PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE
SCHOOL OF ENGINEERING

POLYNOMIAL-TIME REFORMULATIONS OF TEMPORALLY EXTENDED PLANNING PROBLEMS INTO CLASSICAL PLANNING PROBLEMS

JORGE ANDRÉS TORRES VILLARRUBIA

Thesis submitted to the Office of Research and Graduate Studies
in partial fulfillment of the requirements for the degree of
Master of Science in Engineering

Advisor:
JORGE A. BAIER A.

Santiago de Chile, January 2016

PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE
SCHOOL OF ENGINEERING

POLYNOMIAL-TIME REFORMULATIONS OF TEMPORALLY EXTENDED PLANNING PROBLEMS INTO CLASSICAL PLANNING PROBLEMS

JORGE ANDRÉS TORRES VILLARRUBIA

Members of the Committee:
JORGE A. BAIER A.
MARCELO A. ARENAS S.
JORGE PÉREZ R.
JUAN DE DIOS RIVERA

Thesis submitted to the Office of Research and Graduate Studies
in partial fulfillment of the requirements for the degree of
Master of Science in Engineering

Santiago de Chile, January 2016

To my family and friends.

ACKNOWLEDGEMENTS

First, I would like to thank my advisor Jorge Baier, for his kindness, patience, for giving me the chance to attend the IJCAI conference at Buenos Aires and meet great people there. I also thank Marcelo Arenas and Jorge Perez. With Baier, they were the best teachers I ever had to study Theoretical Computer Science.

I would also like to thank my father Jorge, my sister Fernanda and my grandparents Andres and Beatriz by encouraging me in this work. I specially thank my mother Pilar by helping me day to day with her great and wise advices. I have learnt a lot about life and how to understand the world thanks to her.

And many thanks to my Synopsys colleagues and friends for cheering me up in this thesis and making me feel welcome and happy every day.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Linear temporal logic (LTL) is an expressive language that allows specifying temporally extended goals and preferences. A general approach to dealing with general LTL properties in planning is by "compiling them away"; i.e., in a pre-processing phase, all LTL formulas are converted into simple, non-temporal formulas that can be evaluated in a planning state. This is accomplished by first generating a finite-state automaton for the formula, and then by introducing new fluents that are used to capture all possible runs of the automaton. Unfortunately, current translation approaches are worst-case exponential on the size of the LTL formula.

In this thesis, we present a polynomial approach to compiling away LTL goals. Our method relies on the exploitation of alternating automata. Since alternating automata are different from non-deterministic automata, our translation technique does not capture all possible runs in a planning state and thus is very different from previous approaches. We prove that our translation is sound and complete. We also show other variants of this translation that help to improve performance and evaluate them empirically showing their strengths and weaknesses. Specifically, we find classes of formulas in which our translation seems to outperform significantly the current state of the art.

Keywords: Planning State, Alternating Automata, Linear Temporal Logic, LTL formula, Planner, Planning problem, Temporally extended goal

RESUMEN

Lógica lineal temporal (LTL) es un lenguaje que permite especificar objetivos y preferencias temporales. Un método general para tratar en general con propiedades LTL en planificación es por "compilación"; por ejemplo, en una fase de preprocesamiento, todas las fórmulas temporales son transformadas a fórmulas simples y no temporales que pueden evaluarse en un estado de planificación. Esto se logra primero generando un autómata finito no determinista para la fórmula, y luego agregando fluentes nuevos que son usados para capturar todas las posibles ejecuciones del autómata. Desafortunadamente, los métodos de traducción actuales son exponenciales en el peor caso sobre el tamaño de la fórmula.

En esta tesis, presentamos un método polinomial para compilar objetivos temporales. Nuestro método utiliza autómatas alternantes. Dado que los autómatas alternantes son diferentes de los no deterministas, nuestra técnica de traducción no captura todas las posibles ejecuciones en un estado de planificación, y por lo tanto, es muy diferente de otros métodos. Nosotros demostramos que nuestra traducción es correcta y completa. También mostramos otras variantes de nuestro método que ayudan a mejorar el desempeño para algunos planificadores y evaluamos empíricamente mostrando que tiene ventajas y desventajas. Específicamente encontramos clases de fórmulas de las cuales nuestro método tiene mejor desempeño que el actual estado del arte.

Palabras Claves: Estado de planificación, Autómata alternante, Lógica Lineal Temporal, fórmula LTL, Planificador, Problema de planiciación, Objetivo temporalmente extendido

1. INTRODUCTION

1.1. Background

1.1.1. Planning in Artificial Intelligence

In Computer Science and Artificial Intelligence, there are problems that can be formulated as planning tasks, in which the solution consists of finding a sequence of *actions* or a *plan* that reaches a goal inside of a specific world. For example, finding a plan to travel from workplace way back to home or prepare a meal, where the cooking recipe has a "plan" to cook the meal. The world is represented or described by a set of atomic *facts* or *state variables*. The actions are operations that change the current state of the world, by making some *facts* true and other *facts* false. The goal is a condition or *property* that once it is fulfilled, the problem is solved.

One of the well known Planning models is Classical Planning, where actions are performed by a single *agent*. These actions are *deterministic*, meaning that executing the same action on the same planning state always yields the same resulting planning state. For example, a cubic dice can be described by 6 possible states, depending on the number shown on its top. The action of *rolling a dice* is non-deterministic, since rolling twice the dice with the same number on its top might give the same result (both dices might have different resulting numbers). But, if we consider a switch that has two possible states (on and off), then the action of *toggling the switch* is deterministic, because if the switch was turned off, then toggling it will turn it on and viceversa.

In this model of planning, the single agent also can *fully observe* the environment, this means that the agent has full information of the state of the environment. For example, the tic-tac-toe game is a fully observable environment, but a Blackjack game in where some cards are put face down is not fully observable.

With this description, Planning can also be seen as finding a path on a *graph*, in which the nodes correspond to the states of the world and the edges correspond to the transitions made by the actions from one state to the other. The agent must reach the goal, starting from an initial state. The goal can be seen as a single state or a set of states.

As an example, we can model environments as the *blocksworld* domain. In this world, there is a crane, a table and blocks stacked on the table. The main goal is to build a specified target stack of blocks using the crane. The crane can move a single block at a time and only if such a block is the topmost block of the stack. The crane can put the block over another block (if this one doesn't have another one on top of it) or on the table. The states correspond to the different stacks that are built on the table and the single action is *move block*.

Another example is a robot that must move through an office building and must deliver coffee to certain office rooms. In the building, there are offices and kitchens. Rooms can be entered through a door which is opened or closed. The robot can open or close doors, prepare a cup of coffee in the kitchen, move from one room to another if the door that connects them is opened and leave the cup of coffee in an office room. The robot can only carry a single cup of coffee at a time. The state can be described by which room is the robot located at, whether or not the robot is carrying a cup of coffee, which office rooms have a cup of coffee served and which doors are opened or closed.

1.1.2. Planning Technology

Nowadays there are many classical *planners* that can solve planning problems. Planners usually receive as input the representation of a planning problem and give as output or solution the *plan* that reaches or satisfies the planning goal. Most planners work by representing the problem as a *state-search* problem, which is equivalent to represent the problem using a directed graph and solve the *reachability problem*: Given a starting state and a goal state, is it possible to find a path in the graph:

reach the goal state from the starting state by following the edges of the directed graph? If so, how close to the optimal path is this found one? For the second question, planners use what is called *heuristic search* (Bonet and Geffner). This is a method that adds information to the search algorithm about an estimation of how close the planner is to the goal based on the current state.

For example, suppose that the environment consists of a $n \times n$ grid where the agent is in one cell of the grid and must reach a target cell inside the grid. The agent can only move up, down, left or right and there are some cells that cannot be stepped on. Each cell is specified by its position (x, y) on the grid, with position $(0, 0)$ representing the top-left corner, x increases as the agent moves down and y increases as it moves right. A possible heuristic for this problem can be the *Manhattan distance*, which consists of calculating the sums of the strictly horizontal and vertical distances between the current cell that the agent is standing in and the target cell that it must reach. This calculation assumes that the agent will not find some blocking cells that will prevent the agent to keep moving. This assumption is called a *relaxation*, and most planners calculate heuristics by relaxing the problem first and then, solve the relaxed problem.

To represent a planning problem, the de-facto language standard is PDDL. PDDL stands for Planning Domain Description Language, which is a standard encoding for Classical Planning problems. A planning problem that uses PDDL is separated into two files: A Domain file and a Problem file. The Domain file contains information about types of objects, predicates and actions that are specific to represent the environment. The Problem file contains the information about the instantiated objects, the initial planning state and the goal state.

1.1.3. Temporally extended goals

In Classical Planning, the goal is specified with a propositional boolean formula. A state that satisfies this formula can be considered a goal state. For example, consider one of the previous examples about the robot that must carry cups of

coffee to the office rooms and the predicate $coffee\_at(x)$ indicates that a cup of coffee has been served at office room x. If the formula $coffee\_at(lobby\_room) \wedge coffee\_at(main\_room) \wedge \neg coffee\_at(office1) \wedge \neg coffee\_at(office2)$ is used as the goal of the planning problem, it means that the robot must reach a state where the coffee is served at the main room and at the lobby room, but neither it is served at office 1 nor at office 2. However, there's no specification about which order should the robot serve the cups of coffee, whether it should be served on the lobby room before the main room or viceversa.

With the aforementioned example, Classical Planning can only define goals about the final planning state, but not about the path that reaches it. There's no guarantee that the found path satisfies given desirable properties or constraints. For this, we need to express the goals with another language. In this thesis, we consider the language is **Finite Linear Temporal Logic** ($f - LTL$) and allows to model planning problems with temporally extended goals using temporal formulae.

A temporal formula is a logic formula that uses propositional and temporal operators and instead of using a subset of propositional variables to verify if a formula is satisfied, we use a *finite sequence* of subsets of propositional variables. The sequence can be seen as a *timeline* where variables change their truth assignment in function of the *time* or position of each subset in the sequence. Thus, if we want the robot to serve the coffee at the lobby room before it is served at the main room, we can write this formula:

$$\Diamond\, coffee\_at(main\_room) \wedge (\neg coffee\_at(main\_room)\, \mathsf{U}\, coffee\_at(lobby\_room))$$

This formula means that eventually at some point of the time, the coffee will be served at the main room and that the coffee cannot be served at the main room until it has been served at the lobby room. This is a way to express that the coffee is served at the lobby room before it is served at the main room. Finite Linear Temporal Logics

is very useful because it is a more expressive language than Propositional Logic and it is a very natural language to express goals that are based on describing paths.

1.1.4. Planning via Translation using Non deterministic finite automata

In Computer Science, problems are classified by their required difficulty to solve them: How much computational time is required to solve a problem? How much memory is it needed? If I have an algorithm that solves one problem, can this algorithm be used to solve another problem? This is what we call Computational Complexity in an intuitive way. The different levels of difficulty are called Classes of Complexity.

One of the most well known classes of complexity is PSPACE. Intuitively, a problem is in PSPACE if the amount of memory required to solve an instance of length n is polynomial on the size of n. A problem is PSPACE-complete if it is in PSPACE and all other problems in PSPACE can be *reduced* to this one. A reduction is a translation of *instances* of a problem to instances of another problem that takes *polynomial time* on the size of the instance. In other words, if an algorithm to solve instances of the PSPACE-complete problem is known, it is possible to solve any problem in PSPACE by using the algorithm that translates the instance of this problem to the instance of another problem that is PSPACE-complete, and use the known algorithm to solve the translated one.

Bylander showed that the problem of finding a plan for Classical Planning is PSPACE-complete ((Bylander, 1994)). In another related work, De Giacomo and Vardi showed that finding a plan for Classical Planning with Temporally extended goals is also PSPACE-complete ((De Giacomo & Vardi, 1999)). Since nowadays there are very well known planners for Classical Planning with propositional goals, there's the intuition that there must be a polynomial-time translation from instances of Classical Planning with temporally extended goal to Classical Planning with propositional goals.

The authors (Baier & McIlraith, 2006) showed a translation using Non Deterministic Finite Automata (NFA), where the Automata is used to validate the temporal formula and the translated planning problem simulates the behavior of the automata by using additional *facts* or fluents in the planning problem. The drawback with this approach is that this translation has a worst-case exponential time for some formulae like in this following example:

$$\Diamond p_1 \wedge \Diamond p_2 \wedge \ldots \wedge \Diamond p_n$$

With formulae that generate a worst-case exponential time like this, they lead to very poor performance on the planning task, since the generated NFA has at least 2^n states. We need another approach that can deal with *any* temporal formula and avoid the worst-case exponential time.

