X and more Parallelism
Integrating LTL-Next into SAT-based Planning with Trajectory Constraints while Allowing for even more Parallelism

Gregor Behnke and **Susanne Biundo**
Institute of Artificial Intelligence, Ulm University, D-89069 Ulm, Germany
{gregor.behnke, susanne.biundo}@uni-ulm.de

Abstract

Linear temporal logic (LTL) provides expressive means to specify temporally extended goals as well as preferences. Recent research has focussed on compilation techniques, i.e., methods to alter the domain ensuring that every solution adheres to the temporally extended goals. This requires either new actions or an construction that is exponential in the size of the formula. A translation into boolean satisfiability (SAT) on the other hand requires neither. So far only one such encoding exists, which is based on the parallel $\exists$-step encoding for classical planning. We show a connection between it and recently developed compilation techniques for LTL, which may be exploited in the future. The major drawback of the encoding is that it is limited to LTL without the X operator. We show how to integrate X and describe two new encodings, which allow for more parallelism than the original encoding. An empirical evaluation shows that the new encodings outperform the current state-of-the-art encoding.

1 Introduction

Linear temporal logic (LTL (Pnueli 1977)) is a generic and expressive way to describe (state-)trajectory constraints. It is often used to specify temporal constraints and preferences in planning, e.g., to describe safety constraints, to state necessary intermediate goals, or to specify the ways in which a goal might be achieved. Most notably, the semantics of such constraints in PDDL 3.0 (Gerevini and Long 2005) is given in terms of LTL formulae, which is the de-facto standard for specifying planning problems.

Traditionally, LTL constraints are handled by first compiling them into an equivalent Büchi Automaton, and then translating the automaton into additional preconditions and effects for actions (see e.g. Edelkamp (2003)). This compilation might be too expensive as the Büchi Automaton for a formula ϕ can have up to $2^{|\phi|}$ states. Recent work proposed another compilation using Alternating Automata (Torres and Baier 2015). These automata have only $\mathcal{O}(|\phi|)$ states allowing for a guaranteed linear compilation. There are also planners that do not compile the model, but evaluate the formula during forward search, e.g., TALplanner (Doherty and Kvarnström 2001), TLplan (Bacchus and Kabanza 2000), or the work by Hsu et al. (2007). However, heuristics have to be specifically tailored to incorporate the formula, or else the search becomes blind. TALplanner and TLplan even use the temporally extended goals for additional search guidance.

Another option is to integrate LTL into planning via propositional logic. Planning problems can be translated into (a sequence of) boolean formulae. A temporally extended goal can then be enforced by adding additional clauses to this formula. So far only one such encoding has been developed by Mattmüller and Rintanen (2007). It uses an LTL to SAT translation from the model checking community, which assumes that only a single state transition is executed at a time. The main focus of their work lies on integrating the efficient $\exists$-step encoding with this representation of LTL formulae. In the $\exists$-step encoding operators can be executed simultaneously, as long as they are all applicable in the current state, the resulting state is uniquely determined, and there is an ordering in which they are actually executable. Mattmüller and Rintanen presented alterations to the $\exists$-step formula restricting the parallelism such that LTL formulas without the next-operator are handled correctly.

We point out an interesting relationship between the LTL encoding of Mattmüller and Rintanen and the Alternating Automaton encoding by Torres and Baier, showing that both use the same encoding technique, although derived by different means. This insight might prove useful in the future, e.g., to allow for optimisation of the propositional encoding using automata concepts. Next, we show how the propositional encoding by Mattmüller and Rintanen can be extended to also be able to handle the next-operator X. We introduce a new concept – partial evaluation traces – to capture the semantics of the encoding with respect to an LTL formula and show that our extension is correct. Based on partial evaluation traces, we show that the restrictions posed by Mattmüller and Rintanen (2007) on allowed parallelism can be relaxed while preserving correctness. We provide an alteration of their encoding allowing for more parallelism. We present an alternative encoding, also based on partial evaluation traces, which allows for even more parallelism by introducing intermediate timepoints at which the formula is evaluated. Our empirical evaluation of all encodings shows that our new encodings outperform the original one.

2 Preliminaries

Planning

We consider propositional planning without negative preconditions. This is known to be equivalent to STRIPS (allowing for negative preconditions) via compilation. Also note that all our techniques are also applicable in the presence of conditional effects. We do not consider them in this paper to keep the explanation of the techniques as simple as possible. For the extension to conditional effects, see Mattmüller and Rintanen (2007).

Let A be a set of proposition symbols and $Lit(A) = \{a, \neg a \mid a \in A\}$ be the set of all literals over A. An action a is a tuple $a = \langle p, e\rangle$, where p – the preconditions – is a subset of A and e – the effects – is a subset of $Lit(A)$. We further assume that the effects are not self-contradictory, i.e., that for no $a \in A$ both a and $\neg a$ are in e. A state s is any subset of A. An action $a = \langle p, e\rangle$ is executable in s, iff $p \subseteq s$. The state resulting form executing a in s is $(s \setminus \{a \mid \neg a \in e\}) \cup \{a \mid a \in e\}$. A planning problem $\mathcal{P} = \langle A, O, s_I, g\rangle$ consists of a set of proposition symbols A, a set of operators O, the initial state s_I, and the goal $g \subseteq A$. A sequence of actions $o_1, \dots, o_n$ is a plan for $\mathcal{P}$ iff there exists a sequence of states $s_0, \dots, s_{n+1}$ such that for every $i \in \{1, \dots, n+1\}$, o_i is applicable in s_i, its application results in s_{i+1}, $s_0 = s_I$, and $g \subseteq s_{n+1}$. This sequence of states is called an execution trace.

Linear Temporal Logic

Formulae in Linear Temporal Logic (LTL) are constructed over a set of primitive propositions. In the case of planning these are the proposition symbols A. LTL formulae are recursively defined as any of the following constructs, where p is a proposition symbol and f and g are LTL formulae.

$$\bot \mid \top \mid p \mid \neg f \mid f \wedge g \mid f \vee g \mid Xf \mid \mathring{X} f \mid Ef \mid Gf \mid fUg$$

X, $\mathring{X}$, E, G, and U are called temporal operators. There are several further LTL-operators (like $\mathring{U}$, R, or S and T, see e.g. (Biere et al. 2006)). Each of them can be translated into a formula containing only the temporal operators X, $\mathring{X}$, and U. The semantics of an LTL formula ϕ is given with respect to an execution trace. In general this trace can be infinitely long, as LTL can describe repeated behaviour. We consider only LTL over finite traces, which is commonly called LTL_f (De Giacomo and Vardi 2013). The encodings we present can easily be extended to the infinite case (see Mattmüller and Rintanen 2007). The truth value of an LTL_f formula ϕ is defined over an execution trace $\sigma = (s_0, s_1, \dots, s_n)$ as $[[\phi]](\sigma)$ where

$$\begin{aligned}
[[p]](s_0, \sigma) &= p \in s_0 \qquad \text{if } p \in A\\
[[\neg f]](\sigma) &= \neg[[f]](\sigma)\\
[[f \wedge g]](\sigma) &= [[f]](\sigma) \wedge [[g]](\sigma)\\
[[f \vee g]](\sigma) &= [[f]](\sigma) \vee [[g]](\sigma)\\
[[Xf]](s_0, \sigma) = [[\mathring{X} f]](s_0, \sigma) &= [[f]](\sigma)\\
[[Xf]](s_0) &= \bot\\
[[\mathring{X} f]](s_0) &= \top\\
[[Ef]](s_0, \sigma) &= [[f]](s_0, \sigma) \vee [[Ef]](\sigma)\\
[[Gf]](s_0, \sigma) &= [[f]](s_0, \sigma) \wedge [[Gf]](\sigma)\\
[[Ef]](s_0) = [[Gf]](s_0) &= [[f]](s_0)
\end{aligned}$$

$$\begin{aligned}
[[fUg]](s_0, \sigma) = [[g]](s_0, \sigma) \vee\\
([[f]](s_0, \sigma) \wedge [[fUG]](\sigma))\\
[[fUg]](s_0) = [[g]](s_0)
\end{aligned}$$

