. More specifically, two actions a, b are allowed to be permuted precisely when, if for all sequences v, w of actions: if $vabw$ (where juxtaposition denotes concatenation) is an accepted behavior, then $vbaw$ is an accepted behavior as well. In a sense, instead of checking all the execution sequences, the desired property is checked only on representative sequences, which results in significant savings in space and time. Thus, the corner stone of the independence relation is the confluence condition as given in Fig. 1a. The confluence requires that from each state s of the state space the permutations of two independent actions a and b will lead to the same state s'. The actual reduction of the state space is realized during the state space exploration by limiting the search from a given state s to only a subset of the actions that are executable in s.

Symmetry reduction (Ip and Dill 1996; Emerson and Sistla 1996; Clarke et al. 1996) is one of the most successful techniques to tackle the state space explosion problem in model checking. The technique exploits the inherent symmetry of the model which is present in many systems, like mutual exclusion algorithms, cache coherence protocols, bus communication protocols, etc. After observing that the symmetry in the description of the model results in a symmetric state space, the key idea is to partition the state space into equivalence classes of (symmetric) states. Then, the state space exploration can be performed in the usually smaller quotient state space that consists only of (representatives of the) equivalence classes.

The problem of finding canonical, i.e., unique, representatives of equivalence classes is also known as the *orbit problem*. The orbit problem is equivalent to the graph isomorphism problem (Clarke et al. 1996), for which no polynomial algorithm is known. As a result, often with symmetry reduction the verification time can become critical. On the other hand, finding multiple (non-canonical) representatives usually boils down to sorting algorithms (Emerson and Sistla 1996; Bosnacki, Dams, and Holenderski 2000). An obvious drawback of the multiple representatives is that they provide less state space reduction compared to the canonical representatives. However, in practice it often turns out that, with an acceptable increase of the state space, the verification time can be improved significantly by using multiple representatives (Ip and Dill 1996; Bosnacki, Dams, and Holenderski 2000).

It turns out that symmetry and partial order reduction are orthogonal in the sense that they exploit different aspects of the system for reduction of the state space.

A combination of symmetry based on canonical representatives with partial order reduction was presented in (Emerson, Jha, and Peled 1997). This paper can be seen as a follow-up of (Emerson, Jha, and Peled 1997) which brings as an extra contribution the symmetry case with multiple representatives. Along the lines of (Emerson, Jha, and Peled 1997) we derive our result in the more general setting of bisimulation preserving reductions. As symmetry reduction is then considered a special case of a bisimulation preserving reduction, all results are valid for symmetry too.

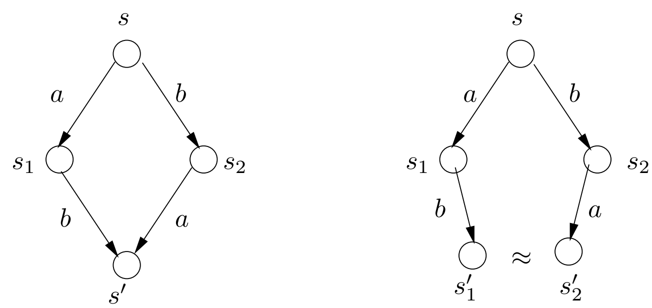
The main contribution of this paper is the use of a new kind of independence, which is weaker than the standard one. As mentioned above, in the usual definition of independence we insist on confluence, i.e., we require that the two paths obtained by permuting the independent actions a and b meet in the same state s'. Instead, in the new definition we relax the confluence condition by allowing the permutations to lead to *bisimilar states* s'_1 and s'_2, as represented in Fig. 1b.

It turns out that almost all property preservation results, like absence of deadlock, safety, and liveness (LTL and CTL^* without the next operator), that can be found in the literature can be reused with a straightforward adaptation.

One can combine symmetry and partial order also in the model checking of timed systems which use discrete time (Bosnacki 2002).

References

Bosnacki, D., and Scheffers, M. 2015. Partial order reduction and symmetry with multiple representatives. In *7th NASA Formal Methods Symposium, 27–29 April, Pasadena, California, Proceedings, (to appear)*, ?–?



(a) (b)

Figure 1: Confluence of independent actions.