1.2. Contributions of this thesis

This thesis presents a new translation technique whose input is a Deterministic Planning problem with Temporally extended goals and whose output is a classical planning problem. This technique uses another kind of automata that deals with the worst-case exponential time: *Alternating automata*. We show that our translation is correct with formal mathematical proofs and we show that this translation is polynomial-time on the size of the original planning problem. We also show how we can optimize this translation for better performance.

We also compare our approach with Baier and McIlraith's translation and we show the pros and cons of both techniques. We show that our approach can handle very well the temporal formulae that Baier and McIlraith's approach, but this one can still perform very well with simpler and easier classes of formulae.

1.2.1. Major Contributions

In this thesis, the major contributions are:

- We describe our translation approach that uses alternating automata.
- We prove that out translation approach is sound and complete.
- We prove that this translation is done in Polynomial time on the size of the original planning problem.
- We present optimized variants of our translation that perform better in practice (i.e. FastForward planner).
- We empirically show that our approach (and its variants) performs much better than Baier and McIlraith's translation for some classes of formulae.

1.2.2. Outline

In the following sections of this thesis, we will formalize the notion of Finite LTL and Alternating Automata. We will show the details of our translation and prove its soundness and correctness. The next section will have variants of our approach and ways to optimize the translation. Next, we show our experimental result and compare our approach with the state of the art and the final section shows the conclusions of this thesis.

2. ARTICLE SUBMITTED TO JOURNAL OF ARTIFICIAL INTELLIGENCE RESEARCH

2.1. Introduction

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

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

State-of-the-art compilation approaches to planning for LTL goals exploit the relationship between LTL and finite-state automata (FSA) (Edelkamp, 2006; Baier & McIlraith, 2006). As a result, the size of the output is worst-case *exponential* in the size of the LTL goal. Since deciding plan existence for both LTL and classical goals is PSPACE-complete (Bylander, 1994; De Giacomo & Vardi, 1999), none of these

approaches is optimal with respect to computational complexity, since they rely on a potentially exponential compilation. From a practical perspective, this worst case is also problematic since the size of a planning instance has a direct influence on planning runtime.

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

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

Most of the material included in this paper appeared in a previous conference publication (Torres & Baier, 2015). This paper extends the previous ones with

(i) a detailed description of two translation modes that have an important impact in performance (OSA and PG),

(ii) a section (Section 2.5) describing a simple but significant optimization to the translation presented previously.

In the rest of the paper, we outline the required background, we describe our AA construction for finite LTL logic, and then show the details of our compilation

approaches, including different modes and optimizations. We continue describing the details of our empirical evaluation. We finish with conclusions.

2.2. Preliminaries

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

2.2.1. Propositional Logic Preliminaries

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

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

2.2.2. Deterministic Classical Planning

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

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

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

$$\rho(s,a) = (s \setminus \{p \mid C \rightarrow \neg p \in \mathit{eff}(a), s \models C\}) \cup$$
$$\{p \mid C \rightarrow p \in \mathit{eff}(a), s \models C\}$$

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

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

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

2.2.3. Alternating Automata

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

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

The formal definition for AA that we use henceforth follows.

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

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

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

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

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

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

2.2.4. Finite LTL

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

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

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

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

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

(i) $\sigma, i \models p$ *iff* $s_i \models p$, *when* $p \in \mathcal{P}$.
(ii) $\sigma, i \models \neg\varphi$ *iff* $\sigma, i \not\models \varphi$
(iii) $\sigma, i \models \psi \wedge \chi$ *iff* $\sigma, i \models \psi$ *and* $\sigma, i \models \chi$
(iv) $\sigma, i \models \psi \vee \chi$ *iff* $\sigma, i \models \psi$ *or* $\sigma, i \models \chi$
(v) $\sigma, i \models \bigcirc\psi$ *iff* $i < n$ *and* $\sigma, (i+1) \models \psi$
(vi) $\sigma, i \models \bullet\psi$ *iff* $i = n$ *or* $\sigma, (i+1) \models \psi$
(vii) $\sigma, i \models \psi \,\mathsf{U}\, \chi$ *iff there exists* $k \in \{i, ..., n\}$ *such that* $\sigma, k \models \chi$ *and for each* $j \in \{i, ..., k-1\}$, *it holds that* $\sigma, j \models \psi$
(viii) $\sigma, i \models \psi \,\mathsf{R}\, \chi$ *iff for each* $k \in \{i, ..., n\}$ *it holds that* $\sigma, k \models \chi$ *or there exists a* $j \in \{i, ..., k-1\}$ *such that* $\sigma, j \models \psi$

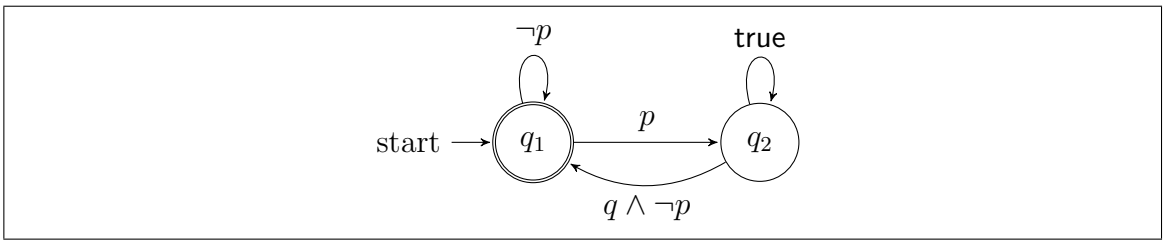

FIGURE 2.1. An NFA for formula $\Box(p \rightarrow \bigcirc\Diamond q)$ that expresses the fact that every time p becomes true in a state, then q has to be true in the state after or in the future. The input to the automaton is a (finite) sequence $s_0 \ldots s_n$ of planning states.

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

2.2.5. Deterministic Planning with LTL goals

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

There are two approaches to compiling away LTL via non-deterministic finite-state automata (Edelkamp et al., 2006; Baier & McIlraith, 2006). We describe both of them in detail below, since both are relevant to our work.

2.2.5.1. B&M's Compilation

B&M's approach compiles away LTL formulas exploiting the fact that for every finite LTL formula φ it is possible to build an NFA that accepts the finite models of φ. To illustrate this, Figure 2.1 shows an NFA for $\Box(p \rightarrow \bigcirc\Diamond q)$. B&M represent the NFA within the planning domain using one fluent per automaton state. In the example of Figure 2.1, this means that the new planning problem contains fluents E_{q_1} and E_{q_2}. The translation is such that if α is a sequence of actions that induces the execution trace $\sigma = s_1 \ldots s_n$, then E_q is true in s_n iff there is some run of the automaton over σ that ends in state q.

Assume a planning instance $P = \langle F, O, I, G\rangle$ in which $p \in I$ and $q \notin I$, and in which actions A_p, $A_{\neg p}$, and A_q, respectively, make p true, p false and q true unconditionally. Assume further that we want to compile away the temporally extended goal G which corresponds to the formula of Figure 2.1. Then the B&M translation generates a problem $P' = \langle F', O', I', G'\rangle$ such that $F' = F \cup \{E_{q_1}, E_{q_2}\}$. The initial state I' is equal to $I \cup \{E_{q_2}\}$; this is because p is true in I. Actions A_p, $A_{\neg p}$, and A_q appear in O' with the all the effects they have in O but with additional ones for representing the updates to E_{q_1} and E_{q_2}. Specifically, $\{E_{q_1}\} \rightarrow E_{q_2}$, and $\{E_{q_1}\} \rightarrow \neg E_{q_1}$ are all additional conditional affects added to A_p. The new effect of $A_{\neg p}$ is $\{q \wedge E_{q_2}\} \rightarrow E_{q_1}$, and the new effect of A_q is $\{\neg p \wedge E_{q_2}\} \rightarrow E_{q_1}$. Finally, $G' = \{E_{q_1}\}$.

Now consider the sequence of actions $\alpha = A_{\neg p}A_q$. In P, α induces an execution trace $\sigma = s_0 s_1 s_2$. The NFA of Figure 2.1 has two runs on σ, namely $\rho_1 = q_1 q_2 q_2$ and $\rho_2 = q_1 q_2 q_1$. Reflecting the fact that there exist two runs, one that ends in q_2 and one that ends in q_1, α induces an execution trace $s'_0 s'_1 s'_2$ on P' such that $s'_2 \models E_{q_2}$ and $s'_2 \models E_{q_1}$.

B&M's translation has the following property, from which soundness and completeness results follow.

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

(i) *There exists a run ρ of A_φ over σ ending in q.*

(ii) *E_q is true in the last state of σ'.*

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

Unfortunately, B&M's translation is worst-case exponential (Baier, 2010); for example, an NFA for $\wedge_{i=1}^{n} \Diamond p_i$ has 2^n states. Baier (Baier, 2010) proposes a formula-partitioning technique that allows the method to generate more compact translations for certain formulas. For example, for the case of $\wedge_{i=1}^{n} \Diamond p_i$ it is possible to build a single automaton for each $\Diamond p_i$ formula, and check satisfaction by conjoining the acceptance condition of the n resulting automata. The method, however, is not applicable to any formula whose NFA would blow up.

2.2.5.2. Edelkamp's Compilation

Edelkamp's approach compilation approach is also automata-based, and like B&M, it takes a planning instance P with an LTL goal as input and generates a planning instance P' that only contains final-state goals. It is briefly described in a short conference paper (Edelkamp, 2006). It relies on building a Büchi automaton (BA)—rather than an NFA—for the LTL goal, which is then interpreted as if it were an NFA. This last step is necessary because the BA are defined to accept infinite sequences of states, rather than the finite sequences produced by classical plans.

Like in B&M's approach, the automaton is encoded in P' by representing the automaton states with fluents. The update of the automaton is handled, however, in a very different way, by using a technique called *synchronized update*. The main idea of synchronized update is that specific actions are used to update the state of the automaton.

Our approach borrows the idea of synchronized update from Edelkamp's compilation but differs from it significantly because we need to update an AA rather than a BA. Indeed, Edelkamp's translation, like B&M's, is able to represent all the possible runs of the automaton in the same planning state, which, as we argue in Section 2.4, does not seem applicable to the case of translating AA.

It is important to remark that the use of BA interpreted as NFA does not yield a correct translation for any finite LTL, therefore Edelkamp's compilation is not correct for general LTL (De Giacomo, Masellis, & Montali, 2014). It is correct, however, for the PDDL3 subset of LTL, for which it was originally proposed.

2.3. Alternating Automata and Finite LTL

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

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

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

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

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

Proof: Suppose that $\sigma = x_1 x_2 \ldots x_n \in \Sigma^*$, where $\Sigma = 2^S$. The proof of the theorem is straightforward from the following lemma: φ: $\sigma, i \models \varphi$ if and only if there exists a sequence $r = Q_{i-1} Q_i \ldots Q_n$, such that: (1) $Q_{i-1} = \{q_\varphi\}$, (2) $Q_n \subseteq \{q_F\}$, (3) For each subset Q_j in the sequence r it holds that $Q_j \subseteq sub(\varphi) \cup \{q_F\}$ and (4) For each $j \in \{i, i+1, \ldots, n\}$ it holds that $Q_j \models \delta(Q_{j-1}, x_j)$. The proof for the lemma follows. It is inductive on the construction of φ.

$\Rightarrow$) Suppose that $\sigma, i \models \varphi$. To prove this direction, it suffices to provide a sequence $r = Q_{i-1} Q_i \ldots Q_n$ satisfying the aforementioned properties. Below we show each sequence. We do not show that they satisfy the four properties; we leave this as an exercise to the reader.

- $\varphi = \ell$, for any literal ℓ. Then $r = (\{q_\ell\}, \emptyset, \ldots, \emptyset)$.