The intuitive of the semantics of temporal operators are: Ef – eventually f, i.e., f will hold at some time, now or in the future, Gf – globally f, i.e., f will hold from now on for ever, fUg – f until g, i.e., g will eventually hold and until that time f will always hold, and Xf – next f, i.e., f holds in the next state of the trace. Since we consider the case of finite LTL, we have – in addition to standard LTL – a new operator: weak next $\mathring{X}$. The formula Xf requires that there is a next state and that f holds in that state. In contrast, $\mathring{X} f$ asserts that f holds if a next state exists; if there is none, $\mathring{X} f$ is always true, taking care of the possible end of the state sequence.

As a preprocessing step, we always transform an LTL formula ϕ into negation normal form without increasing its size, i.e., into a formula where all negations only occur directly before atomic propositions. This can be done using equivalences like $\neg Gf = E\neg f$. Next, we add for each proposition symbol $a \in A$ a new proposition symbol $\overline{a}$. Its truth value will be maintained such that it is always the inverse of a. I.e. whenever an action has $\neg a$ as its effect, we add the effect $\overline{a}$ and when it has the effect a we add $\neg\overline{a}$. Lastly, we replace $\neg a$ in ϕ with $\overline{a}$, resulting in a formula not containing negation.

Given a planning problem $\mathcal{P}$ and a LTL formula ϕ, LTL planning is the task of finding a plan π whose execution trace σ will satisfy ϕ, i.e., for which $[[\phi]](\sigma)$. For a given LTL formula ϕ we define $A(\phi)$ as the set of predicates contained in ϕ and $\mathcal{S}(\phi)$ to be the set of all its subformulae. We write $[o]_e^\phi$ for the intersections of the effects of o and $A(\phi)$, i.e. all those effects that occur in ϕ.

3 State-of-the-art LTL→SAT encoding

As far as we are aware, there is only a single encoding of LTL planning problems into boolean satisfiability, developed by Mattmüller and Rintanen (2007). They adapted a propositional encoding for LTL developed by Latvala et al. 2004 for bounded model checking. The main focus of Mattmüller and Rintanen's work lies on integrating modern, non-sequential encodings of planning problems into the formula. The encoding models evaluating the LTL formula in timesteps, which correspond to the states in a trace. In Latvala et al.'s encoding (which was not developed for planning, but for a more general automata setting) only a single action may be executed at each timestep in order to evaluate the formula correctly. Research in translating planning problems into propositional formulae has however shown that such sequential encodings perform significantly worse than those that allow for a controlled amount of parallel action execution (Rintanen, Heljanko, and Niemelä 2006). Mattmüller and Rintanen addressed the question of how to use the LTL encoding by Latvala et al. in a situation where multiple state transitions take care in parallel – as is the case in these planning encodings. They used the property of stutter-equivalence which holds for LTL_{-X} (i.e. LTL without the X and $\mathring{X}$ operators) to integrate Latvala et al.'s encoding. To exploit stutter-equivalence, they had to restrict the allowed amount of parallelism to ensure correctness.

Their encoding, which we will denote with M&R'07, is based on the $\exists$-step encoding of propositional planning by Rintanen, Heljanko, and Niemelä (2006). As such, we start by reviewing the $\exists$-step encoding in detail. In this encoding the plan is divided into a sequence of timesteps $0, \dots, n$. Each timestep t is assigned a resulting state using decision variables a^t for all $a \in A$ and $t \in \{1, \dots, n+1\}$, each indicating that the proposition symbol a holds at timestep t, i.e. after executing the actions at timestep $t-1$. The initial state is represented by the variables a^0. Actions can be executed between two neighbouring timesteps t and $t+1$, which is represented by decision variables o^t for all $o \in O$ and $t \in \{0, \dots, n\}$. If o^t is true the action o is executed at time t. The encoding by Kautz and Selman (1996) is then used to determine which actions are executable in a^t and how the state a^{t+1} resulting from their application looks like. In a sequential encoding, one asserts for each timestep t that at most one o^t atom is true. Intuitively, this is necessary to ensure that the state a^{t+1} resulting from executing the actions o^t is uniquely determined. Consider, e.g., a situation where two actions `move-a-to-b` and `move-a-to-c` are simultaneously applicable, but result in conflicting effects. Executing these two actions in parallel has no well-defined result. Interestingly, the mentioned constraint is not necessary in this case, as the encoding by Kautz and Selman already leads to an unsatisfiable formula. There are however situations, where the resulting state is well-defined, but it is not possible to execute the actions in any order. Consider two actions `buy-a` and `buy-b`, both requiring money, spending it, and achieving possession of `a` and `b`, respectively. Both actions are applicable in the same state and their parallel effects are well-defined, as they don't conflict. It is not possible to find a sequential ordering of these two actions that is executable, as both need money, which won't be present before executing the second action. This situation must be prohibited, which can easily be achieved by forbidding parallel action execution at all, as in the sequential encoding.

In the $\exists$-step encoding, executing actions in parallel is allowed. Ideally, we would like to allow any subset S of o^t to be executable in parallel, as long as there exists a linearisation of S that is executable in the state s^t represented by a^t and all executable linearisations lead to the same state s^{t+1}. This property is however practically too difficult to encode (Rintanen, Heljanko, and Niemelä 2006). Instead, the $\exists$-step encoding uses a relaxed requirement. Namely, (1) all actions in S must be executable in s^t, then it chooses a total order of all actions O, (2) asserts that if a set of actions S is executable, it must be executable in that order, and (3) that the state reached after executing them is this order is s^{t+1}. The encoding by Kautz and Selman ensures the first and last property. The $\exists$-step encoding has to ensure the second property. It however does not permit all subsets $S \subseteq O$ to be executable in parallel, but only those for which this property can be guaranteed.

As a first step, we have to find an appropriate order of actions in which as many subsets $S \subseteq O$ as possible can be executed. For this, the Disabling Graph (DG) is used. It determines which actions can be safely executed in which order without checking the truth of propositions inside the formula. In practice, one uses a relaxed version of the DG (i.e. one containing more edges), as it is easier to compute (Rintanen, Heljanko, and Niemelä 2006).

Definition 1. *Let $\mathcal{P} = \langle A, O, s_I, g\rangle$ be a planning problem. An action $o_1 = \langle p_1, e_1\rangle$* affects *an other action $o_2 = \langle p_2, e_2\rangle$ iff $\exists l \in A$ s.t. $\neg l \in e_1$ and $l \in p_2$.*

A Disabling Graph $DG(\mathcal{P})$ is a directed graph $\langle O, E\rangle$ with $E \subseteq O \times O$ that contains all edges (o_1, o_2) where o_1 affects o_2 and a state s exists that is reachable from s_I in which both o_1 and o_2 are applicable.