Bosnacki, D.; Dams, D.; and Holenderski, L. 2000. Symmetric spin. In Havelund, K.; Penix, J.; and Visser, W., eds., *SPIN Model Checking and Software Verification, 7th International SPIN Workshop, Stanford, CA, USA, August 30 - September 1, 2000, Proceedings*, volume 1885 of *Lecture Notes in Computer Science*, 1–19. Springer.

Bosnacki, D. 2002. Partial order and symmetry reductions for discrete time. In *Workshop on Real-Time Tools, RT-TOOLS 2002, Proceedings*, ?–?

Clarke, E. M.; Jha, S.; Enders, R.; and Filkorn, T. 1996. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design* 9(1/2):77–104.

Emerson, E. A., and Sistla, A. P. 1996. Symmetry and model checking. *Formal Methods in System Design* 9(1/2):105–131.

Emerson, E. A.; Jha, S.; and Peled, D. 1997. Combining partial order and symmetry reductions. In Brinksma, E., ed., *Tools and Algorithms for Construction and Analysis of Systems, Third International Workshop, TACAS '97, Enschede, The Netherlands, April 2-4, 1997, Proceedings*, volume 1217 of *Lecture Notes in Computer Science*, 19–34. Springer.

Godefroid, P. 1996. *Partial-Order Methods for the Verification of Concurrent Systems - An Approach to the State-Explosion Problem*, volume 1032 of *Lecture Notes in Computer Science*. Springer.

Ip, C. N., and Dill, D. L. 1996. Better verification through symmetry. *Formal Methods in System Design* 9(1/2):41–75.

Peled, D. 1994. Combining partial order reductions with on-the-fly model-checking. In Dill, D. L., ed., *Computer Aided Verification, 6th International Conference, CAV '94, Stanford, California, USA, June 21-23, 1994, Proceedings*, volume 818 of *Lecture Notes in Computer Science*, 377–390. Springer.

Valmari, A. 1996. The state explosion problem. In Reisig, W., and Rozenberg, G., eds., *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996*, volume 1491 of *Lecture Notes in Computer Science*, 429–528. Springer.

UPMurphi Released: PDDL+ Planning for Hybrid Systems

Giuseppe Della Penna
University of L'Aquila
Italy
giuseppe.dellapenna@univaq.it

Benedetto Intrigila
University of Roma "Tor Vergata"
Italy
intrigil@mat.uniroma2.it

Daniele Magazzeni
King's College London
United Kingdom
daniele.magazzeni@kcl.ac.uk

Fabio Mercorio
University of Milan-Bicocca
Italy
fabio.mercorio@unimib.it

Abstract

In this tool paper, we present the release of UPMurphi, a universal planner for PDDL+ domains. Planning for hybrid domains has found increasing attention in the planning community, motivated by the need to address more realistic scenarios. While a number of techniques for planning with a subset of PDDL+ domains have been proposed, UPMurphi is able to handle the full range of PDDL+ features, including nonlinear continuous processes, exogenous events, Timed Initial Literals and numeric Timed Initial Fluents.

This paper describes the UPMurphi framework and presents its main features, together with a guide for using the tool, and some examples where UPMurphi has been successfully applied.

1 Introduction

A hybrid system is one in which there are both continuous control parameters and discrete logical modes of operation. It represents a powerful model to describe the dynamic behaviour of modern engineering artefacts. Hybrid systems frequently occur in practice, e.g., in robotics or embedded systems. Dealing with hybrid systems is becoming more and more an important challenge, as many real-world scenarios feature a mixture of discrete and continuous behaviours. Some example applications include coordination of activities of a planetary lander, oil refinery management, autonomous vehicles. Such scenarios motivate the need to reason with mixed discrete-continuous domains.

Planning for hybrid domains has found increasing attention in the planning community, motivated by the need to address more realistic scenarios and to interact with robotics and control frameworks. PDDL+ (Fox and Long 2006) is the extension of PDDL which allows the modelling of hybrid domains through the use of discrete actions, processes (that model continuous change over time), and exogenous events (that model changes that are initiated by the environment).

A number of techniques for PDDL+ planning have been proposed (Penberthy and Weld 1994; McDermott 2003; Li and Williams 2008; Coles et al. 2012; Shin and Davis 2005; Coles and Coles 2014; Molineaux, Klenk, and Aha 2010; Bryce and Gao 2015; Bogomolov et al. 2014; 2015). However, despite the recent efforts in proposing new algorithms and approaches for this domain, UPMurphi is currently the only available tool able to handle the full range of PDDL+ features.