Suppose that the lemma holds for any φ with less than m operators and that for any α and β with less than m operators, their respective sequences are $r' = Q'_{i-1}Q'_iQ''_{i+1}\dots Q'_n$ and $r'' = Q''_{i-1}Q''_iQ''_{i+1}\dots Q''_n$. Now, let φ be a formula with m operators:

- $\varphi = \alpha \vee \beta$. Then $\sigma, i \models \alpha$ or $\sigma, i \models \beta$. Without loss of generality, suppose that $\sigma, i \models \alpha$. Then $r = (\{q_\varphi\}, Q'_i, Q'_{i+1}, \dots, Q'_n)$.
- $\varphi = \alpha \wedge \beta$. Then $r = (\{q_\varphi\}, (Q'_i \cup Q''_i), (Q'_{i+1} \cup Q''_{i+1}), \dots, (Q'_n \cup Q''_n))$.
- $\varphi = \bigcirc\alpha$. Then $\sigma, (i+1) \models \alpha$. In this case, the sequence for α is $r' = Q'_iQ''_{i+1}\dots Q'_n$.
  With this, the sequence r for $\bigcirc\alpha$ is $r = (\{q_\varphi\}, \{q_\alpha\}, Q''_{i+1}, \dots, Q'_n)$.
- $\varphi = \bullet\alpha$. Then $i = n$ or $\sigma, (i+1) \models \alpha$. If $i = n$, the sequence $r = (Q_{n-1}, Q_n) = (\{q_\varphi\}, \{q_F\})$. If $i < n$, consider the same sequence r for the case $\bigcirc\alpha$.
- $\varphi = \alpha \,\mathsf{U}\, \beta$. Then, there exists $k \geq i$ such that $\sigma, k \models \beta$ and for every $j \in \{i, \dots, k-1\}$ it holds that $\sigma, j \models \alpha$. For β, assume its sequence is $r_k = (Q^k_{k-1}, Q^k_k, \dots, Q^k_n)$ and for each α that is satisfied by σ, j, assume its sequence is $r_j = (Q^j_{j-1}, Q^j_j, \dots, Q^j_n)$. The sequence $r = Q_{i-1}Q_i \dots Q_n$ is given by:

$$Q_j = \begin{cases} \{q_{\alpha \,\mathsf{U}\, \beta}\}, \text{ if } j = i-1 \\ \{q_{\alpha \,\mathsf{U}\, \beta}\} \cup \bigcup_{x=i}^{j} Q^x_j, \text{ if } i-1 < j < k \\ \bigcup_{x=i}^{k} Q^x_j, \text{ if } j \geq k \end{cases}$$

- $\varphi = \alpha \,\mathsf{R}\, \beta$. Then, for each $k \in \{i, \dots, n\}$ it holds that $\sigma, k \models \beta$ or there exists a $j \in \{i, \dots, k-1\}$ such that $\sigma, j \models \alpha$. If there is no such j, then $\sigma, k \models \beta$ for every $k \in \{i, \dots, n\}$ and for each one of them, assume their sequence will correspond to $r_k = (Q^k_{k-1}, Q^k_k, \dots, Q^k_n)$. The sequence

$r = Q_{i-1}Q_i \dots Q_n$ is given by:

$$Q_k = \begin{cases} \{q_{\alpha \mathsf{R} \beta}\}, \text{ if } k = i-1 \\ \{q_{\alpha \mathsf{R} \beta}\} \cup \bigcup_{x=i}^{k} Q_k^x, \text{ if } i-1 < k < n \\ \{q_F\}, \text{ if } k = n \end{cases}$$

If there is a $j \in \{i, \dots, k-1\}$ such that $\sigma, j \models \alpha$, consider the minimum such j and assume its sequence is $r' = (A_{j-1}, A_j, \dots, A_n)$. For $k \in \{i, \dots, j\}$, the sequences for β will be $r_k = (B_{k-1}^k, B_k^k, \dots, B_n^k)$. The sequence $r = Q_{i-1}Q_i \dots Q_n$ is given by:

$$Q_k = \begin{cases} \{q_{\alpha \mathsf{R} \beta}\}, \text{ if } k = i-1 \\ \{q_{\alpha \mathsf{R} \beta}\} \cup \bigcup_{x=i}^{k} B_k^x, \text{ if } i-1 < k < j \\ A_k \cup \bigcup_{x=i}^{j} B_k^x, \text{ if } k \geq j \end{cases}$$

$\Leftarrow$) Suppose that there exists a sequence $r = Q_{i-1}Q_i \dots Q_n$ for φ that satisfies the four properties. To prove that $\sigma, i \models \varphi$, it should be straightforward for $\varphi = \ell$. For the inductive steps, where α is a direct subformula of φ, the sequence r must be used to create a new sequence r' for α (ensuring that r' satisfies the four properties) and use the implication of $\sigma, i \models \alpha$. This finishes the proof for the theorem. ■

2.4. Compiling Away finite LTL Properties

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

First, we argue that the idea underlying both Edelkamp's and B&M's translations would *not* yield an efficient translation if applied to AA. Recall in both approaches if $E_{q_1}, \dots, E_{q_n}$ are true in a planning state s, then there are n runs of the automaton, each of which ends in $q_1, \dots, q_n$ (Theorem 1). In other words, the planning state keeps track of *all* of the runs of the automaton. To apply the same principle to AA, we would need to introduce one fluent for each *subset* of states

of the AA, therefore generating a number of fluents exponential on the size of the original formula. This is because runs of AA are sequences of *sets* of states, so we would require states of the form E_R, where R is a set of states.

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

2.4.1. Translating LTL via LTL Synchronization

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

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

Given a planning problem $P = \langle F, O, I, G \rangle$, our translation generates a problem P' in which there is one (new) fluent q for each state q of the AA A_G. The compilation is such that the following property holds: if $\alpha = a_1 a_2 \dots a_n$ is applicable in the initial state of P, then there exists a set $\mathcal{A}_\alpha$ of action sequences of the form

$\alpha_0 a_1 \alpha_1 a_2 \alpha_2 \ldots a_n \alpha_n$, where each α_i is a sequence of synchronization actions whose sole objective is to update the fluents representing A_G's state.

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

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

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

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

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

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

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

TABLE 2.1. The synchronization actions for LTL goal G in NNF. Above ℓ, $\alpha \,\mathsf{R}\, \beta$, $\alpha \,\mathsf{U}\, \beta$, and $\bigcirc\alpha$ are assumed to be in the set of subformulas of G. In addition, ℓ is assumed to be a literal.

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

during synchronization, each q^S defines the state of the automaton prior to synchronization. The precondition of *copy* is simply $\{\mathbf{copy}, \mathbf{ok}\}$, while its effect is defined by:

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

As soon as the **sync** fluent becomes true, the second phase of synchronization begins. Here the only executable actions are those that update the state of the automaton, which are defined in Table 2.1. Note that one of the actions deletes the **ok** fluent. This can happen, for example while synchronizing a formula that actually expresses the fact that the action sequence has to conclude now.

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

The set O_s is defined as the one containing actions *copy*, *world*, and all actions defined in Table 2.1.

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

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

2.4.2. Properties

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

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

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

- $\varphi = \ell$. Assume ℓ is positive literal. Then there is a planning state s in which $trans(q^S_\ell)$ was executed. Since the precondition requires $\ell \in s$ and ℓ can only be added by an action from O_w, then $\ell \in t$. By definition,

$\delta(q_\ell, t) = \top$, and it is clear that $\rho(t, \alpha_i) \models \delta(q_\varphi, t)$. The argument is analogous for a negative literal ℓ.

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

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

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

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

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

Proof: Note that T is a state in *world* mode getting ready to go into *synchronization* mode after the *copy* action has been executed. The main idea of the proof is to choose the order of the subformulae to be synchronized, where the first one corresponds to the largest subformula of the current state, the second one corresponds to the second largest subformula and so on. Note that when an action $trans(q^S_\alpha)$ is executed, it

always happens that at most two fluents q^S_β and q^S_γ are added, and the formulae β and γ are strict subformulae of α. This means that a subformula will never get synchronized twice in a single synchronization phase σ. Since the number of subformulae is linear on $|G|$, this means that the length of σ must be $O(|G|)$. ■

2.4.3. Towards More Efficient Translations

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

2.4.3.1. An Order for Synchronization Actions (OSA)

Consider the goal formula is $\alpha \wedge \beta$ and that currently both q_α and q_β are true. The planner has two equivalent ways of completing the synchronization: by executing first $trans(q_\alpha)$ and then $trans(q_\beta)$, or by inverting such a sequence. By enforcing an order between these synchronizations, we can reduce the branching factor of the synchronization phase.

The synchronization order is simple to enforce by modifying preconditions and effects of synchronization actions so that states are synchronized following a topological order of the parse tree of G. To formalize this, consider the alternating automata $A_G = (Q, \Sigma, \delta, q_G, \{q_F\})$ for the temporal formula G and the following sets:

- $Q^S = \{q^S \mid q \in Q\}$ the synchonizing states of Q.
- $Q^T = \{q^T \mid q \in Q\}$ the token states of Q.
- $F^S = \{\mathbf{copy}, \mathbf{sync}, \mathbf{world}, \mathbf{ok}\}$ the special fluents.

Fluents from Q^T are used as *tokens*, to indicate the state that must be checked for synchronization. As such, if $q^T \in Q^T$ is true, then state q must be checked for synchronization. Consider the function $succ : Q^T \to Q^T$ where $succ(q^T)$ is the successor state of q^T in the topological ordering. Also, consider $first(Q^T)$ and $last(Q^T)$ respectively as the first and last state of Q^T according to the topological ordering. Note that $succ(last(Q^T))$ is never defined. Actually, as a convention, the state q_F is always considered as $last(Q^T)$.

P' is built as follows:

- $F' = F \cup Q \cup Q^S \cup Q^T \cup \{\mathbf{copy}, \mathbf{sync}, \mathbf{world}, \mathbf{ok}\}$
- $I' = I \cup \{q_G, \mathbf{copy}, \mathbf{ok}\}$
- $G' = \{\mathbf{world}, \mathbf{ok}\} \cup \overline{(Q \setminus \{q_F\})}$
- $O' = O_s \cup O_w$

The actions from $O_w = \{a' \mid a \in O\}$ are the same as before. For all a' in O_w:

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

The precondition of the *copy* action, like before, is $\{\mathbf{copy}, \mathbf{ok}\}$, the effect, however, is different:

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

Note that in this mode, the action *copy* "activates" the first token to trigger the synchronization in topological order. Table 2.2 shows the synchronization for each subformula in O_s. Note that the action $trans(q_F^S)$ immediately returns to world mode, since q_F is the last state of the topological ordering. Unlike the first table that shows the synchronizing actions, the effects here are conditional. If the respective synchronizing fluent q^S is true during action $trans(q^S)$, then its effects are applied.

2.4.3.2. Positive Goals (PG)

The goal condition of the translated instance requires being in and every $q \in Q$ to be false, except for q_F. On the other hand, action *copy*, which has to be performed after each world action, has precisely the effect of making every $q \in Q$ false. This may significantly hurt performance if search relies on heuristics that relax negative effects of actions, like the FF heuristic (Hoffmann & Nebel, 2001), which is key to the performance of state-of-the-art planning systems (see e.g., (Richter & Helmert, 2009)).

TABLE 2.2. The synchronization actions (OSA) for LTL goal G in NNF. Used to enforce topological ordering.