The DG is a domain property and is not tied to any specific timestep, as such the restrictions it poses apply to every timestep equally. The DG encodes which actions disable the execution of other actions after them in the same timestep, i.e., we ideally want the actions to be ordered in the opposite way in the total ordering chosen by the $\exists$-step encoding. If the DG is acyclic, we can execute all actions in the inverted order of the disabling graph, as none will disable an action occurring later in that order. If so, the propositional encoding does not need any further clauses, as any subset S of actions can be executed at a timestep – provided that their effects do not interfere.

The DG is in practice almost never acyclic. Problematic are only strongly connected components (SCCs) of the DG, were we cannot find an ordering s.t. we can guarantee executability for all subsets of actions. Instead we fix some order $\prec$ for each SCC, asserting that the actions in it will always be executed in that order, and ensure in the SAT formula that if two actions o_1 and o_2 with $o_1 \prec o_2$ are executed, o_1 does not affect o_2. This way, we can safely ignore *some* of the edges of the DG – as their induced constraints are satisfied by the fixed ordered $\prec$ – while others have to be ensured in the formula. I.e. if we ensure for every edge (o_1, o_2) with $o_1 \prec o_2$ that if o_1 is part of the executed subset o_2 is not, we know that there is a linearisation in which the chosen subset S is actually executable. To ensure this property, Rintanen, Heljanko, and Niemelä introduced *chains*. A chain $chain(\prec; E; R; l)$ enforces that whenever an action o in E is executed, all actions in R that occur after a in $\prec$ cannot be executed. Intuitively, E are those actions that produce some **e**ffect a, while the actions in R **r**ely on $\neg a$ to be true. The last argument l is a label that prohibits interference between multiple chains.

$$
\begin{aligned}
&chain(o_1, \dots, o_n; E; R; l) = \\
&\quad \bigwedge \{o_i \rightarrow \mathrm{d}^{j,l} \mid i < j, o_i \in E, o_j \in R, \{o_{i+1}, .., o_{j-1}\} \cap R = \emptyset\} \\
&\quad \cup \{l^i \rightarrow a^{j,l} \mid i < j, \{o^i, o^j\} \subseteq R, \{o_{i+1}, .., o_{j-1}\} \cap R = \emptyset\} \\
&\quad \cup \{l^i \rightarrow \neg o_i \mid o^i \in R\}
\end{aligned}
$$

To ensure that for any SCC S of $DG(\mathcal{P})$ the mentioned condition holds for the chosen ordering $\prec$ of S, we generate for every proposition symbol $a \in A$ a chain with

$$
\begin{aligned}
E_a &= \{o \in S \mid o = \langle p, e\rangle \text{ and } \neg a \in e\} \\
R_a &= \{o \in S \mid o = \langle p, e\rangle \text{ and } a \in p\}
\end{aligned}
$$

Based on the $\exists$-step encoding, Mattmüller and Rintanen (2007) added support for LTL formulae ϕ by exploiting the stutter-equivalence of LTL_{-X}. This stutter-equivalence ensures that if multiple actions are executed in a row but don't change the truth any of the predicates in $A(\phi)$, the truth of the formula is not affected, i.e., the truth of the formula does not depend on how many of these actions are executed in a row. Consequently the formula only needs to be checked whenever the truth of propositions in $A(\phi)$ changes. Their construction consists of two parts. First, they add clauses to the formula expressing that the LTL formula ϕ is actually satisfied by the trace restricted to the states where propositions in $A(\phi)$ change. These states are the ones represented in the $\exists$-step encoding by a^t atoms. Second, they add constraints to the $\exists$-step parallelism s.t. in every timestep the first action executed according to $\prec$ that changes proposition symbols in $A(\phi)$ is the only one to do so. Other actions in that timestep may not alter the state w.r.t to $A(\phi)$ achieved by that first action, but can assert the same effect.

In their paper, they provide a direct translation of ϕ into a proposition formula. In practice however, this formula cannot be given to a SAT solver, as it requires a formula in conjunctive normal form (CNF). The formula given in the paper is not in CNF and translating it into CNF can lead to a CNF of exponential size. They instead introduce additional variables (Mattmüller 2006), allowing them to generate (almost) a CNF[1]. For every sub-formula $\psi \in \mathcal{S}(\phi)$ and every timestep t they introduce the variable ψ^t_{LTL}, stating that ψ holds for the trace starting at timestep t. They then assert: (1) that ϕ^0_{LTL} holds and (2) that for every ψ^t_{LTL} the consequences must hold that make ψ true for the trace starting at time t. The latter is expressed by clauses $\psi^t_{LTL} \rightarrow [[\psi]]^t$, where $[[\psi]]^t$ is given in Tab. 1. Note that the M&R'07 encoding cannot handle the next operators X and $\mathring{X}$, as they are sensitive to stuttering, i.e., stutter-equivalence does not hold for formulae that contain X or $\mathring{X}$. We have added the encoding for X (Latvala et al. 2004). In addition, we have restricted the original encoding from infinite to finite LTL-traces and added a new encoding for $\mathring{X}$. Note that the M&R'07 encoding will lead to wrong results if used with the presented encoding of the X and $\mathring{X}$ operators. It is however correct, if used in conjunction with a sequential encoding (Latvala et al. 2004). We show in Sec. 5 how the M&R'07 encoding can be changed to handle X and $\mathring{X}$ correctly. Lastly, clauses need to be added in order to ensure that actions executed in parallel do not alter the truth of propositions in $A(\phi)$ – except for the first action that actually changes them. The extension of $\exists$-step encoding achieving this is conceptually simple, as it consists of only two changes to the original encoding:

1. add for every two actions o_1, o_2 which are simultaneously applicable the edge (o_1, o_2) to the DG iff $[o_2]^\phi_e \setminus [o_1]^\phi_e \neq \emptyset$, i.e., o_2 would change more than o_1 with respect to $A(\phi)$.
2. add for every literal $l \in Lit(A(\phi))$ and SCC of $DG(\mathcal{P})$ with its total order $\prec$ the chain $chain(\prec; E^\phi_l; R^\phi_l; \phi_l)$

[1]The translations of $f \wedge g$, Gf and fUg contain one conjunction each, which can be multiplied out easily.

ϕ	$t < n$	$t = n$
$[[p]]^t \quad p \in A$	p^t	p^t
$[[f \wedge g]]^t$	$f^t_{LTL} \wedge g^t_{LTL}$	$f^t_{LTL} \wedge g^t_{LTL}$
$[[f \vee g]]^t$	$f^t_{LTL} \vee g^t_{LTL}$	$f^t_{LTL} \vee g^t_{LTL}$
$[[Xf]]^t$	f^{t+1}_{LTL}	$\bot$
$[[\mathring{X}f]]^t$	f^{t+1}_{LTL}	$\top$
$[[Ef]]^t$	$f^t_{LTL} \vee (Ef)^{t+1}_{LTL}$	f^t_{LTL}
$[[Gf]]^t$	$f^t_{LTL} \wedge (Gf)^{t+1}_{LTL}$	f^t_{LTL}
$[[fUg]]^t$	$g^t_{LTL} \vee ((fUg)^{t+1}_{LTL} \wedge f^t_{LTL})$	f^t_{LTL}

Table 1: Transition rules for LTL formulae

with

(a) $E^\phi_l = \{o \in O \mid o = \langle p, e\rangle$ and $l \notin e\}$

(b) $R^\phi_l = \{o \in O \mid o = \langle p, e\rangle$ and $l \in e\}$;

Mattmüller and Rintanen (2007) have proven that these clauses suffice to ensure a correct and complete encoding.