The purpose of this paper is to accompany the release of the UPMurphi tool. To this aim, in the rest of the paper we overview the main features of the planner and we then describe the main techniques used for dealing with hybrid domains. In Section 3 we describe the general framework and provide a guide for using the tool. In Section 4 we survey some applications for which UPMurphi has been successfully used. Section 5 concludes the paper.

What UPMurphi Can Do. UPMurphi is a forward-search planner for PDDL+ domains. It can handle the whole PDDL+ language, including nonlinear continuous processes, exogenous events, Timed Initial Literals and numeric Timed Initial Fluents. It can be used either to find a single plan from the initial state to a goal state or to find a universal plan, i.e., a policy for handling the state space generated from the initial state.

2 How UPMurphi Works

UPMurphi is based on the planning-as-model-checking paradigm (Cimatti et al. 1997), and it is built on top of the CMurphi model checker (Cached Murphi Web Page 2006).

Planning in hybrid domains is challenging because in addition to the discrete state explosion problem, the continuous behaviour causes the reachability problem generally even to be undecidable. UPMurphi handles the hybrid dynamics through discretisation of time and continuous variables and by planning within a finite horizon. In this way, the state space is finite.

UPMurphi implements the *Discretise and Validate* approach (Della Penna et al. 2009) which is sketched in Figure 1. Here, the continuous dynamics of the system is relaxed into a discretised model, where discrete time steps and corresponding step functions for continuous values are used in place of the original continuous dynamics. Then, UPMurphi performs a forward reachability analysis in the discretised state space, searching for a path from the initial state to a state satisfying the goal condition. The discrete solution is then validated against the continuous model through the plan validator VAL (Howey, Long, and Fox 2004) to check whether the solution is valid or not. If it is invalid, the discretisation is refined and the process iterates. If UPMurphi fails to find a plan at one discretisation the process can be iterated at a finer grained discretisation. The validation output can guide the user in identifying a suitable finer discretisation.

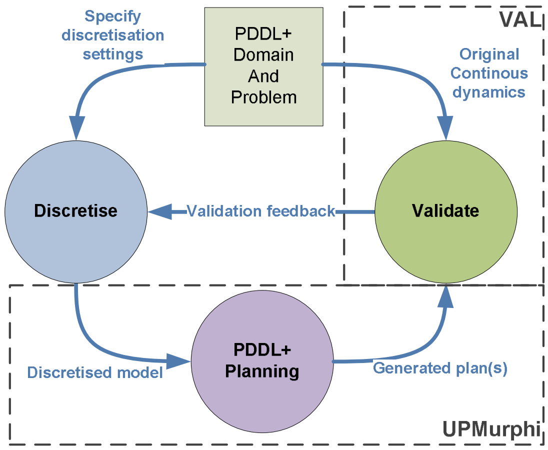


Figure 1: Graphical representation of the D&V approach

3 How to Use UPMurphi

The overall UPMurphi architecture is sketched in Figure 3. UPMurphi can be invoked through the `upmc` command by passing both the PDDL+ domain and problem as arguments. This intitiates the following chain of operations.

PDDL+ translation: the PDDL-to-UPmurphi compiler, built on top of the VAL PDDL+ parser, takes as input a PDDL+ model and outputs a semantically equivalent UPMurphi model (up to the discretisation of time and continuous variables)[1] `<domain_name>.m`. This, in turn, is compiled into an executable `<domain_name>_planner`.

PDDL+ Planning: The executable generated in the previous phase can now be invoked to start the planning/universal planning tasks. Several options can be specified to fine-tune this phase, as we describe below. The UPMurphi engine applies an explicit algorithm for building the system dynamics, and searches it through a forward search.

Plan output: Once the final plan(s) have been generated, they are written by default as PDDL+ plans, but the user can choose among a variety of different output formats (i.e., text, binary, and CSV), even in verbose mode.

Plans VALidation: UPMurphi is designed to automatically interface with the VAL plan validator, so that the generated plans are executed and validated. To enable this functionality, the user must install VAL separately.

This process is completely automatic. When running the planner, the user can specify the discretisation to be used for time and continuous variables, although the default settings can be used.

[1]The translation process is fully detailed in (Della Penna, Magazzeni, and Mercorio 2012).

3.1 UPMurphi Features

In this section we give details of the main UPMurphi features.