Sync Action	Precondition	Effect
$trans(q^S_\ell)$	$\{\mathbf{sync}, \mathbf{ok}, q^T_\ell\}$	$\{\neg q^T_\ell, succ(q^T_\ell), q^S_\ell \rightarrow \neg q^S_\ell, (q^S_\ell \wedge \ell) \rightarrow \neg\mathbf{ok}\}$
$trans(q^S_F)$	$\{\mathbf{sync}, \mathbf{ok}, q^T_F\}$	$\{\neg q^T_F, q^S_F \rightarrow \neg q^S_F, q^S_F \rightarrow \neg\mathbf{ok}, \mathbf{world}, \neg\mathbf{sync}\} \cup$ $\{q^S \rightarrow q, \neg q^S \mid q^S \in Q^S \wedge q \in Q\}$
$trans(q^S_{\alpha\wedge\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^T_{\alpha\wedge\beta}\}$	$\{\neg q^T_{\alpha\wedge\beta}, succ(q^T_{\alpha\wedge\beta}), q^S_{\alpha\wedge\beta} \rightarrow \{q^S_\alpha, q^S_\beta, \neg q^S_{\alpha\wedge\beta}\}\}$
$trans_1(q^S_{\alpha\vee\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^T_{\alpha\vee\beta}\}$	$\{\neg q^T_{\alpha\vee\beta}, succ(q^T_{\alpha\vee\beta}), q^S_{\alpha\vee\beta} \rightarrow \{q^S_\alpha, \neg q^S_{\alpha\vee\beta}\}\}$
$trans_2(q^S_{\alpha\vee\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^T_{\alpha\vee\beta}\}$	$\{\neg q^T_{\alpha\vee\beta}, succ(q^T_{\alpha\vee\beta}), q^S_{\alpha\vee\beta} \rightarrow \{q^S_\beta, \neg q^S_{\alpha\vee\beta}\}\}$
$trans(q^S_{\bigcirc\alpha})$	$\{\mathbf{sync}, \mathbf{ok}, q^T_{\bigcirc\alpha}\}$	$\{\neg q^T_{\bigcirc\alpha}, succ(q^T_{\bigcirc\alpha}), q^S_{\bigcirc\alpha} \rightarrow \{q_\alpha, \neg q^S_{\bigcirc\alpha}\}\}$
$trans_1(q^S_{\bullet\alpha})$	$\{\mathbf{sync}, \mathbf{ok}, q^T_{\bullet\alpha}\}$	$\{\neg q^T_{\bullet\alpha}, succ(q^T_{\bullet\alpha}), q^S_{\bullet\alpha} \rightarrow \{q_F, \neg q^S_{\bullet\alpha}\}\}$
$trans_2(q^S_{\bullet\alpha})$	$\{\mathbf{sync}, \mathbf{ok}, q^T_{\bullet\alpha}\}$	$\{\neg q^T_{\bullet\alpha}, succ(q^T_{\bullet\alpha}), q^S_{\bullet\alpha} \rightarrow \{q_\alpha, \neg q^S_{\bullet\alpha}\}\}$
$trans_1(q^S_{\alpha\,\mathsf{U}\,\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^T_{\alpha\,\mathsf{U}\,\beta}\}$	$\{\neg q^T_{\alpha\,\mathsf{U}\,\beta}, succ(q^T_{\alpha\,\mathsf{U}\,\beta}), q^S_{\alpha\,\mathsf{U}\,\beta} \rightarrow \{q^S_\beta, \neg q^S_{\alpha\,\mathsf{U}\,\beta}\}\}$
$trans_2(q^S_{\alpha\,\mathsf{U}\,\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^T_{\alpha\,\mathsf{U}\,\beta}\}$	$\{\neg q^T_{\alpha\,\mathsf{U}\,\beta}, succ(q^T_{\alpha\,\mathsf{U}\,\beta}), q^S_{\alpha\,\mathsf{U}\,\beta} \rightarrow \{q^S_\alpha, q_{\alpha\,\mathsf{U}\,\beta}, \neg q^S_{\alpha\,\mathsf{U}\,\beta}\}\}$
$trans_1(q^S_{\alpha\,\mathsf{R}\,\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^T_{\alpha\,\mathsf{R}\,\beta}\}$	$\{\neg q^T_{\alpha\,\mathsf{R}\,\beta}, succ(q^T_{\alpha\,\mathsf{R}\,\beta}), q^S_{\alpha\,\mathsf{R}\,\beta} \rightarrow \{q^S_\beta, q_F, \neg q^S_{\alpha\,\mathsf{R}\,\beta}\}\}$
$trans_2(q^S_{\alpha\,\mathsf{R}\,\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^T_{\alpha\,\mathsf{R}\,\beta}\}$	$\{\neg q^T_{\alpha\,\mathsf{R}\,\beta}, succ(q^T_{\alpha\,\mathsf{R}\,\beta}), q^S_{\alpha\,\mathsf{R}\,\beta} \rightarrow \{q^S_\beta, q^S_\alpha, \neg q^S_{\alpha\,\mathsf{R}\,\beta}\}\}$
$trans_3(q^S_{\alpha\,\mathsf{R}\,\beta})$	$\{\mathbf{sync}, \mathbf{ok}, q^T_{\alpha\,\mathsf{R}\,\beta}\}$	$\{\neg q^T_{\alpha\,\mathsf{R}\,\beta}, succ(q^T_{\alpha\,\mathsf{R}\,\beta}), q^S_{\alpha\,\mathsf{R}\,\beta} \rightarrow \{q^S_\beta, q_{\alpha\,\mathsf{R}\,\beta}, \neg q^S_{\alpha\,\mathsf{R}\,\beta}\}\}$

To improve heuristic guidance, we define a new goal condition which does not require or use literals that are deleted by the *copy* action. For this, we define a new set of fluents:

$$Q^D \;=\; \{q^D \mid q \in Q\}$$

Fluent q^D will represent that state q is *totally synchronized*. We say that a state q_α is totally synchronized if and only if formula α is totally synchronized. In turn, a formula α will be considered as *totally synchronized* if once its respective action $trans(q^S_\alpha)$ is executed, it can be guaranteed that $trans(q^S_\alpha)$ cannot be executed in the future. In other words, a state q_α is totally synchronized in the execution if there is no executable action *trans* that adds q_α as a positive effect.

For example, suppose that $\Diamond(p \wedge \bigcirc q) = G$ is the objective goal of the planning problem. According to Table 2.1, where $\Diamond(p \wedge \bigcirc q)$ is equivalent to $\top\,\mathsf{U}(p \wedge \bigcirc q))$, this formula has two *trans* actions. If $trans_1(q^S_G)$—which intuitively happens when $p \wedge \bigcirc q$ has become true at some point of the plan's execution—, then the formula G

will never need to be synchronized ever again. Nevertheless, if $trans_2(q^S_G)$ is executed then G will need to be synchronized again in the next iteration.

An important observation is that if α is a subformula of the goal formula, its synchronization depends on its superformulae, because the superformulae split into their subformulae whenever an action *trans* is executed over them. In our example, the subformula $\bigcirc q$ of G, can become totally synchronized only if its superformulae are totally synchronized. This is because if G or other superformula is synchronized again, it will eventually split itself into $\bigcirc q$.

To define the dynamics of the fluents in Q^D we only need to add extra effects to the *trans* actions. Specifically, for each $trans(q^S_\alpha)$ action that does not add q_α, we include the conditional effect $\{q^D_\beta \mid \beta \in super(\alpha)\} \rightarrow q^D_\alpha$, where $super(\alpha)$ is the set of subformulae of G that are proper superformulae of α. If $super(\alpha)$, which is the case of $G = \alpha$, then the effect is unconditional and q^D_α is added right away.

Now we define the new goal. This is essential to provide guidance to the heuristic. Recall that in the previous translation the objective was have no states of the form q_α in the goal state. Observe that in this new translation, every time a state of the form q^S_α is deleted from the state, its corresponding q^D_α is added. The new goal now establishes that a particular set of subformulae of the goal formula G are synchronized.

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

In case the goal is a literal or $\varphi = \bigcirc\beta$, it is straightforward than the formula and their subformulae are totally synchronized when then the goal is achieved. For

$\varphi = \alpha \wedge \beta$, both subformulae must be totally synchronized, but in the case of $\varphi = \alpha \vee \beta$, only one of them needs to be, because it is enough to satisfy one of them. The case for $\varphi = \bullet\beta$ is also a choice: β must be totally synchronized or q_F must be the final state.

The definition for the case $\varphi = \alpha \star \beta$, where $\star \in \{\mathsf{U}, \mathsf{R}\}$, is less obvious. For the U operator, a necessary condition to satisfy the formula $\alpha \,\mathsf{U}\, \beta$ is that β must be eventually satisfied. However, α doesn't need to be totally synchronized, because if so then it would force α to be satisfied at the same point when β is satisfied. If in the current point in time, β is not satisfied, the *trans* actions will ensure that α is synchronized accordingly, this is due to the lemma for the Correctness Theorem. The sequence of *trans* actions will correctly simulate the behavior of the Alternating Automata to satisfy α. A similar argument goes for the R operator, we only care to totally synchronize β and α is not needed. If the planning stops in a state where we know that β was totally synchronized, then we know that this state must be final (q_F).

2.5. Optimizing the Translation

Recall that each resulting plan in the translated planning problem is of the form $\alpha_0 a_1 \alpha_1 a_2 \alpha_2 \ldots a_n \alpha_n$, where α_i is a sequence of synchronization actions. Thus the size of each α_i is an important factor in the final length of the plan. Since search algorithms are ultimately used to find these plans, intuitively it seems important to aim at optimizing plan length; this is because the running time of the search algorithms depend on solution depth.

Table 2.1 shows the actions needed to synchronize a generic goal G. This table is instantiated when translating a specific formula. When looking at the instantiated table it is not hard to see that it is usually possible to reduce the number of synchronization actions. Table 2.3 shows the instantiation for goal $G = \Diamond(p \wedge (q \wedge \neg r))$. Note there is a total of 7 actions, one for each subformulae. In addition, note that to

TABLE 2.3. The synchronization actions for LTL goal $G = \Diamond(p \wedge (q \wedge \neg r))$.

Sync Action	Precondition	Effect
$trans(q_p^S)$	$\{\mathbf{sync}, \mathbf{ok}, q_p^S, p\}$	$\{\neg q_p^S\}$
$trans(q_q^S)$	$\{\mathbf{sync}, \mathbf{ok}, q_q^S, q\}$	$\{\neg q_q^S\}$
$trans(q_{\neg r}^S)$	$\{\mathbf{sync}, \mathbf{ok}, q_{\neg r}^S, \neg r\}$	$\{\neg q_{\neg r}^S\}$
$trans(q_{p\wedge(q\wedge\neg r)}^S)$	$\{\mathbf{sync}, \mathbf{ok}, q_{p\wedge(q\wedge\neg r)}^S\}$	$\{q_p^S, q_{q\wedge\neg r}^S, \neg q_{p\wedge(q\wedge\neg r)}^S\}$
$trans(q_{q\wedge\neg r}^S)$	$\{\mathbf{sync}, \mathbf{ok}, q_{q\wedge\neg r}^S\}$	$\{q_q^S, q_{\neg r}^S, \neg q_{q\wedge\neg r}^S\}$
$trans_1(q_{\Diamond(p\wedge(q\wedge\neg r))}^S)$	$\{\mathbf{sync}, \mathbf{ok}, q_{\Diamond(p\wedge(q\wedge\neg r))}^S\}$	$\{q_{(p\wedge(q\wedge\neg r))}^S, \neg q_{\Diamond(p\wedge(q\wedge\neg r))}^S\}$
$trans_2(q_{\Diamond(p\wedge(q\wedge\neg r))}^S)$	$\{\mathbf{sync}, \mathbf{ok}, q_{\Diamond(p\wedge(q\wedge\neg r))}^S\}$	$\{q_{\Diamond(p\wedge(q\wedge\neg r))}, \neg q_{\Diamond(p\wedge(q\wedge\neg r))}^S\}$

TABLE 2.4. The reduced synchronization actions for LTL goal $G = \Diamond(p \wedge (q \wedge \neg r))$.

Sync Action	Precondition	Effect
$trans_1(q_{\Diamond(p\wedge(q\wedge\neg r))}^S)$	$\{\mathbf{sync}, \mathbf{ok}, q_{\Diamond(p\wedge(q\wedge\neg r))}^S, p, q, \neg r\}$	$\{\neg q_{\Diamond(p\wedge(q\wedge\neg r))}^S\}$
$trans_2(q_{\Diamond(p\wedge(q\wedge\neg r))}^S)$	$\{\mathbf{sync}, \mathbf{ok}, q_{\Diamond(p\wedge(q\wedge\neg r))}^S\}$	$\{q_{\Diamond(p\wedge(q\wedge\neg r))}, \neg q_{\Diamond(p\wedge(q\wedge\neg r))}^S\}$

finish the synchronization of G there are two options: (1) to verify that the current planning state satisfies the literals p, q, and $\neg r$ or (2) delay this verification for a future state. Intuitively this suggests that only 2 actions are actually needed. Indeed, Table 2.4 shows one way of synchronizing the same goal with only two actions.

The objective of this section is to propose a method to obtain these simplifications automatically. As we see in Section 2.6, these simplifications do have a positive impact on search performance.

2.5.1. Synchronizing action graphs (SAG)

The previous example suggests that we need to *merge* actions when possible. Indeed, the actions that synchronize $(p \wedge (q \wedge \neg r))$, $(q \wedge \neg r)$, q, p, and $\neg r$ do not require to be different: it can all be done with a single action. This merge process indeed corresponds to generating a single *macro action* (e.g., (Botea, Enzenberger, Müller, & Schaeffer, 2005)) from a set of simple actions.

To implement these simplifications, we use a data structure over which we define two operations: *merge* and *split*. Intuitively a merge is an operation that creates

macro actions, while a split is an operation that will facilitate the creation of a macro action. Below a formal definition of our graphs.

DEFINITION 7 (Synchronizing Action Graph). *Given a LTL formula φ, a* synchronizing action graph *(SAG) is a directed graph represented by a tuple $G = (V, E, F, P, S, L)$ where:*

- *V is the set of nodes*
- *$F : V \rightarrow sub(\varphi)$ is a function that associates a subformula of φ with each node $u \in V$.*
- *$P : V \rightarrow 2^{Lit(Var(\varphi))}$ is a function that indicates which set of literals will be assigned to each node u. Here $Var(\varphi)$ denotes the set of propositional variables in φ. Intuitively P corresponds to the precondition of the action that this node will represent respectively.*
- *$S : V \rightarrow 2^{sub(\varphi)}$ is a function that indicates which synchronizing state fluents are added when the respective action represented by this node is executed.*
- *$L : V \rightarrow 2^{sub(\varphi)}$ is a function that indicates which automata state fluents are added when the respective action represented by this node is executed.*
- *$E \subseteq V \times V$ is the set of edges of the graph, where $(u, v) \in E$ if and only if $F(v) \in S(u)$.*

Intuitively, each node of the graph corresponds to a synchronizing action *trans*. There is an arc between a node u and a node v if as a result of executing the action that corresponds to u, the action that corresponds to v can now execute.

