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 $C \rightarrow L$, where C is a set of literals over P and L is a literal over P. When $C \rightarrow L$ 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:

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

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

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

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 $\mathsf{LDL_f}$

$\mathsf{LTL_f}$: 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 $\pi, i \models \neg\Diamond\neg\varphi$

$\mathsf{LDL_f}$: $\mathsf{LDL_f}$ (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 ($\mathsf{LTL\text{-}RE}$)[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 $\mathsf{LTL\text{-}RE}$ through the use of a special predicate, "**occ**" which ranges over the ground actions in a planning problem description, $\mathcal{A}$.

$\mathsf{LTL\text{-}RE}$ 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 $\mathsf{LTL\text{-}RE}$ is given by the following grammar:

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

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."

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

3.3 Semantics

$\mathsf{LTL\text{-}RE}$ 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 $\mathsf{LTL\text{-}RE}$ 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:

- $\pi, i \models \textbf{occ}(a)$ 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. □

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 $\beta = b_0 b_1 \dots b_n$ in which $b_i = s_i \cup \{a_i\}$, for $i \in \{0, \dots, n-1\}$, 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 $\alpha = a_1 a_2 \dots a_n$ 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 $\alpha_0 a_1 \alpha_1 a_2 \alpha_2 \dots a_n \alpha_n$, 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:

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

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:

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

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 blowup 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.