Disk-based Search. Starting with an initial discretisation (e.g., the one provided by default), the Discretise and Validate process should be iterated until a valid discretisation is used. However, the finer the discretisation, the larger the resulting state space, and this may lead to the state explosion phenomenon too early. To mitigate this issue, in this release UPMurphi employs the disk during the forward-search for storing both the state space generated so far and the current solution. Specifically, UPMurphi exploits the disk to store the full state description (i.e., the state values) whereas only the state signature is stored in memory using 40-bits for the encoding. This approach is beneficial for continuous domains as they often present a high number of discretised real values that grow the state size. Furthermore, it also allows trying several discretisation settings without affecting the number of states that can be visited during the search. UPMurphi is able to adapt its algorithm to increase or decrease the disk usage with respect to the user specified options and the size of the system under analysis. Furthermore, to avoid an excessive time overhead, the disk structures have been designed and implemented by taking into account their usage patterns, i.e., how (and how frequently) each structure is accessed during each phase of the planning process. This makes it possible to reduce the number of disk seek-and-read operations, which are the bottleneck of any disk algorithm, as seeks suffer from a latency time that is much higher than the actual read/write time. To give an example, UPMurphi privileges sequential read/writes, at the cost of duplicating some information and/or requiring more disk space, which is not a problem as large disks are nowadays very common.

Serialisation. Thanks to the disk-based exploration, UPMurphi can store the system graph, and access to it directly without having to load data into memory. This would allow one to *resume* the analysis by loading a (previously visited) system graph also on another machines.

State Compression. UPMurphi inherits from CMurphi a number of techniques to optimise the state representation (i.e., the *bit compression* and *hash-compaction*), and adapts them working even with the disk-based exploration described above.

Exploration Strategy. UPMurphi distinguishes between two planning modalities, namely (i) *Planning* in which a feasible plan is generated to reach a goal and (ii) *Universal Planning* that could be seen as a *collection* of plans (*aka* a set of policies) able to bring the system to the goal from *any* reachable state for which a plan exists in the given setting. Note that both these modalities support the specification of the optimality requirement for minimising the plan makespan[2].

[2]Here optimality is dependent on the discretisation and the finite horizon used for planning.

(Some) Exploration Settings. UPMurphi provides some other options that can be set for customising the state space exploration. They include the specification of the (highest) amount of memory to be used by the planning process, the enabling of a deadlock check (here intended as a non-goal state without any action applicable), and the specification of either the maximum number of BFS levels to explore or a maximum plan length.

Stepwise Exploration. UPMurphi allows the use of a step-by-step exploration useful for debugging purposes. At each step the user can specify which action has to be applied (among the ones applicable in the current state). The values of each PDDL+ predicate and fluent are shown to the user and the process iterates.

Discretisation settings. By default, UPMurphi discretises the time to 0.1 units while real scale and real fraction digits are set to 8 and 2 respectively. Although we found these values suitable for a number of planning problems, the user is free to specify different values by passing them as arguments while invoking the UPMurphi planner.

Supporting the PDDL+ semantics. UPMurphi has been designed to support the PDDL+ semantics according to the start-process-stop model introduced by (Fox and Long 2006), that works by transforming a durative-action into a chain of PDDL+ elements, namely: (i) a pair start/end actions that apply the discrete effects *at start* and *at end* of the action respectively; (ii) a process that applies the continuous change over the action execution (iii) and an event that checks whether all the *overall* durative-action conditions are satisfied during its execution. This motivated the need to properly model the processes and events interaction within UPMurphi to fully support the whole PDDL+ semantics. Figure 2 shows an example of how UPMurphi represents and reasons with the PDDL+ elements as a whole over the discretised time-line. The time is uniformly discretised in clock ticks (T) and a built-in action time-passing (TP) is responsible for advancing the time accordingly. Then, an action A_1 can activate a process $P1$ that, after three clock ticks, triggers event $E3$, which in turn activates process $P2$. UPMurphi is able to handle process/events interleaving as well as Timed Initial Literals and numeric Timed Initial Fluents. Clearly, a fine enough discretisation must be used in order to capture the happenings of TIFs and TILs. On the other hand, the time granularity of TIFs and TILs can be used as a guidance for choosing the initial time discretisation.

Limitations. The main limitation of UPMurphi is that currently there is no heuristic to guide the search, and a blind BFS is performed. UPMurphi also requires the PDDL+ domain to be typed for being processed. Finally, the only metric actually supported by UPMUrphi is `:minimize total-time`.