2.5.2. Building a SAG

Given an LTL formula φ in NNF we carry out two steps prior to building a SAG. First carry out a standard conversion of subsequent applications of the binary Boolean operator $\wedge$ (respectively, $\vee$) into a single, multiary $\wedge$ (respectively, $\vee$).

Then, we modify the *trans* actions of Table 2.1, for the conjunction and disjunction, in the following way:

- If $\varphi = \bigvee_{i=1}^{n} \alpha_i$, then there are n actions $trans_i(q_\varphi^S)$ $(i \in \{1, \ldots, n\})$, with precondition $\{\mathbf{sync}, \mathbf{ok}, q_\varphi^S\}$ and effect $\{\neg q_\varphi^S, q_{\alpha_i}^S\}$.
- If $\varphi = \bigwedge_{i=1}^{n} \alpha_i$, then action $trans(q_\varphi^S)$ has precondition $\{\mathbf{sync}, \mathbf{ok}, q_\varphi^S\}$ and effect $\{\neg q_\varphi^S\} \cup \{q_{\alpha_i}^S \mid i \in \{1, \ldots, n\}\}$.

To create an SAG from φ, for each ψ in the set of subformulae of φ we perform the two following steps:

(i) If ψ is a literal ℓ, create a node u with the following attributes:
- $F(u) = \ell$
- $P(u) = \{\ell\}$
- $S(u) = L(u) = \emptyset$

(ii) Otherwise, if ψ is not a literal φ, then we refer to Table 2.1, and define $\mathit{eff}(\varphi)$ as the effect of action $trans(q_\varphi^S)$. Then:
- $F(u) = \varphi$
- $P(u) = \emptyset$
- $S(u) = \{\chi \mid q_\chi^S \in \mathit{eff}(\psi)\}$
- $L(u) = \{\chi \mid q_\chi \in \mathit{eff}(\psi)\}$

As a last step, we create edges between two nodes according to its definition; i.e., we add an edge between u and v if $F(v) \in S(u)$.

Note that building a SAG this way guarantees that the directed graph will be acyclic.

2.5.2.1. An Example SAG

Figure 2.2 shows the SAG for $\varphi = \Box(\neg p \vee \Diamond q)$. A node is built for every *single* synchronizing action needed for the translation. For example, consider the subformula $\Diamond q$. Since there are two possible actions to synchronize this subformula,

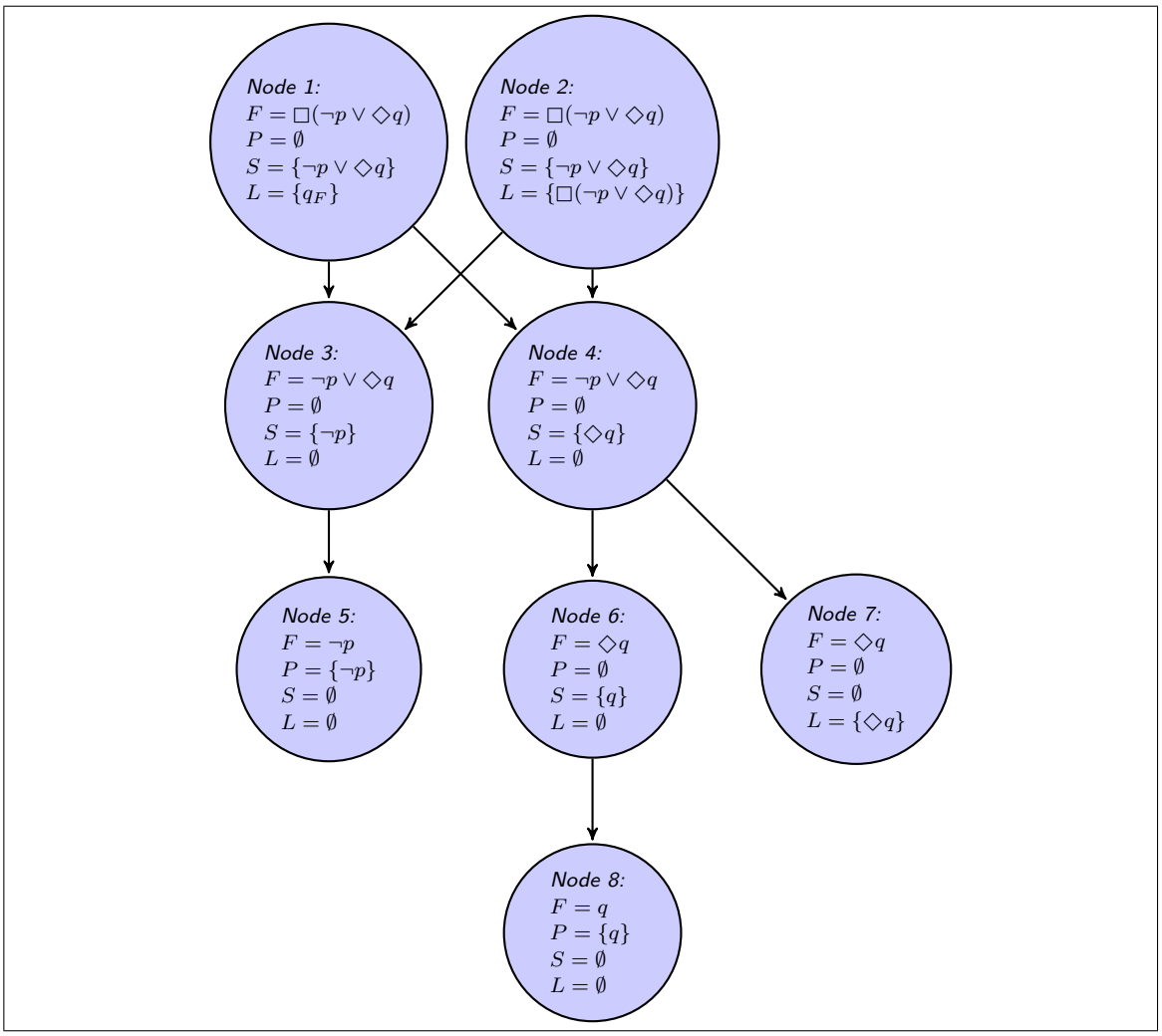

FIGURE 2.2. A SAG for the LTL formula $\varphi = \Box(\neg p \vee \Diamond q)$

then two nodes are created. The edges here represent a *dependency* between the involved nodes. To illustrate this, let V denote *Node 6* in the figure. Since the action represented by V cannot be executed until the action represented by *Node 4* is executed there is an arc between *Node 4* and *Node 6*.

2.5.3. Operations on a SAG

The two operations that allows us to generate a simplified set of synchronizing actions are *merge* and *split*. Now we define formally what these operations do.

2.5.3.1. Merge

As we mentioned above, intuitively merge builds a macro action from two nodes in the graph. In our example of Figure 2.2, it is not hard to see that the actions for Nodes 6 and 8 can be converted into a macro action that would have q in its precondition and $S = L = \emptyset$. The merge operation applies to two connected nodes but it is not always applicable. Consider for example Nodes 1 and 3. Although they are connected, it would not be correct to eliminate Node 3 because there is an associated dependency between Nodes 2 and 3.

We say that a merge operation between nodes u and v is applicable if the following two conditions hold:

(i) u is a predecessor of v

(ii) either $F(v)$ is a literal or v is a sink node and the only successor of u

Note that this definition implies that a node corresponding to a literal can always be merged even if it has multiple predecessors.

The if a merge is applicable to nodes u and v, then a new node u' is created on the SAG, and the following operations are carried out:

(i) $F(u') = F(u)$

(ii) $P(u') = P(u) \cup P(v)$

(iii) $S(u') = S(u) \setminus F(v)$

(iv) $L(u') = L(u) \cup L(v)$

(v) The edge (u, v) is deleted from the SAG.

(vi) Replace u by u'.

Note that v is not eliminated from the SAG, since it could be needed for other merging operations.

Continuing with our example, Figure 2.3 shows the result of execution all applicable merges to the SAG of Figure 2.2. In particular, a merge between Nodes 6 and 8 has been performed, resulting in the addition of Node 7.1. Note that the

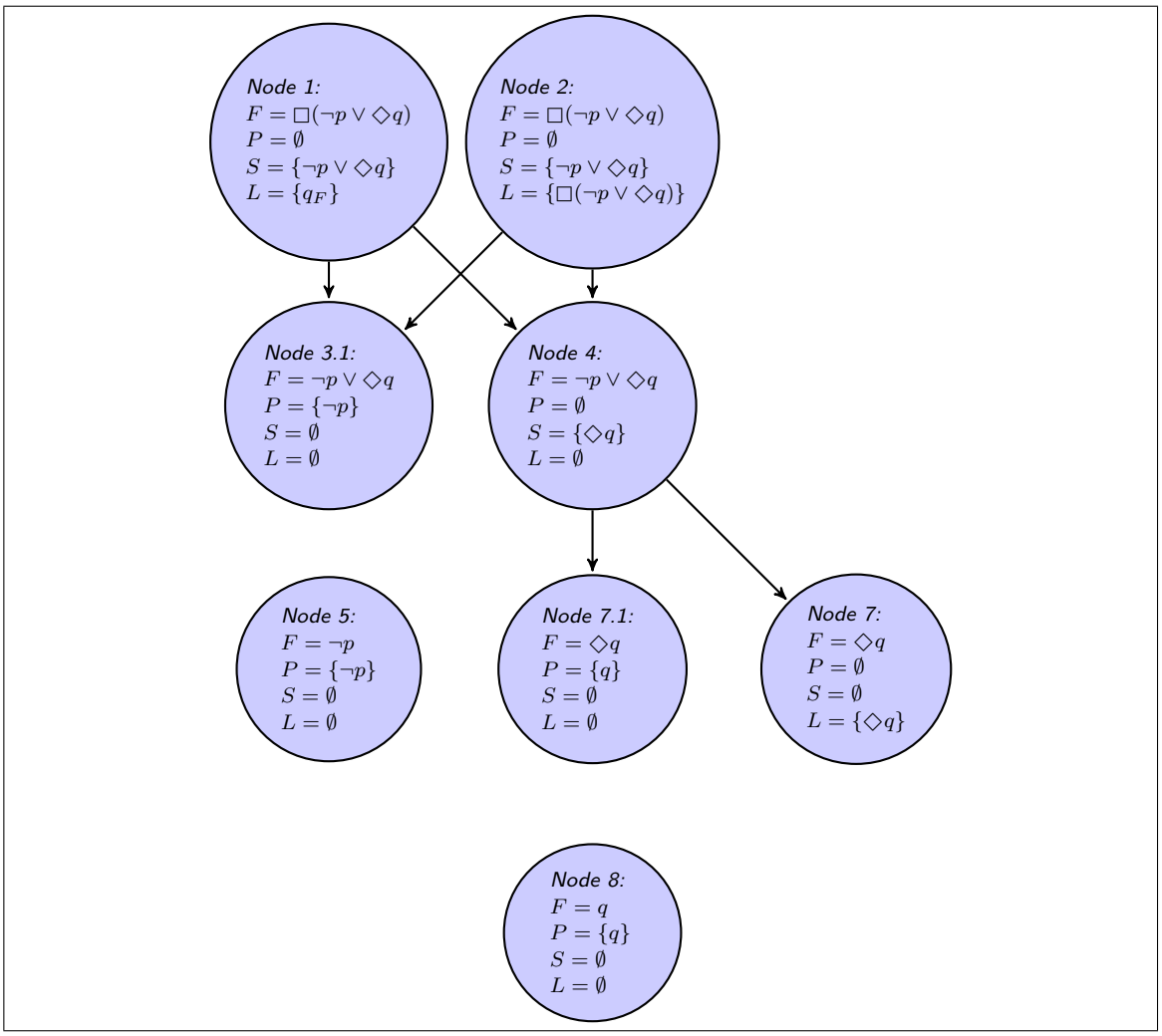

FIGURE 2.3. A SAG for the LTL formula $\varphi = \Box(\neg p \vee \Diamond q)$, after merging literals

precondition for Node 8 is "transferred" to Node 7.1. Node 8 is not removed from the SAG. Thus, if Node 8 had had another incoming edge the merge would still allow the action for Node 8 synchronize other subformulae.