4 Alternating Automata and M&R'07

In recent years, research on LTL planning focussed on translation based approaches. There, the original planning problem is altered in such a way that all solutions for the new problem adhere to the formula (e.g. (Baier and McIlraith 2006; Torres and Baier 2015), see (Camacho et al. 2017) for an overview). Traditionally, these approaches translate the LTL formula into a Büchi automaton (essentially a finite state machine, with a specific accepting criterion for infinite traces) and then integrate the automaton into the model. The major drawback of these translations is that the Büchi automaton for an LTL formula can have up to $2^{|\phi|}$ many states.

Torres and Baier (2015) proposed a translation diverging from the classical LTL to Büchi translation. They instead based it on Alternating Automata, which are commonly used as an intermediate step when constructing the Büchi automaton for an LTL ϕ formula (see e.g. (Gastin and Oddoux 2001)). Alternating Automata have a guaranteed linear size in $|\phi|$, but have a more complex transition function.

Definition 2. *Given a set of primitive propositions A, an alternating automaton is a 4-tuple $\mathcal{A} = (Q, \delta, I, F)$ where*

- *Q is a finite set of states*
- *$\delta : Q \times 2^A \rightarrow \mathcal{B}^+(Q)$ is a transition function, where $\mathcal{B}^+(Q)$ is the set of positive propositional formulas over the set of states Q, i.e., those formulae containing only $\vee$ and $\wedge$.*
- *$I \subseteq Q$ is the initial state*
- *$F \subseteq Q$ is set of final states.*

A run of an alternating automaton over a sequence of sets of propositions (execution trace) $(s_1, \ldots, s_n)$ is a sequence of sets of states $(Q_0, \ldots, Q_n)$ such that

- $Q_0 = I$
- $\forall i \in \{1, \ldots, n\} : Q_i \models \bigwedge_{q \in Q_{i-1}} \delta(q, s_i)$

The alternating automaton accepts the trace iff $Q_n \subseteq F$

Torres and Baier (2015) generate an alternating automaton for an LTL formula ϕ as follows. They choose Q as the set of sub-expression of ϕ starting with a temporal operator plus a state q_F representing that the end of the execution trace has been reached. Being in a state q means, that from the current time on, we have to fulfill the formula q. The automaton is given as $A_\phi = (Q, \delta, \{q_\phi\}, \{q_F\})$ where $Q = \{q_\alpha \mid \alpha \in \mathcal{S}(\phi)\} \cup \{q_F\}$ and the transition function δ is defined as follows:

$$\begin{aligned}
\delta(q_l, s) &= \begin{cases} \top & \text{, if } l \in s \\ \bot & \text{, if } l \notin s \end{cases} \\
\delta(q_F, s) &= \bot \\
\delta(q_{f \vee g}, s) &= \delta(q_f, s) \vee \delta(q_g, s) \\
\delta(q_{f \wedge g}, s) &= \delta(q_f, s) \wedge \delta(q_g, s) \\
\delta(q_{Xf}, s) &= q_f \\
\delta(q_{\mathring{X}f}, s) &= q_F \vee q_f \\
\delta(q_{Ef}, s) &= \delta(f, s) \vee q_{Ef} \\
\delta(q_{Gf}, s) &= \delta(f, s) \wedge (q_{Gf} \vee q_F) \\
\delta(q_{fUg}, s) &= \delta(q_g, s) \vee (\delta(q_f, s) \wedge q_{fUg})
\end{aligned}$$

Note that we have to enumerate all states that are relevant to the formula, i.e., all states $s \subseteq 2^{A(\phi)}$, to construct the formula. Using the Alternating Automaton as the basis for a translation leads to a guaranteed linear increase in size when constructing the translated planning problem. This is due to the fact that the encoding does not actually has to construct the automaton, but only has to simulate its states. We will elaborate on this later. Also, it was demonstrated that the new encoding is more efficient that other current translation techniques (Torres and Baier 2015).

A drawback of their translation was the need for introducing additional actions, performing bookkeeping on the current state of the alternating automaton. A translation into SAT, on the other hand, will not have this drawback, as we will show. We will again extend the $\exists$-step encoding and call the encoding AA (**A**lternating **A**utomaton). The restriction posed on parallelism by M&R'07 does not depend on the encoding of the formula itself, as long as it does not contain the X or $\mathring{X}$ operators[2]. We introduce new decision variables q^t for each state $q \in Q$ and timestep t, signifying that the automaton A_ϕ is in state q after executing the actions of timestep t. To express the transition function of the Alternating Automaton, we use formulae of the form $\left(q^t \wedge \bigwedge_{a \in s} a\right) \rightarrow \delta(q, s)$ for each state q of the automaton and set of propositions s. We also replace each occurrence of a state q_f in $\delta(q, s)$ with the decision variable q_f^{t+1} and introduce intermediate decision variables to break down complex formulae as in the M&R'07 encoding. The following theorem holds by construction.

Theorem 1. *AA in conjunction with M&R'07's $\exists$-step encoding is correct for LTL_{-X}.*

[2]The actual encoding of the formula can be exchanged in the proof of their main theorem as long as a similar version of their Theorem 2 can be proven, which is obvious in our case.

We here want to point out that the AA encoding is not (as the one by Torres and Baier (2015)) polynomial in the size of the formula. The reason lies in the explicit construction of the alternating automaton, which requires a single transition for every possible state that might be relevant to the formula, i.e., for every subset of $A(\phi)$. The translation encoding by Torres and Baier circumvents this construction by adding new operators, which can evaluate the necessary expression during planning. I.e. they have actions for each transition rule $\delta(\cdot, \cdot)$, which produce as their effects the right-hand sides of these above equations. Lastly, they introduce synchronisation actions to ensure that $\delta(\cdot, \cdot)$ is fully computed before another "real" actions is executed.

If we apply this idea to the AA encoding, we would end up with the M&R'07 encoding. Since the states of the automaton are the sub-formulae of ϕ starting with a temporal operator, these decision variables are identical to the ψ^t_{LTL} variables of the M&R'07 encoding, where ψ starts with a temporal operator. The encoding by Torres and Baier also needs to introduce state variables for every sub formulae not starting with a temporal operator to represent the step-wise computation of $\delta(\cdot, \cdot)$ correctly. If translated into propositional variables, these correspond to the ψ^t_{LTL} variables of M&R'07, where ψ does not start with a temporal operator. Lastly, the transition rules for both encodings are identical.

As such, the M&R'07 encoding can also be interpreted as a direct translation of an Alternating Automaton into propositional logic using the compression technique of Torres and Baier (2015). Interestingly, the original proof showing correctness of the LTL-part of the M&R'07 encoding by Latvala et al. (2004) does not rely on this relationship to Alternating Automata, neither do they mention this connection. We think it is an interesting theoretical insight, as it might enable to further improve LTL encoding, e.g., based on optimisations of the Alternating Automaton.