4 UPMurphi's Track Record

UPMurphi has been applied to several challenging PDDL+ domains. In the following we present some of them.

The Planetary Lander (Fox and Long 2006). A rover has to perform two observation tasks, that require either to perform the corresponding preparation tasks or to execute a single cumulative preparation task for both observations. The goal is to find a solution minimising the plan makespan. As a challenge, the domain presents a nonlinear system dynamics, deriving from the system equations (e.g., the energy generated by solar panels is influenced by the position of the sun), concurrence between processes (i.e., the rover may generate energy while is charging the battery), and processes/events interactions that may invalidate the plan due to tight resources and time constraints. In Figure 4 we show a log of a UPMurphi execution for the planetary lander. UPMurphi here searches for a feasible plan using up to 1Gb RAM and outputs the resulting plan in PDDL+ format. Finally, the plan is shown and saved into a file as well. More details on the use of UPMurphi in this domain can be found in (Della Penna, Magazzeni, and Mercorio 2012).

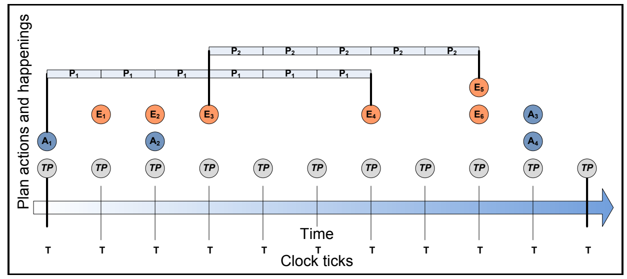


Figure 2: Processes in the discretised plan timeline

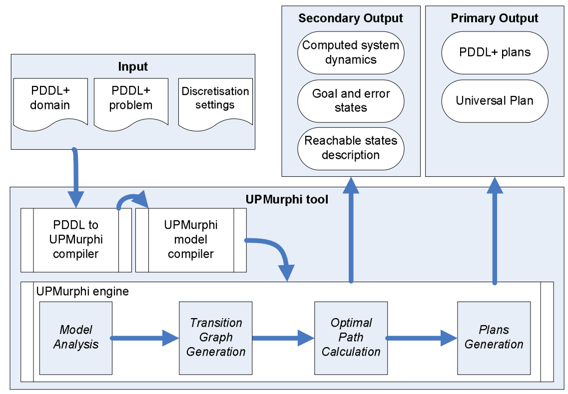


Figure 3: Overview of the UPMurphi architecture

The Batch Chemical Plant. A production system is designed to obtain a concentrated saline solution recycling the remaining part for the next cycle. The system has many tightly connected components, regulated by a nonlinear dynamics (due to the equations modelling temperature and concentration variations), unknown action durations and a large set of safety constraints. UPMurphi has been used to synthesise a set of policies for many different initial production configurations. Note that thanks to the efficient use of the disk during the state space exploration, UPMurphi was able to generate up to 7 million plans. More details on the use of UPMurphi for the batch chemical plant can be found in (Della Penna et al. 2010).

```
cmd: ./planetary_lander_planner -search:f -m1000 -format:pddl

* Source domain: planetary_lander.pddl
* Source problem: planetary_lander_problem.pddl
* Planning Mode: Feasible Plan
* Output format: PDDL+
* Epsilon separation: 0.001
* Output target: "planetary_lander_problem_plan.pddl"

* UPMurphi Model: planetary_lander
* State size 498 bits (rounded to 64 bytes).
* Allocated memory: 1000 Megabytes

**  Time Discretisation = 0.1
**  Digits for representing the integer part of a real =  8
**  Digits for representing the fractional part of a real =  2

=== Analyzing model... ===============================
* Maximum size of the state space: 64726931 states.
  with states hash-compressed to 40 bits.
[0:0:2.87] states explored: 100000, actions fired: 101880
BFS level: 63, states queued: 7368, goals found: 0, errors: 0
max plan length: 6.30
34843.21 states/sec, 35498.26 actions/sec, 0.15\% memory used
....
[0:1:30.32] states explored: 3000000, actions fired: 3194058
BFS level: 170, states queued: 7657, goals found: 0, errors: 0
max plan length: 17.00
33215.23 states/sec, 35363.80 actions/sec, 4.63\% memory used

======================================================
Model exploration complete (in 92.76 seconds).
3282884 actions fired
1 start states
3084414 reachable states
1 goals found

=== Building model dynamics... =======================
Transition Graph mode: Memory Image
Model dynamics rebuilding complete (in 97.71 seconds).
3084414 states
3282884 transitions
out degree: min 0 max 10 avg 1.06

=== Finding paths... ========================
* Search Algorithm: Feasible Plan.

=== Collecting plans... ==============================
Plan(s) generation complete (in 98.41 seconds).
1 plans
plan length (actions): min 186 max 186 avg 186.00
plan duration (time): min 0 max 180 avg 180.00
plan weight: min 0 max 180 avg 180.00

=== Writing final results... =========================
* Output format: PDDL+
* Output target: "planetary_lander_problem_plan.pddl".

; --Plan #00001--------------------------
; -- Discretisation: 0.100----------------
; --------------------------------------
0.000: ( fullprepare dum unit1) [3.000]
3.001: ( observe2 unit1) [8.000]
11.002: ( observe1 unit1) [7.000]
; --------------------------------------
; --Plan duration: 18.002, weight: 0180----
; --------------------------------------
```