2.5.3.2. Split

Splitting consists of copying a node with n successors n times, assigning each of its successors to a single copy. A split is only applied to nodes that have the potential of facilitating a subsequent merge operation because our ultimate objective is to reduce the final number of synchronization actions. To see why a split may become

necessary, observe Node 4 of Figure 2.3. None of its successors can be merged into Node 4 because it has two successors.

A split of a node u is applicable if the following conditions hold:

(i) the top-level Boolean operator in $F(u)$ is neither an $\wedge$ nor an R,

(ii) u has at least one successor, and

(iii) every successor of u is a sink.

The first condition may not seem obvious at first sight, but it is desirable. Indeed, if the top-level operation for a node u is $\wedge$ or R, then it is likely that $S(u)$ contains more than one formula (cf. Table 2.1). Since we need to maintain the property of an edge in a SAG—that is, $(u, v) \in E$ if and only if $F(v) \in S(u)$—by splitting such a node we would require the addition of more nodes and edges, which is counter-productive. Seeing why the second condition is necessary is straightforward. A split operation does not make sense if the node u is a sink, because it will not allow a subsequent merge. The third condition is necessary to avoid breaking the internal structure of a SAG when the node u is duplicated (dependencies).

Applying a split over a node u whose set of successors is $\{v_1, \ldots, v_n\}$ results in the generation of a set of n new nodes $\{u_1, \ldots, u_n\}$, such that:

- $F(u_i) = F(u)$, $P(u_i) = P(u)$, $S(u_i) = S(u)$, and $L(u_i) = L(u)$, and
- a new arc (u_i, v_i) is added to the SAG,

for every $i \in \{1, \ldots, n\}$. In addition, u is removed from the SAG.

Figure 2.4 shows the application of a split operation over Node 4 followed by a merge operation. The resulting SAG is shown in Figure 2.5.

2.5.3.3. The Simplification Algorithm

The simplification algorithm for a SAG is straightforward and consists of the following steps:

(i) While there are two nodes u and v for which a merge applicable, apply a merge between u and v.

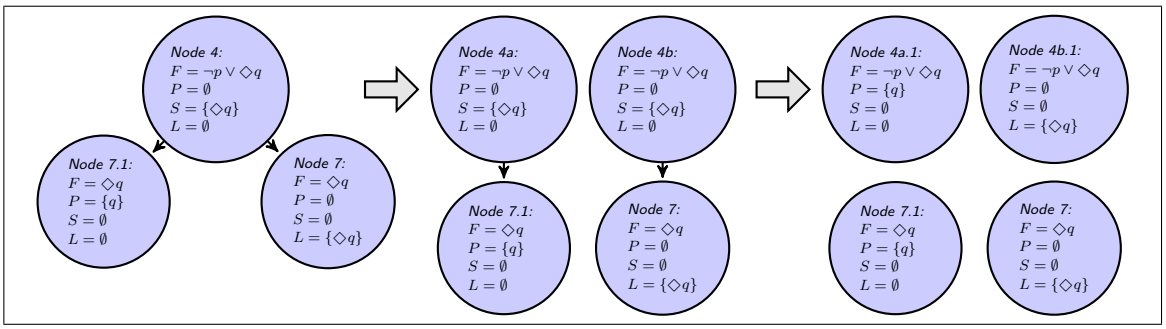

FIGURE 2.4. An example for a splitting operation. A step of duplication and a step of merging.

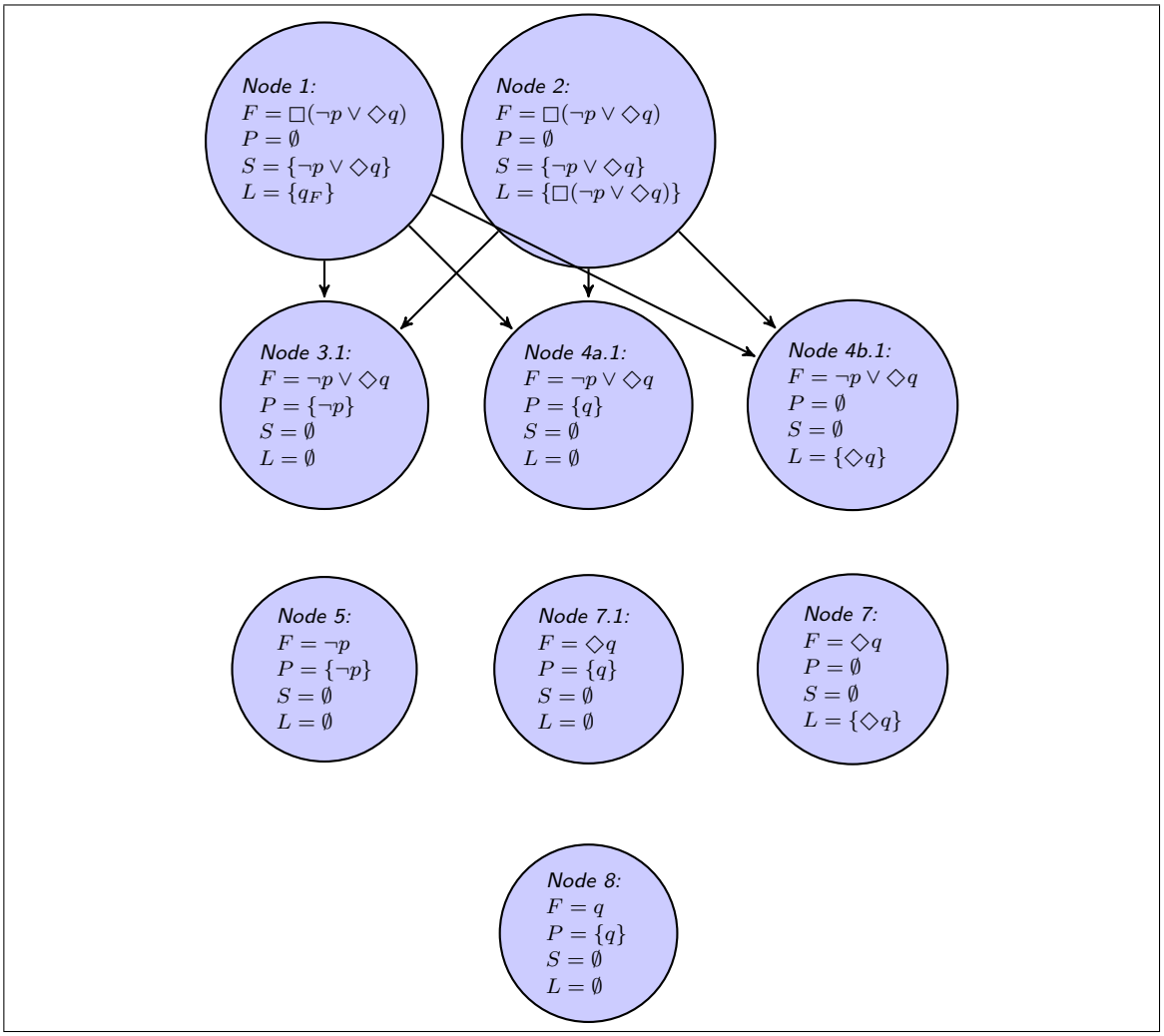

FIGURE 2.5. A SAG for the LTL formula $\varphi = \Box(\neg p \vee \Diamond q)$, after merging literals and splitting

(ii) If u is a node that can be split, apply the split operation to u.

(iii) Stop if after executing both previous steps no operation has been applied. Otherwise go back to Step 1.

Once the algorithm has finished its execution, further simplification is carried out, eliminating some nodes in the graph. Specifically, we remove three types of nodes:

- Nodes that lead to dead ends. These nodes require the plan to finish in a certain state, but at the same time require a temporal formula to be checked in the future of such a state same. Specifically, u leads to a dead end iff $L(u)$ contains two elements or more, among which it is q_F.
- Nodes with inconsistent preconditions. The merge process could generate a node u such that both a literal and its complement are in $P(u)$. We say such nodes have inconsistent preconditions.
- Unreachable nodes. Intuitively these nodes could be generated by a merge between nodes u and v, which does not eliminate v. Reachable nodes can be inductively defined as follows.

  (i) If u is such that $F(u) = \varphi$, where φ is the LTL goal formula, then u is reachable.

  (ii) If u is reachable and (u, v) is an arc in the SAG, then v is reachable.

  (iii) Finally, if u is reachable and node v is such that $F(v) \in L(u)$, then v is reachable.

  Note that the last condition may declare a node v as reachable from u even though (u, v) is not an arc in the graph. This is actually correct because when $F(v) \in L(u)$ the action represented by u is enabling the execution of v in the next round of synchronization.

Figure 2.6 shows SAG of our example after node removal.

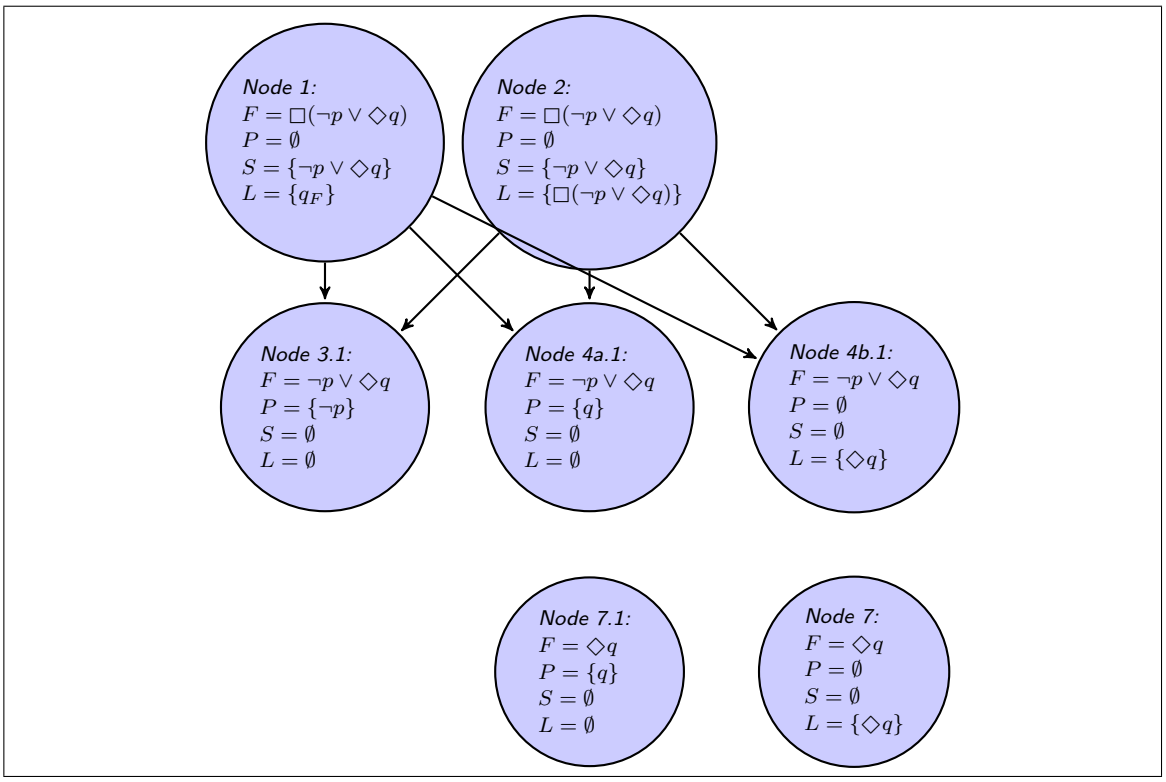

FIGURE 2.6. A SAG for the LTL formula $\varphi = \Box(\neg p \vee \Diamond q)$, after merging literals, splitting and removing nodes

2.5.4. Extracting Actions from a SAG

Obtaining synchronization actions from a SAG is straightforward. For each node u we generate action A_u such that:

$$prec(A_u) = \{\mathbf{sync}, \mathbf{ok}, q^S_{F(u)}\} \cup P(u),$$
$$\mathit{eff}(A_u) = \{\neg q^S_{F(u)}\} \cup \{q^S_\alpha \mid \alpha \in S(u)\} \cup \{q_\alpha \mid \alpha \in L(u)\}.$$

We can also incorporate the ideas in the OSA mode (Section 2.4.3.1) here too. Recall that in that mode there we computed a topological order of the parse tree of the goal formula, and that the set Q_T represented token states. Here, we compute a topological order of the graph, and if u and v are successors in the topological order we say define $succ(q^T_{F(u)}) = q^T_{F(v)}$. Following the same idea presented earlier, the precondition and effect of every action A_u associated to node u is:

$$
\begin{aligned}
prec(A_u) &= \{\mathbf{sync}, \mathbf{ok}, q^T_{F(u)}\} \\
\mathit{eff}(A_u) &= \{\neg q^T_{F(u)}, \Gamma \rightarrow \neg q^S_{F(u)}, succ(q^T_{F(u)})\} \cup \{\Gamma \rightarrow q^S_\alpha \mid \alpha \in S(u)\} \cup \\
&\quad \{\Gamma \rightarrow q_\alpha \mid \alpha \in L(u)\},
\end{aligned}
$$

where $\Gamma = \{q^S_{F(u)}\} \cup P(u)$.