5 X, Parallelism, and Partial Evaluation

We have noted that both M&R'07 and AA cannot handle LTL formulae containing the X or $\mathring{X}$ operators in conjunction with the $\exists$-step encoding. They are however correct if used together with the sequential encoding, where only a single action to be executed at each timestep. In order to derive extensions that can handle X and $\mathring{X}$, we first present a new theoretical foundation for both encodings. We will use that fact that in M&R'07, we know which parts of the formula are made true at which time and by which propositions. To formalise this, we introduce evaluation traces, which specify how an LTL formula is fulfilled over a trace.

Definition 3. *Let ϕ be an LTL formula. We call a sequence $\psi = (f_0, \dots, f_n)$ with $f_i \subseteq \mathcal{S}(\phi)$ an evaluation trace for ϕ iff $\phi \in f_0$ and for all $i \in \{0, \dots, n\}$*

1. *if $f \vee g \in f_i$ then $f \in f_i$ or $g \in f_i$*
2. *if $f \wedge g \in f_i$ then $f \in f_i$ and $g \in f_i$*
3. *if $Xf \in f_i$ then $i < n$ and $f \in f_{i+1}$*
4. *if $\mathring{X}f \in f_i$ then $i = n$ or $f \in f_{i+1}$*
5. *if $Ef \in f_i$ then $f \in f_i$ or $i < n$ and $Ef \in f_{i+1}$*
6. *if $Gf \in f_i$ then $f \in f_i$ and if $i < n$ then $Gf \in f_{i+1}$*

7. if $fUg \in f_i$ then $g \in f_i$ or $i < n$ and $f \in f_i$ and $fUg \in f_{i+1}$

A trace $\pi = (s_0, \dots, s_n)$ satisfies an evaluation trace $\psi = (f_0, \dots, f_n)$ iff for all $a \in f_i \cap A$ also $a \in s_i$.

The following theorem follows directly, as the definition just emulates checking an LTL formula.

Theorem 2. *An execution trace π satisfies an LTL formula ϕ iff an evaluation trace ψ for ϕ exists that satisfies π.*

In M&R'07, the LTL formula is only evaluated after a set of parallel actions have been executed. To capture this, we define partial evaluation traces.

Definition 4. *Let $\pi = (s_0, \dots, s_n)$ be an execution trace and ϕ and LTL formula. We call an evaluation trace $\theta = (f_0, \dots, f_l)$ with $l \leq n$ a partial evaluation trace (PET) for π if a sequence of indices $0 = i_0 < i_1 < \dots < i_l = n + 1$ exists such that for each $k \in \{0, \dots, l-1\}$ holds*

$$s_{i_k} \cap (f_k \cap A) = \dots = s_{i_{k+1}-1} \cap (f_k \cap A)$$

and if $Xf \in f_k$ or $\overset{\circ}{X} f \in f_k$ and $k > 0$ then $i_{k-1} + 1 = i_k$. The PET θ is satisfied by the execution trace π, iff the execution trace $(s_{i_1-1} \cap f_0 \cap A, \dots, s_{i_l-1} \cap f_l \cap A)$ satisfies θ in the sense of Def. 3.

A satisfying valuation of the M&R'07 encoding corresponds to a partial evaluation trace that satisfies the formula ϕ. M&R'07 also asserts that PET is satisfied by the execution trace corresponding to the sequential plan generated by the $\exists$-step formula. The main property of Def. 4, which is necessary for showing that we actually generate a PET, is ensured by the chains added to the original $\exists$-step encoding and the additional edges in the Disabling Graph. The following theorem states that every partial evaluation trace that is satisfied by an executed trace can be extended to a full evaluation trace and thus forms a witness that the execution trace satisfies the LTL formula. This gives us a second, independent proof of correctness for the M&R'07 encoding.

Theorem 3. *Let π be a trace and θ be an PET for the formula ϕ on π. If π satisfies θ, then π satisfies ϕ.*

Proof. We need to show that the PET $\psi = (f_0, \dots, f_l)$ can be extended to a full evaluation trace on $\pi = (s_0, \dots, s_n)$, s.t. π satisfies that evaluation trace. If so, we can apply Thm. 2 and conclude that π also satisfies ϕ. Let $i_0, \dots, i_l$ be the indices of Def. 4 for which θ is a PET. We claim that

$$\theta^* = (\overbrace{f_0, \dots f_0}^{i_1 - i_0 \text{ times}}, \overbrace{f_1, \dots, f_1}^{i_2 - i_1 \text{ times}}, \dots, \overbrace{f_l, \dots, f_l}^{i_l - i_{l-1} \text{ times}})$$

is an evaluation trace that satisfies π, and that π satisfies ϕ. First we show that θ^* is an evaluation trace. We start by proving the enumerated properties of Def. 3. Consider the ith element f^* of θ^* and let f^{**} be the $i + 1$th element (if such exists).

1. trivially satisfied
2. trivially satisfied
3. $Xf \in f^*$. We know that f^* is the only repetition some f_j in the trace, as θ is a PET. Also $f^{**} = f_{j+1}$. Consequently $f \in f^{**}$. In case f^{**} does not exist, θ cannot be an PET.
4. $\overset{\circ}{X} f \in f^*$. We know that f^* is the only repetition of some f_j in the trace, as θ is a PET. In case f^{**} does not exist, we have nothing to show. Else, $f^{**} = f_{j+1}$ and $f \in f^{**}$.

For the last three requirements relating to the temporal operators E, G, and U, we can distinguish three cases depending on where f^* is situated in the sequence θ^*

- f^* is the last element of θ^*
  5. if $Ef \in f^*$ then $f \in f^*$, as θ is a PET.
  6. if $Gf \in f^*$ then $f \in f^*$, as θ is a PET.
  7. if $fUg \in f^*$ then $g \in f^*$, as θ is a PET.
- $f^* \neq f^{**}$, i.e., the last repetition of f^*. We know that $f^* = f_j$ and $f^{**} = f_{j+1}$ for some $j \in \{0, \dots, l-1\}$.
  5. if $Ef \in f^*$ then either $f \in f_j = f^*$ or $Ef \in f_{j+1} = f^{**}$
  6. if $Gf \in f^*$ then $f \in f_j = f^*$ and $Gf \in f_{j+1} = f^{**}$
  7. if $fUG \in f^*$ then either $g \in f_{j+1} = f^{**}$ or $f \in f_j = f^*$ and $fUg \in f_{j+1} = f^{**}$
- if $f^* = f^{**}$
  5. if $Ef \in f^*$ then $Ef \in f^{**}$
  6. if $Gf \in f^*$ then $Gf \in f^{**}$ and $f \in f^*$
  7. if $fUg \in f^*$ then either $g \in f^*$, or $f \in f^*$, but then also $fUG \in f^{**}$, since $f^* = f^{**}$

$\phi \in f_0$ holds as θ is a PET, which concludes conclude the proof that θ^* is an evaluation trace.

Lastly, we have to show that θ^* satisfies π, i.e., we have to show for each timestep $j \in \{0, \dots n\}$ and every $a \in A$ with is true in the ith element of θ^* that $a \in s_j$ holds. Consider first the indices of the last repetitions of each f_k, i.e., the states $s_{i_k - 1}$. Since θ is a PET, it satisfies $(s_{i_1-1} \cap f_0 \cap A, \dots, s_{i_l-1} \cap f_l \cap A)$, so it satisfies the required property for all time-steps $i_k - 1$. Consider any other timestep t and its next timestep in the PET $i_k - 1$ (which always exists, since the last index is equal to n). Since θ is a PET, we know that $s_t \cap (f_k - 1 \cap A) = s_{i_k-1} \cap (f_k \cap A)$. We have chosen to set the tth element of θ^* to f_k. Since $s_{i_k-1} \cap (f_k \cap A)$ satisfies the required property for f_k, so must $s_t \cap (f_k \cap A)$ and thus s_t itself (it can have only more true predicates). $\square$

We can now use this result to integrate support for X and $\overset{\circ}{X}$ into the M&R'07 encoding. For that, we have to assert that the second last condition of Def. 4 holds, as all other requirements are already checked by the M&R'07 encoding. We first add four new variables per time step t.