Figure 4: Log of a UPMurphi exeuction for the *Planetary Lander* domain

	Planetary Lander	*Chemical Plant*
State Space Size	10^{24}	10^{29}
Reachable States	$31,965,220$	$29,968,861$
Generated Plans	$5,309,514$	$7,154,464$

Table 1: Some statistics for Planetary Lander and Chemical Plant domains

Nonlinear generator. It is the continuous model of the well-known generator domain (Howey and Long 2003). A generator is powered by a fuel tank with a limited capacity of 60 fuel units and consumes one fuel unit per second. During the generator activity (modelled by the consume durative action), two fuel tanks of 25 fuel units each can be used to refuel it (through the refuel durative action). The refuelling process has a variable duration (i.e., its duration must be decided by the planner) and is described by the Torricelli's law, which makes the system dynamics nonlinear. Moreover, the domain also involves concurrency, since the consume and refuel actions take place continuously and concurrently, and are modelled through continuous processes. The goal is to make the generator run for 100 seconds. Table 2 summarises the results of the universal planning process after three Discretise and Validate iterations. We first considered a time discretisation of 5.0 and 2.5, both resulting in invalid solutions. We then refined the discretisation to 1.0 which proved to be fine enough for obtaining valid plans.

Time discretisation (sec)	5.0	2.5	1.0
State space size	10^{15}	10^{16}	10^{18}
Reachable states	$26,276$	$399,189$	$29,119,047$
Generated plans	0	$10,015$	$126,553$
Total synthesis time	3.7	20.71	$1,430.11$
Valid	NO	NO	**YES**

Table 2: Universal Plan statistics for the generator domain with time discretisation from 5.0 down to 1.0 seconds

Other examples were UPMurphi has been applied can be found in (Della Penna, Magazzeni, and Mercorio 2012), while works built on top of UPMurphi are described in (Fox, Long, and Magazzeni 2012; Campion et al. 2013; Boselli et al. 2014) and (Mezzanzanica et al. 2015).

5 Concluding Remarks

In this paper we presented the release of the PDDL+ planner UPMurphi, overviewing its main features that allow it to handle the full range of PDDL+ features, including nonlinear continuous processes, exogenous events, Timed Initial Literals and numeric Timed Initial Fluents. On a practical note, UPMurphi has been designed to work natively on Linux distributions (Ubuntu specifically), but it has been extensively tested on Windows with Cygwin environment, too. Finally, a MacOS compilation is also supported. Please refer to the UPMurphi web page (UPMurphi Web Page 2015) for download, installation instructions, more details, and news about UPMurphi development.

References

Bogomolov, S.; Magazzeni, D.; Podelski, A.; and Wehrle, M. 2014. Planning as model checking in hybrid domains. In *Proceedings of the Twenty Eighth Conference on Artificial Intelligence (AAAI-14)*. AAAI Press.

Bogomolov, S.; Magazzeni, D.; Minopoli, S.; and Wehrle, M. 2015. PDDL+ planning with hybrid automata: Foundations of translating must behavior. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS-15)*. AAAI Press.

Boselli, R.; Cesarini, M.; Mercorio, F.; and Mezzanzanica, M. 2014. Planning meets data cleansing. In *The 24th International Conference on Automated Planning and Scheduling (ICAPS-14)*, 439–443.