The main difference between this version and that of Section 2.4.3.1 is that before q_F was always the last state of the topological order. Here, instead, the node for q_F might have been merged into another node. If $succ(q^T_{F(u)})$ is not defined (because $F(u)$ is the last subformula in the topological order), then $succ(q^T_{F(u)})$ will be replaced by **endturn**, a new special fluent. We define a new action *endturn*, which is similar to the previous **world** action), its precondition and effect will be:

$$
\begin{aligned}
prec(endturn) &= \{\mathbf{sync}, \mathbf{ok}, \mathbf{endturn}\} \\
eff(endturn) &= \overline{Q^S} \cup \{\neg\mathbf{sync}, \neg\mathbf{endturn}, \mathbf{world}\} \cup \{q^S \rightarrow q \mid q \in Q\}
\end{aligned}
$$

It is also possible to use the PG described in Section 2.4.3.2. For each node u and its respective action A_u, if $F(u) \notin L(u)$ then we add $eff(a)$:

$$\big(\{q^D_\beta \mid \beta \in super(F(u))\} \cup P(u)\big) \rightarrow q^D_{F(u)},$$

where $super(F(u))$ is the set of superformulae that are *still* in the SAG.

2.6. Empirical Evaluation

The objective of our evaluation was to compare our approach with existing translation approaches, over a range of general LTL goals, to understand when it is convenient to use one or other approach. We chose to compare to B&M's rather than Edelkamp's because the former seems to yield better performance (Baier, Bacchus, & McIlraith, 2009). We do not compare against other existing systems that handle

PDDL3 natively, such as LPRPG-P (Coles & Coles, 2011), because efficient translations for the (restricted) subset of LTL of PDDL3 into NFA are known (Gerevini et al., 2009).

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

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

Our translator was implemented in SWI-Prolog. It takes a domain and a problem in PDDL with an LTL goal as input and generates PDDL domain and problem files. It also receives an additional parameter specifying the translation mode which can be any of the following: *simple*, *OSA*, *PG*, and *OSA+PG*, where *simple* is the translation of Section 2.4, and *OSA*, *PG* are the optimizations described in Section 2.4.3. *OSA+PG* is the combination of *OSA* and *PG*.

Table 2.5 shows a representative selection of the results we obtained. It shows translation time (TT), plan length (PL), planning time (PT), the number of planning states that were evaluated before the goal was reached (PS). Times are displayed in seconds. For our translators we also include the length of the plan without synchronization actions (WPL). NR means the planner/translator did not return a plan. For each problem, a special name of the form $x0n$ was assigned, where x corresponds to a specific family of formula and n it's parameter (i.e. For the problem $a04$, the goal formula $\alpha \cup \bigwedge_{i=1}^{n} \beta_i$ was used, with $n = 4$).

Each family of formulae corresponds to: a: $\alpha \cup \bigwedge_{i=1}^{n} \beta_i$, b: $\bigvee_{i=1}^{n} \Diamond p_i \,\mathsf{U}\, \Diamond q$, c: $\Diamond(\alpha \wedge \bigwedge_{i=1}^{n} \Diamond \beta_i)$, d: $\bigwedge_{i=1}^{n}(\alpha \,\mathsf{U}\, \beta_i)$, e: $\Diamond(\bigwedge_{i=1}^{n} \Diamond \beta_i)$, f: $\bigwedge_{i=1}^{n} \Diamond p_i$, g: $\bigwedge_{i=1}^{3} \Diamond p_i \wedge \bigwedge_{i=1}^{n-3} q_i \,\mathsf{U}\, r_i$, i: $\Diamond(\bigvee_{i=1}^{n} \Diamond \beta_i)$, j: $\Diamond(\bigvee_{i=1}^{n} \Box \beta_i)$ and k: $\bigvee_{i=1}^{n}(\alpha \,\mathsf{U}(p_i \wedge q_i))$.

We observe mixed results. B&M yields superior results on some problems; e.g., $f03$ and $f05$ of *openstacks* (of the form $\bigwedge_{i=1}^{n} \Diamond p_i$). The performance gap is probably due to the fact that (1) the B&M problem requires fewer actions in the plan and (2) B&M's output for these goals is quite compact on the size of the formula. On the other hand, there are other goal formulas in which our approach outperforms B&M. For example, problems of the form $a0n$ in openstacks and blocksworld, and of the form $b0n$ in blocksworld. In those cases, the B&M translator is forced to generate the whole automaton, because it has to deal with nested subformulae in which the distributive property does not hold. As a consequence, B&M generates an output exponential in n, which results in higher translation time and eventually in the the planner running out of memory.

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

For the case of the class formula k, we observe that the first translator handles this formula pretty well, since the $\wedge$ operator can distribute over the U operator. We see that when n is increased, the time needed to translate the planning problem is linear. This also happens with our approach. This shows that it is still possible to use the old approach to handle some classes of formulae.

The use of positive goals has an important influence in performance possibly because the heuristic is more accurate, leading to fewer expansions. OSA, on the other hand, seems to negatively affect planning performance in $\mathrm{FF}_{\mathcal{X}}$. The reason

is the following: $\text{FF}_{\mathcal{X}}$ will frequently choose the wrong synchronization action and therefore its enforced hill climbing algorithm will often fail. This behavior may not be observed in planners that use complete search algorithms.

For the experiments with SAG, we did not consider using OSA, since the results in Table 2.5 show that using OSA usually doesn't impact too much the performance of the planner or worsens it, because it needs to evaluate more planning states. If SAG is used to optimize the translation, we see that for most scenarios the time needed to translate a problem has a slight overhead. This is expected due to the extra operations that are added in this process (graph construction and optimization). We also see that in many cases, the planning is decreased, due to the number of actions that were reduced during the translation using SAG. This is also seen in the number of planning states that were evaluated when SAG is used. Looking at Table 2.6, we see that if the formula used is simple, then B&M outperforms our approach. However, if B&M is not capable to translate the problem, due to the exponential blow up, then our approach (PG + NonOSA + SAG) performs better than our other variants in most cases.

2.7. Conclusions

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

An interesting observation is our approach is not limited to goal formulae. For example, it can be adapted for the case of LTL preferences (Baier, Bacchus, & McIlraith, 2007). Indeed, since in this case LTL preferences would be reduced to simple non-temporal subgoals, approaches like that of Keyder and Geffner (Keyder & Geffner, 2009) would be applicable to reduce the problem to one solvable by standard cost-optimizing deterministic planner.

Finally, it is possible to adapt our approach to other languages for which there exists an alternating automaton. Indeed, (Triantafillou, Baier, & McIlraith, 2015) have already adapted the approach presented in this paper for the case of planning with goals represented in Linear Dynamic Logic (De Giacomo & Vardi, 2013). Our approach has also the potential to be applied in scenarios in which actions are non-deterministic and thus has applicability to the well-known LTL synthesis problem (De Giacomo & Vardi, 2015).

TABLE 2.5. Experimental results for a variety of LTL planning tasks.

Openstacks Domain

	B&M's translator			Non-PG + Non-OSA			Non-PG + OSA			PG + Non-OSA			PG + OSA		
	TT	PL	PS	TT	PL	WPL	PT	PS	TT	PL	WPL	PT	PS	TT	PL	WPL	PT	PS	TT	PL	WPL	PT	PS
a03	0.373	23	0.10	34	0.463	136	21	6.44	222245	0.486	309	21	10.86	203461	0.475	167	23	0.05	1413	0.501	319	22	0.25	3421
a04	1.594	0	NR	NR	0.470	156	22	22.49	592081	0.504	392	22	24.04	417585	0.496	192	24	0.10	2763	0.527	405	23	0.52	6573
a05	21.852	0	NR	NR	0.482	179	23	103.30	1573433	0.523	481	23	54.07	872446	0.525	213	24	0.23	5906	0.564	497	24	1.17	13074
e03	0.377	23	0.10	34	0.459	117	21	9.08	294097	0.481	287	21	10.57	202873	0.469	167	23	0.05	1413	0.491	297	22	0.23	3217
e04	1.599	0	NR	NR	0.472	125	22	31.93	755539	0.498	369	22	23.41	418240	0.489	192	24	0.10	2763	0.517	382	23	0.48	6176
e05	22.390	0	NR	NR	0.478	133	23	149.88	1958261	0.518	457	23	53.27	876816	0.513	213	24	0.22	5906	0.548	473	24	1.05	12292
f03	0.246	23	0.02	34	0.455	142	21	5.41	196938	0.474	265	21	8.78	179157	0.462	166	23	0.05	1412	0.483	274	22	0.21	3100
f05	0.268	25	0.02	36	0.477	199	23	78.71	1364638	0.503	433	23	48.62	834783	0.504	212	24	0.22	5905	0.536	448	24	1.01	12172
g02	0.256	24	0.10	214	0.466	166	21	20.15	533753	0.491	331	21	18.44	341998	0.484	197	22	0.41	10439	0.509	342	22	0.48	6498
g03	0.266	24	0.13	224	0.474	215	22	138.11	1892750	0.510	415	22	35.84	635715	0.506	231	22	1.53	35117	0.542	410	22	1.07	13364
Rovers Domain																								
e03	0.407	10	0.05	16	0.481	62	10	68.39	1173227	0.507	144	10	46.04	544034	0.498	67	10	0.02	453	0.517	140	10	0.02	448
e04	1.639	0	NR	NR	0.494	0	0	NR	NR	0.539	1	0	NR	NR	0.517	106	15	0.05	1324	0.545	252	15	0.07	1201
f01	0.242	4	0.00	5	0.460	25	4	0.03	1757	0.471	31	4	0.04	1353	0.462	22	4	0.00	64	0.469	28	4	0.00	62
f02	0.255	7	0.00	10	0.468	45	7	0.55	25490	0.487	73	7	1.84	44002	0.475	42	7	0.01	316	0.489	69	7	0.01	181
f03	0.270	10	0.01	15	0.481	68	10	7.67	264568	0.501	133	10	41.78	522432	0.490	66	10	0.02	452	0.505	129	10	0.03	425
i04	0.299	3	0.01	4	0.499	19	2	0.04	1522	0.528	49	2	0.04	847	0.525	17	3	0.00	43	0.548	47	3	0.02	301
j04	0.301	3	0.01	4	0.501	19	2	0.03	1290	0.537	49	2	0.05	962	0.522	26	5	0.01	148	0.554	78	5	0.20	3494
Blocksworld Domain																								
a03	1.627	2	0.02	3	0.448	22	2	0.12	4115	0.472	46	2	0.04	792	0.464	19	2	0.00	32	0.490	41	2	0.01	146
a04	22.220	0	NR	NR	0.458	25	2	1.94	45113	0.492	55	2	0.31	4692	0.486	22	2	0.00	50	0.523	50	2	0.01	228
a05	471.574	0	NR	NR	0.471	28	2	38.99	474906	0.514	64	2	2.52	24761	0.522	25	2	0.01	77	0.559	59	2	0.04	330
b03	9.801	1	0.00	2	0.446	11	1	0.00	88	0.473	31	1	0.01	122	0.464	8	1	0.00	12	0.494	25	1	0.00	85
b04	423.327	1	0.00	2	0.463	11	1	0.01	172	0.494	37	1	0.02	248	0.490	8	1	0.00	16	0.523	31	1	0.01	147
b05	NR	NR	NR	NR	0.473	11	1	0.02	285	0.519	43	1	0.05	492	0.527	8	1	0.00	22	0.566	37	1	0.02	269
c03	0.383	2	0.58	4	0.443	24	2	0.09	3607	0.470	46	2	0.01	357	0.465	22	2	0.00	101	0.489	41	2	0.01	97
c04	1.809	0	NR	NR	0.456	27	2	1.80	44666	0.490	55	2	0.06	1067	0.487	25	2	0.00	194	0.519	50	2	0.02	162
c05	25.655	0	NR	NR	0.470	30	2	48.30	610049	0.509	64	2	0.31	3325	0.520	28	2	0.04	514	0.558	59	2	0.04	287
d03	0.234	2	0.00	3	0.452	24	2	0.11	3451	0.481	49	2	0.07	1612	0.470	21	2	0.00	34	0.502	43	2	0.00	106
d04	0.243	2	0.01	3	0.467	28	2	2.66	48436	0.508	61	2	1.08	14118	0.513	25	2	0.00	53	0.550	55	2	0.02	172
d05	0.251	2	0.01	3	0.486	32	2	84.40	637178	0.535	73	2	17.99	104398	0.566	29	2	0.01	81	0.609	67	2	0.04	256
e03	3.813	0	NR	NR	0.457	25	2	0.16	5680	0.489	52	2	0.09	1700	0.483	22	2	0.00	35	0.517	46	2	0.00	150
e04	182.181	0	NR	NR	0.476	29	2	6.16	122233	0.517	64	2	1.15	14576	0.532	26	2	0.01	54	0.567	58	2	0.03	257
e05	NR	NR	NR	NR	0.495	0	0	NR	NR	0.547	76	2	19.13	106729	0.584	30	2	0.01	82	0.638	70	2	0.06	396
k03	0.299	4	0.02	8	0.475	27	3	1.21	26811	0.515	85	3	4.21	40628	0.527	1	0	0.00	10	0.568	82	4	0.14	1226
k04	0.326	4	0.04	8	0.509	27	3	1.73	35170	0.553	105	3	6.25	55223	0.599	1	0	0.00	10	0.662	102	4	0.33	2549
k05	0.353	4	0.06	8	0.520	27	3	2.08	38853	0.592	125	3	8.96	72670	0.707	1	0	0.00	10	0.773	122	4	0.64	4665

TABLE 2.6. Comparison table of experimental results for a variety of LTL planning tasks using and not using SAG.