- $exactOne^t$ – exactly one action is executed at time t
- $atLeastOne^t$ – at least one action is executed at time t
- $atMostOne^t$ – at most one action is executed at time t
- $none^t$ – no action is executed at any time $\geq t$

To enforce the semantics of these variables, we add the following clauses per timestep:

$$\forall o \in O : none^t \rightarrow \neg o^t$$
$$none^t \rightarrow none^{t+1}$$
$$atLeastOne^t \rightarrow \bigvee_{o \in O} o^t$$
$$exactOne^t \rightarrow atLeastOne^t \wedge atMostOne^t$$

Encoding the $atMostOne^t$ atom is a bit more complicated. A native encoding requires $\mathcal{O}(|O|^2)$ clauses. There are however better encodings for the at most one constraint in SAT. We have chosen the log-counter encoding, which introduces $\log(|S|)$ new variables while only requiring $|S|\log(|S|)$ clauses (Frisch et al. 2005). To ensure the semantics of the atom $atMostOne^t$, we add it as a guard to the log-counter encoding, i.e., we add $\neg atMostOne^t$ to every clause. If $atMostOne^t$ is required to be true, the log-counter clauses have to satisfies, i.e., at most one action atom can be true. If $atMostOne^t$ can be false, we can simply set it to $\bot$ thereby satisfying all log-counter clauses. We lastly set $exactOne^t$ for $t = -1$ to $\top$ and $atLeastOne^{n+1}$ to false. Based on these new variables, we can add the constraints necessary for evaluating X and $\mathring{X}$ correctly under the $\exists$-step encoding. For each timestep t, we add

$$(Xf)^t_{LTL} \to exactOne^{t-1} \land f^{t+1} \land atLeastOne^t$$
$$(\mathring{X}f)^t_{LTL} \to exactOne^{t-1}$$
$$(\mathring{X}f)^t_{LTL} \to (f^{t+1} \land atLeastOne^t) \lor none^t$$

A valuation of this encoding represents a partial evaluation trace satisfied by the execution trace of its actions. By applying Thm. 3, we know that a satisfying evaluation trace for the plan exists. Showing completeness for the encoding is trivial, since a sequential assignment satisfies the formula for every plan. We will also call this extended encoding M&R'07 in our evaluation, as it is exactly identical to this one, provided no X or $\mathring{X}$ operator is present.

So far, we have not used the – in our view – most significant improvement: Relaxing the restrictions on parallelism to only those actually needed by the current formula. Doing so based on our theorem is surprisingly easy. Consider a timestep of the M&R'07 encoding, in which actions can be executed in parallel, which are given an implicit order $\prec$ by the $\exists$-step encoding. For each literal $l \in Lit(A(\phi))$ a chain ensures that the action changing it is applied first, i.e., that the "first" action of a timestep performs all changes relevant to the whole formula and can thus check the formula only in the resulting state. By applying Def. 4 and Thm. 3, we have to ensure this property only for those proposition symbols that need to be evaluated after the actions have been executed, i.e., for all a^{t+1}_{LTL} that are true. We can do this simply by adding the literal $\neg a^t_{LTL}$ as a guard (simiar to $atMostOne^t$) to every clause in the chains $chain(\prec; E^\phi_l; R^\phi_l; \phi_a)$ and $chain(\prec; E^\phi_l; R^\phi_l; \phi_{\neg a})$. If we have to make the literal a^t_{LTL} true, the chains become active, if not they are inactive (as they are trivially satisfied by $\neg a^t_{LTL}$). We will denote this improved encoding with Improved-M&R'07.

To illustrate the effects of the improved M&R'07 encoding, consider the following planning problem. There are six proposition symbols a, b, c, d, e and f, of which a and b are true in the initial state and e has to hold in the goal state. There are five actions described in Tab. 2. We consider this domain in conjunction with the formula $\phi = G((a \land d) \to Ef) = G(\neg a \lor \neg d \lor Ef)$. The disabling graph, extended by the edges needed for the M&R'07 encoding, is depicted in Fig. 1. This planning problem has only a single solution, namely: Y, X, V, W, Z. Under the M&R'07 encoding, we need four timesteps to find a plan, i.e., {Y}, {X}, {V}, {W,Z}. This is due to the fact that most action's effects contain one of the three predicates contained in ϕ, but not the others. Using the improved M&R'07 encoding, we only need three timesteps, as now Y and X can be executed together in the first timestep. The reason for this beeing possible is quite unintuitive, but it shows the strength of our approach. In the M&R'07 encoding, the encoding correctly detects that there is a state in which a and d are true simultaneously[3] after the action Y has been executed and that thus Ef has to hold after executing Y. In the plan for the improved encoding, the solver can simply choose to achieve Ef after {Y,X} has been executed. This way the solver is never forced to achieve $\neg a$ or $\neg d$ and can thus completely ignore the associated constraints, i.e., chains.

	X	Y	Z	V	W
pre	a,b	a,b	c,d	c,d	g
add	c	d	e	g	f
del	a		c		

Table 2: Actions in the example domain

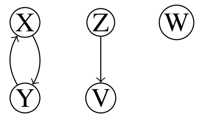

Figure 1: Extended disabling graph for the example domain.

6 Parallelism with Tracking

There is however still room for more parallelism even compared to the improved M&R'07 encoding. The key observation is that in many domains only a few actions actually influence the truth of variables in an LTL formula, and that those that do are usually close to each other in a topological ordering of the inverse disabling graph. The $\exists$-step semantic guarantees that if actions are executed in parallel, they can be sequentially executed in this order. Let this ordering be $(o_1, \dots, o_n)$. We can divide it into blocks, such that for each block $(o_i, \dots, o_j)$ it holds that

$$[o_i]^\phi_e \supseteq [o_{i+1}]^\phi_e \supseteq \cdots \supseteq [o_j]^\phi_e$$

Along the actions in a block the effects that contain predicates in $A(\phi)$ can only "decrease". A block forms a set of actions that can always – without further checking at runtime – be executed in parallel in the M&R'07 encoding. The number of such blocks is surprisingly small for most domains (see Tab. 3). We denote with $B = ((o_1, \dots, o_i), \dots, (o_j, \dots, o_n))$ the sequence of blocks for a given ordering of actions.

If an action from a blocks has been executed, at least (usually more) the first action of the next block cannot be executed any more in the same timestep, as it would change the truth of some $a \in A(\phi)$, even it would be possible

[3]Technically both $\overline{a}$ and $\overline{d}$ are not true.

in the pure $\exists$-step encoding. We present a method to circumvent this restriction on parallelism. Instead of restricting the amount of parallel actions executing inside a timestep, we (partially) trace the truth of an LTL formula within that timestep to allow maximal parallelism. This is based on the insight that all proofs by Mattmüller and Rintanen (2007) do not actually require an action to be present at any timestep, i.e., the set of actions executed in parallel can also be empty. So, conceptually, we split each timestep into $|B|$ many timesteps and restrict the actions in the ith splitted step to be those of the ith block. Then we use the M&R'07 encoding, without the need to add chain-clauses apart from those of the $\exists$-step encoding, as they are automatically fulfilled. The resulting encoding would be sound and complete for LTL formulae without X and $\mathring{X}$ by virtue of the results proven by Mattmüller and Rintanen (2007).