Bryce, D., and Gao, S. 2015. SMT-based nonlinear PDDL+ planning. In *Proceedings of the Twenty Nineth Conference on Artificial Intelligence (AAAI-15)*. AAAI Press.

Cached Murphi Web Page. 2006. `http://www.di.univaq.it/gdellape/murphi/cmurphi.php`.

Campion, J.; Dent, C.; Fox, M.; Long, D.; and Magazzeni, D. 2013. Challenge: Modelling unit commitment as a planning problem. In *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS-13)*.

Cimatti, A.; Giunchiglia, F.; Giunchiglia, E.; and Traverso, P. 1997. Planning via model checking: A decision procedure for *AR*. In *Recent Advances in AI Planning, 4th European Conference on Planning, (ECP'97)*, 130–142.

Coles, A. J., and Coles, A. I. 2014. PDDL+ planning with events and linear processes. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS-14)*.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2012. COLIN: Planning with continuous linear numeric change. *Journal of Artificial Intelligence Research (JAIR)* 44:1–96.

Della Penna, G.; Magazzeni, D.; Mercorio, F.; and Intrigila, B. 2009. UPMurphi: A tool for universal planning on PDDL+ problems. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS-09)*. AAAI.

Della Penna, G.; Intrigila, B.; Magazzeni, D.; and Mercorio, F. 2010. A PDDL+ benchmark problem: The batch chemical plant. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS-10)*, 222–225.

Della Penna, G.; Magazzeni, D.; and Mercorio, F. 2012. A universal planning system for hybrid domains. *Applied Intelligence* 36(4):932–959.

Fox, M., and Long, D. 2006. Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research (JAIR)* 27:235–297.

Fox, M.; Long, D.; and Magazzeni, D. 2012. Plan-based policies for efficient multiple battery load management. *J. Artif. Intell. Res. (JAIR)* 44:335–382.

Howey, R., and Long, D. 2003. Vals progress: The automatic validation tool for PDDL2.1 used in the international planning competition. In *Proc. of ICAPS Workshop on the IPC*.

Howey, R.; Long, D.; and Fox, M. 2004. VAL: Automatic plan validation, continuous effects and mixed initiative planning using PDDL. In *16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, 294–301.

Li, H. X., and Williams, B. C. 2008. Generative planning for hybrid systems based on flow tubes. In *ICAPS*, 206–213.

McDermott, D. V. 2003. Reasoning about autonomous processes in an estimated-regression planner. In *ICAPS*, 143–152.

Mezzanzanica, M.; Boselli, R.; Cesarini, M.; and Mercorio, F. 2015. A model-based evaluation of data quality activities in KDD. *Inf. Process. Manage.* 51(2):144–166.

Molineaux, M.; Klenk, M.; and Aha, D. W. 2010. Planning in dynamic environments: Extending htns with nonlinear continuous effects. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI*.

Penberthy, J. S., and Weld, D. S. 1994. Temporal planning with continuous change. In *AAAI*, 1010–1015.

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

UPMurphi Web Page. 2015. `https://github.com/gdellapenna/UPMurphi`.

Hybrid Systems: Guided Search, Abstractions, and Beyond

Sergiy Bogomolov
IST Austria, Austria
sergiy.bogomolov@ist.ac.at

Abstract

Hybrid systems represent an important and powerful formalism for modeling real-world applications such as embedded systems. A verification tool like SpaceEx is based on the exploration of a symbolic search space (the region space). As a verification tool, it is typically optimized towards proving the absence of errors. In some settings, e.g., when the verification tool is employed in a feedback-directed design cycle, one would like to have the option to call a version that is optimized towards finding an error path in the region space. A recent approach in this direction is based on guided search. Guided search relies on a cost function that indicates which states are promising to be explored, and preferably explores more promising states first. In this talk, we present two approaches to define and compute efficient cost functions. We develop our approaches on the top of the symbolic hybrid model checker SpaceEx which uses regions as its basic data structures.

In the first part of the talk, we introduce a box-based distance measure which is based on the distance between regions in the concrete state space. In the second part of the talk, we discuss an abstraction-based cost function based on pattern databases for guiding the reachability analysis. For this purpose, a suitable abstraction technique that exploits the flexible granularity of modern reachability analysis algorithms is introduced. We illustrate the practical potential of our approaches in several case studies.