	B&M's translator				Non-PG + Non-OSA + Non-SAG				PG + Non-OSA + Non-SAG				Non-PG + Non-OSA + SAG				PG + Non-OSA + SAG			
	TT	PL	PT	PS	TT	PL	WPL	PT	PS	TT	PL	WPL	PT	PS	TT	PL	WPL	PT	PS	TT	PL	WPL	PT	PS
Openstacks Domain																								
a03	0.373	23	0.10	**34**	0.463	136	21	6.44	222245	0.475	167	23	0.05	1413	0.544	112	22	1.39	29530	0.552	134	23	0.04	747
a04	1.594	0	NR	NR	0.470	156	22	22.49	592081	0.496	192	24	0.10	2763	0.554	121	22	2.68	47929	0.561	164	24	0.08	**1460**
a05	21.852	0	NR	NR	0.482	179	23	103.30	1573433	0.525	213	24	0.23	5906	0.565	141	23	4.66	86136	0.575	190	25	0.14	**2653**
e03	0.377	23	0.10	**34**	0.459	117	21	9.08	294097	0.469	167	23	0.05	1413	0.545	92	21	5.45	39496	0.550	134	23	0.04	747
e04	1.599	0	NR	NR	0.472	125	22	31.93	755539	0.489	192	24	0.10	2763	0.565	101	22	5.98	42650	0.557	164	24	0.07	**1460**
e05	22.390	0	NR	NR	0.478	133	23	149.88	1958261	0.513	213	24	0.22	5906	0.556	113	23	12.19	46068	0.566	190	25	0.14	**2653**
f03	0.246	23	0.02	**34**	0.455	142	21	5.41	196938	0.462	166	23	0.05	1412	0.541	125	22	0.56	14465	0.541	133	23	0.04	745
f05	0.268	25	0.02	**36**	0.477	199	23	78.71	1364638	0.504	212	24	0.22	5905	0.548	168	23	2.02	43121	0.558	189	25	0.14	2651
g02	0.256	24	0.10	**214**	0.466	166	21	20.15	533753	0.484	197	22	0.41	10439	0.544	129	21	1.06	23480	0.549	148	23	0.06	1238
g03	0.266	24	0.13	**224**	0.474	215	22	138.11	1892750	0.506	231	22	1.53	35117	0.549	157	22	2.14	43955	0.560	169	23	0.14	2394
Rovers Domain																								
e03	0.407	10	0.05	**16**	0.481	62	10	68.39	1173227	0.498	67	10	0.02	453	0.572	0	0	NR	NR	0.577	64	12	0.02	511
e04	1.639	0	NR	NR	0.494	0	0	NR	NR	0.517	106	15	0.05	1324	0.581	0	0	NR	NR	0.584	100	17	0.04	**1190**
f01	0.242	4	0.00	**5**	0.460	25	4	0.03	1757	0.462	22	4	0.00	64	0.547	20	4	0.02	744	0.552	18	4	0.00	26
f02	0.255	7	0.00	**10**	0.468	45	7	0.55	25490	0.475	42	7	0.01	316	0.558	36	7	0.02	993	0.561	44	9	0.01	403
f03	0.270	10	0.01	**15**	0.481	68	10	7.67	264568	0.490	66	10	0.02	452	0.566	54	10	0.02	1001	0.569	63	12	0.02	509
i04	0.299	3	0.01	**4**	0.499	19	2	0.04	1522	0.525	17	3	0.00	43	0.559	13	2	0.00	80	0.566	11	2	0.00	24
j04	0.301	3	0.01	**4**	0.501	19	2	0.03	1290	0.522	26	5	0.01	148	0.611	17	2	0.02	981	0.628	0	0	NR	NR
Blocksworld Domain																								
a03	1.627	2	0.02	**3**	0.448	22	2	0.12	4115	0.464	19	2	0.00	32	0.535	16	2	0.00	70	0.535	14	2	0.00	30
a04	22.220	0	NR	NR	0.458	25	2	1.94	45113	0.486	22	2	0.00	50	0.537	17	2	0.32	7386	0.548	15	2	0.00	**34**
a05	471.574	0	NR	NR	0.471	28	2	38.99	474906	0.522	25	2	0.01	77	0.546	18	2	14.62	58202	0.558	16	2	0.00	**48**
b03	9.801	1	0.00	**2**	0.446	11	1	0.00	88	0.464	8	1	0.00	12	0.522	9	1	0.00	18	0.528	7	1	0.00	15
b04	423.327	1	0.00	**2**	0.463	11	1	0.01	172	0.490	8	1	0.00	16	0.527	9	1	0.00	28	0.535	7	1	0.00	19
b05	NR	NR	NR	NR	0.473	11	1	0.02	285	0.527	8	1	0.00	**22**	0.531	9	1	0.00	44	0.543	7	1	0.00	25
c03	0.383	2	0.58	**4**	0.443	24	2	0.09	3607	0.465	22	2	0.00	101	0.525	16	2	0.01	433	0.532	14	2	0.00	31
c04	1.809	0	NR	NR	0.456	27	2	1.80	44666	0.487	25	2	0.00	194	0.532	17	2	0.08	2768	0.541	15	2	0.00	**49**
c05	25.655	0	NR	NR	0.470	30	2	48.30	610049	0.520	28	2	0.04	514	0.542	18	2	1.09	21881	0.551	16	2	0.00	**74**
d03	0.234	2	0.00	**3**	0.452	24	2	0.11	3451	0.470	21	2	0.00	34	0.536	18	2	0.00	311	0.546	16	2	0.00	42
d04	0.243	2	0.01	**3**	0.467	28	2	2.66	48436	0.513	25	2	0.00	53	0.551	20	2	0.06	1255	0.564	18	2	0.00	57
d05	0.251	2	0.01	**3**	0.486	32	2	84.40	637178	0.566	29	2	0.01	81	0.567	22	2	0.40	5665	0.586	20	2	0.00	83
e03	3.813	0	NR	NR	0.457	25	2	0.16	5680	0.483	22	2	0.00	**35**	0.547	19	2	0.00	552	0.561	17	2	0.00	44
e04	182.181	0	NR	NR	0.476	29	2	6.16	122233	0.532	26	2	0.01	**54**	0.563	21	2	0.14	3534	0.584	19	2	0.00	59
e05	NR	NR	NR	NR	0.495	0	0	NR	NR	0.584	30	2	0.01	**82**	0.583	23	2	5.75	28464	0.610	21	2	0.01	85
k03	0.299	4	0.02	8	0.475	27	3	1.21	26811	0.527	1	0	0.00	10	0.552	20	3	0.20	4219	0.575	5	1	0.00	**6**
k04	0.326	4	0.04	8	0.509	27	3	1.73	35170	0.599	1	0	0.00	10	0.574	20	3	0.28	5256	0.605	5	1	0.00	**6**
k05	0.353	4	0.06	8	0.520	27	3	2.08	38853	0.707	1	0	0.00	10	0.594	20	3	0.35	6246	0.639	5	1	0.00	**6**

References

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

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

Baier, J. A. (2010). *Effective search techniques for non-classical planning via reformulation* (Ph.D. in Computer Science). University of Toronto.

Baier, J. A., Bacchus, F., & McIlraith, S. A. (2007, January). A heuristic search approach to planning with temporally extended preferences. In *Proceedings of the 20th International joint Conference on Artificial Intelligence* (p. 1808-1815). Hyderabad, India.

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

Baier, J. A., & McIlraith, S. A. (2006). Planning with first-order temporally extended goals using heuristic search. In *Proceedings of the 21st national Conference on Artificial Intelligence* (p. 788-795). Boston, MA.

Bonet, B., & Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, *129*(1-2), 5-33.

Botea, A., Enzenberger, M., Müller, M., & Schaeffer, J. (2005). Macro-FF: Improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research*, *24*, 581-621.

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

Coles, A. J., & Coles, A. (2011). LPRPG-P: relaxed plan heuristics for planning with preferences. In *Proceedings of the 21th International Conference on automated planning and Scheduling.*

Cresswell, S., & Coddington, A. M. (2004, August). Compilation of LTL goal formulas into PDDL. In R. L. de Mántaras & L. Saitta (Eds.), *Proceedings of the 16th european Conference on Artificial Intelligence* (p. 985-986). Valencia, Spain: IOS Press.

De Giacomo, G., & Vardi, M. Y. (2015). Synthesis for LTL and LDL on finite traces. In *Proceedings of the 24th International joint Conference on Artificial Intelligence* (pp. 1558–1564). Retrieved from `http://ijcai.org/papers15/Abstracts/IJCAI15-223.html`

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

De Giacomo, G., & Vardi, M. Y. (1999, September). Automata-theoretic approach to planning for temporally extended goals. In S. Biundo & M. Fox (Eds.), *Ecp* (Vol. 1809, p. 226-238). Durham, UK: Springer.

De Giacomo, G., & 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.* Beijing, China. Retrieved from `http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6997`

Edelkamp, S. (2006). On the compilation of plan constraints and preferences. In *Proceedings of the 16th International Conference on automated planning and scheduling.*

Edelkamp, S., Jabbar, S., & Naizih, M. (2006, July). Large-scale optimal PDDL3 planning with MIPS-XXL. In *5th International Planning Competition Booklet* (p. 28-30). Lake District, England.

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

Gerth, R., Peled, D., Vardi, M. Y., & Wolper, P. (1995, July). Simple on-the-fly automatic verification of linear temporal logic. In *Proceedings of the 15th International symposium on protocol specification, testing and verification* (p. 3-18). Warsaw, Poland.

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

Keyder, E., & Geffner, H. (2009). Soft goals can be compiled away. *Journal of Artificial Intelligence Research*, *36*, 547–556. Retrieved from `http://dx.doi.org/10.1613/jair.2857`

Muller, D. E., Saoudi, A., & Schupp, P. E. (1988). Weak Alternating Automata Give a Simple Explanation of Why Most Temporal and Dynamic Logics are Decidable in Exponential Time. In *Proceedings of the 3rd annual symposium on logic in computer science* (pp. 422–427).

Pnueli, A. (1977). The temporal logic of programs. In *Proceedings of the 18th ieee symposium on foundations of computer science* (pp. 46–57).

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

Richter, S., Helmert, M., & Westphal, M. (2008). Landmarks revisited. In *Proceedings of the 23rd aaai Conference on Artificial Intelligence* (p. 975-982). Chicago, IL.

Rintanen, J. (2000, August). Incorporation of temporal logic control into plan operators. In W. Horn (Ed.), *Proceedings of the 14th european Conference on Artificial Intelligence* (p. 526-530). Berlin, Germany: IOS Press.

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

Torres, J., & Baier, J. A. (2015). Polynomial-time reformulations of LTL temporally extended goals into final-state goals. In *Proceedings of the 24th International joint Conference on Artificial Intelligence* (pp. 1696–1703).

Triantafillou, E., Baier, J. A., & McIlraith, S. (2015). A unifying framework for planning with LTL and regular expressions. In *Icaps workshop on model-checking and automated planning.*