We can however improve the formula even further. In the proposed encoding, we would compute the state after each block using the Kautz&Selman encoding. This is unnecessary, as know from the $\exists$-step encoding that we only need to compute it after all blocks of one original timestep have been executed. We only need to trace the truth of propositions in $A(\phi)$ between blocks. For that we don't need to check preconditions – they are already ensured by the $\exists$-step encoding. We end up with the $\exists$-step encoding, where we add at every timestep a set of intermediate timepoints at which the truth of propositions in $A(\phi)$ and the truth of the LTL formula ϕ is checked. Thus we call this encoding OnParallel.

As we have noted above, this construction works only for the original M&R'07 encoding, as supporting the X and $\mathring{X}$ requires to be able to specify that in the next timestep some action must be executed. This might not be possible with splitted timesteps, as the next action to be executed may only be contained in a timestep $|B|$ steps away. Luckily, we can fix this problem by slightly altering the encoding we used to track the truth of X and $\mathring{X}$ operators.

$$(Xf)^t \rightarrow atMostOne^{t-1} \wedge ((atLeastOne^{it} \wedge f^{t+1}_{LTL}) \vee (nextNone^{t+1} \wedge (Xf)^{t+1}_{LTL}))$$

$$(\mathring{X}f)^t \rightarrow atMostOne^{t-1} \wedge ((f^{t+1}_{LTL} \wedge atLeastOne^{t}) \vee (nextNone^{t+1} \wedge (\mathring{X}f)^{t+1}_{LTL}) \vee none^{t}))$$

The semantics of $noneAt^t$ is ensured by clauses $nextNone^t \rightarrow \neg o^t$ for all $i \in O$. Lastly, we add $\neg Xf^{n++11}_{LTL}$ for any $Xf \in \mathcal{S}$ stating that a next-formula cannot be made true at the last timestep. This would else be possible, since $atMostOne^n$ could simply be made true. The OnParallel encoding is correct by applying Thm. 3.

7 Evaluation

We have conducted an evaluation in order to asses the performance of our proposed encodings. We used the same experimental setting as Mattmüller and Rintanen (2007) in their original paper. We used the domains `trucks` and `rover` from the preference track of IPC5 (the original paper considered only `rover`), which contain temporally extended goals to specify preference. In these domains, temporally-extended goals are formulated using the syntax of PDDL 3.0 (Gerevini and Long 2005). We parse the preferences and transform them into LTL formulae using the patterns defined by Gerevini and Long (Gerevini and Long 2005). As did Mattmüller and Rintanen, we interpret these preferences as hard constraints and randomly choose a subset of three constraints per instance[4]. To examine the performance of our encoding for X and $\mathring{X}$, we have also tested the instances of the `trucks` domain with a formula that contains these operators. We have used the following formula[5]:

$$\phi = \forall ?l\text{–}Location ?t - Truck : G(at(?l, ?t) \rightarrow \mathring{X}(\neg at(?l, ?t) \vee \mathring{X}\neg at(?l, ?t)))$$

It forces each truck to stay at a location for at most one timestep – either it leaves the location right after entering it, or in the next timestep. The domain contains an explicit symbolic representation of time, which is used in temporal goals. When planning with ϕ, the number of timesteps is never sufficed to find a plan satisfying ϕ. We have therefore removed all preconditions, effects, and action parameters pertaining to the explicit representation of time. As a result, the domain itself is easier than the original one. We denote these instances in the evaluation with trucks-XY-ϕ.

Each planner was given 10 minutes of runtime and 4GB of RAM per instance on an Intel Xeon E5-2660 v3. We've used the SAT solver Riss6 (Manthey, Stephan, and Werner 2016), one of the best-performing solvers in the SAT Competition 2016. We have omitted results for all the trucks instances $11 - 20$, as no planner was able to solve them.

Table 3 shows the results of our evaluation. We show per instance the number of ground actions and blocks. The number of blocks is almost always significantly smaller than the number of ground operators. In the largest `rover` instance, only $\approx 1.4\%$ of operators start a new block. For the `trucks` domain this is $\approx 1.7\%$ for the largest instance.

For every encoding, we show both the number of parallel steps (i.e. timesteps) necessary to find a solution, as well as the time needed to solve the respective formula and the number of sequential plan steps found by the planner. In almost all instances the OnParallel encoding performs best, while there are some where the improved M&R'07 encoding is faster. Our improvement to the M&R'07 encoding almost always leads to a faster runtime. Also, the improved parallelism actually leads to shorter parallel plans. In approximately half of the instances we can find plans with a fewer parallel steps. In the experiments with the formula ϕ containing the $\mathring{X}$ operator, this is most pronounced. The OnParallel encoding cuts the number of timesteps by half and is hence significantly faster, e.g., on trucks-03-ϕ where the runtime is reduced from 165s to 6s. On the other hand, the sequential plans found are usually a few actions longer, although the same short plan could be found – this result is due to the non-determinism of the SAT solver.

[4]Mattmüller and Rintanen (2007) noted that it is impossible to satisfy all constraints at the same time and that a random sample of more than three often leads to unsolvable problems. If a sample of 3 proved unsolvable we have drawn a new one.

[5]We handlethese lifted LTL constraints by grounding them using the set of delete-relaxed reachable ground predicates.

Instance			M&R'07			Improved-M&R'07			OnParallel			AA		
	$\lvert O\rvert$	$\lvert B\rvert$	p. steps	s. steps	time	p. steps	s. steps	time	p. steps	s. steps	time	p. steps	s. steps	time
rover-01	63	10	7	16	**0.10**	7	16	0.11	7	16	**0.10**	7	**14**	0.23
rover-02	53	8	5	13	0.06	**4**	13	0.03*	**4**	14	**0.03**	4	**12**	0.04
rover-03	76	9	7	16	0.25	7	20	0.2*	**6**	17	**0.16**	7	**14**	0.35
rover-04	86	16	5	13	0.13	5	11	0.12*	**3**	**9**	**0.05**	5	14	0.12
rover-05	144	15	7	**26**	0.38	7	30	0.39	**6**	27	**0.29**	7	**26**	0.63
rover-06	178	16	10	42	1.08	**9**	42	0.73*	**8**	43	**0.6**	10	**39**	1.39
rover-07	151	7	5	31	0.37	5	29	0.33*	5	28	**0.31**	5	**21**	0.59
rover-08	328	9	7	40	1.06	7	43	1.11	7	45	**0.95**	7	**36**	1.24
rover-09	362	19	9	62	3.47	9	56	**3.13***	9	59	3.38	9	**36**	3.92
rover-10	382	14	6	54	1.73	**5**	46	1.17*	**4**	44	**0.77**	6	**40**	1.62
rover-11	436	9	10	**42**	3.91	10	**42**	3.77*	**9**	43	**2.93**	10	47	6.46
rover-12	366	15	5	27	1.28	5	29	1.18*	5	33	**1.07**	5	**25**	1.43
rover-13	749	11	8	**61**	4.51	8	65	4.75	8	66	**3.99**	8	62	5.27
rover-14	525	19	7	**40**	3.66	7	43	3.66	7	42	**3.31**	7	50	4.13
rover-15	751	12	8	64	6.18	**7**	60	4.85*	**7**	60	**4.36**	7	**47**	5.34
rover-16	671	14	6	44	3.66	6	47	3.41*	6	50	**3.32**	6	**43**	3.99
rover-17	1227	13	11	106	34.97	11	105	34.49*	11	103	**31.45**	11	**62**	37.17
rover-18	1837	49	5	65	10.63	5	65	11.04	5	**64**	**10.11**	5	70	10.59
rover-19	2838	17	8	91	20.28	8	94	19.63*	8	100	**19.00**	8	**86**	21.38
rover-20	3976	58	8	130	65.99	8	127	65.97*	8	119	**65.2**	8	**98**	66.38
trucks-01	333	82	8	15	0.69	**7**	**14**	**0.44***	**7**	15	0.50	8	15	0.98
trucks-02	624	56	9	20	1.73	**8**	**17**	1.26*	**8**	18	**1.21**	9	18	1.87
trucks-03	1065	68	10	24	2.43	10	**22**	2.47	**9**	**22**	**1.68**	10	23	6.34
trucks-04	1692	48	12	27	8.48	11	27	**6.68***	11	26	6.93	12	**25**	12.80
trucks-05	2541	85	12	29	13.70	11	30	11.07*	11	**27**	**10.56**	12	28	13.37
trucks-06	4928	112	14	37	42.01	14	36	**39.39***	14	**35**	41.37	14	**35**	88.09
trucks-07	7380	267	14	40	171.81	**13**	38	119.95*	**13**	39	**119.75**	14	**36**	247.5
trucks-08	9760	260	13	42	**151.22**	13	41	153.05	13	**39**	153.81	–	–	TLE
trucks-09	12628	203	14	45	432.99	14	**42**	373.30*	14	44	**362.54**	14	43	465.15
trucks-10	16032	267	–	–	TLE	–	–	TLE	–	–	TLE	–	–	TLE
trucks-01-ϕ	123	6	18	18	2.12	18	18	2.27	**9**	18	**0.24**	18	18	18.45
trucks-02-ϕ	162	6	24	24	14.44	24	24	14.55	**12**	24	**1.09**	24	24	498.93
trucks-03-ϕ	201	6	29	29	164.56	29	29	165.64	**15**	29	**5.97**	–	–	TLE
trucks-04-ϕ	240	6	–	–	TLE	–	–	TLE	**18**	36	**84.20**	–	–	TLE
trucks-05-ϕ	279	6	–	–	TLE	–	–	TLE	–	–	TLE	–	–	TLE

Table 3: SAT-solver runtime and solution length of several encodings. Per run, we give both the number of parallel steps (p. steps) and the number of sequential (s. steps). Minimum run-times and sequential plan lengths are marked in bold. If the number of parallel plan steps decreased for any of the two new encodings, they are also marked bold. Decreases in Runtime for the improve M&R'07 encoding compared to the original one are marked with an asterisk. TLE indicates exceeding the timelimit of 10 minutes.

8 Conclusion

In this paper, we have improved the state-of-the-art in translating LTL planning problems into propositional formulae in several ways. We have first pointed out an interesting theoretical connection between the propositional encoding by Mattmüller and Rintanen (2007) and the compilation technique by Torres and Baier (2015). Next, we have presented a new theoretical foundation for the M&R'07 encoding – partial evaluation traces. Using them, we presented (1) a method to allow the X and $\overset{\circ}{X}$ operators in the M&R'07 encoding, (2) a method to further improve the parallelism in the M&R'07 encoding, and (3) a new encoding for LTL planning. In an evaluation, we have shown that both our improved M&R'07 encoding and the OnParallel encoding perform empirically better than the original encoding by Mattmüller and Rintanen. We plan to use the developed encoding in a planning-based assistant (Behnke et al. 2018) for enabling the user to influence the instructions he is presented by the assistant, which is turn are based on the solution generated by a planner. Instructions given by the user can be interpreted as LTL goal and integrated into the plan using the presented techniques.

Acknowledgement

This work is funded by the German Research Foundation (DFG) within the technology transfer project "Do it yourself, but not alone: Companion Technology for Home Improvement" of the Transregional Collaborative Research Centre SFB/TRR 62. The industrial project partner is the Corporate Research Sector of the Robert Bosch GmbH.

References

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

Behnke, G.; Schiller, M.; Kraus, M.; Bercher, P.; Schmautz, M.; Dorna, M.; Minker, W.; Glimm, B.; and Biundo, S. 2018. Instructing novice users on how to use tools in DIY projects. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence and the 23rd European Conference on Artificial Intelligence (IJCAI-ECAI 2018)*. AAAI Press.

Biere, A.; Heljanko, K.; Junttila, T.; Latvala, T.; and Schuppan, V. 2006. Linear encodings of bounded LTL model checking. *Logical Methods in Computer Science* 2(5):1–64.

Camacho, A.; Triantafillou, E.; Muise, C.; Baier, J. A.; and McIlraith, S. A. 2017. Non-deterministic planning with temporally extended goals: Ltl over finite and infinite traces. In *Proceedings of the 31st National Conference on AI (AAAI 2017)*, 3716–3724. AAAI 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 2013)*, 854–860. AAAI Press.

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

Edelkamp, S. 2003. On the compilation of plan constraints and preferences. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS 2006)*. AAAI Press.

Frisch, A.; Peugniez, T.; Doggett, A.; and Nightingale, P. 2005. Solving non-boolean satisfiability problems with stochastic local search: A comparison of encodings. *Journal of Automated Reasoning (JAR)* 35(1-3):143–179.

Gastin, P., and Oddoux, D. 2001. Fast ltl to büchi automata translation. In *Proceedings of the 13th International Conference on Computer Aided Verification (CAV 2001)*, 53–65. Springer-Verlag.

Gerevini, A., and Long, D. 2005. Plan constraints and preferences in PDDL3. Technical report, Department of Electronics for Automation, University of Brescia.

Hsu, C.-W.; Wah, B.; Huang, R.; and Chen, Y. 2007. Constraint partitioning for solving planning problems with trajectory constraints and goal preferences. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, 1924–1929. AAAI Press.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI)*, 1194–1201.

Latvala, T.; Biere, A.; Heljanko, K.; and Junttila, T. 2004. Simple bounded LTL model checking. In *Proceedings of the 5th Conference on Formal Methods in Computer-Aided Design (FMCAD 2004)*, 189–200. FMCAD Inc.

Manthey, N.; Stephan, A.; and Werner, E. 2016. Riss 6 solver and derivatives. In *Proceedings of SAT Competition 2016: Solver and Benchmark Descriptions*. University of Helsinki.

Mattmüller, R., and Rintanen, J. 2007. Planning for temporally extended goals as propositional satisfiability. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, 1966–1971. AAAI Press.

Mattmüller, R. 2006. Erfüllbarkeitsbasierte Handlungsplanung mit temporal erweiterten Zielen.

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

Rintanen, J.; Heljanko, K.; and Niemelä, I. 2006. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence* 170(12-13):1031–1080.

Torres, J., and Baier, J. A. 2015. Polynomial-time reformulations of LTL temporally extended goals into final-state goals. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, 1696–1703. AAAI Press.