A Decision Procedure for a Fragment of Linear Time Mu-Calculus

Yao Liu, Zhenhua Duan* **and Cong Tian***
ICTT and ISN Laboratory, Xidian University
Xi'an, 710071, P. R. China
yao_liu@stu.xidian.edu.cn, {zhhduan,ctian}@mail.xidian.edu.cn

Abstract

In this paper, we study an expressive fragment, namely $\mathcal{G}_\mu$, of linear time μ-calculus as a high-level goal specification language. We define Goal Progression Form (GPF) for $\mathcal{G}_\mu$ formulas and show that every closed formula can be transformed into this form. Based on GPF, we present the notion of Goal Progression Form Graph (GPG) which can be used to describe models of a formula. Further, we propose a simple and intuitive GPG-based decision procedure for checking satisfiability of $\mathcal{G}_\mu$ formulas which has the same time complexity as the decision problem of Linear Temporal Logic (LTL). However, $\mathcal{G}_\mu$ is able to express a wider variety of temporal goals compared with LTL.

1 Introduction

Linear Temporal Logic (LTL) is a convenient formalism for specifying and verifying properties of reactive systems [Pnueli, 1977]. Also, due to its simplicity, it has been extensively used in planning for temporally extended goals [Bacchus and Kabanza, 1998; De Giacomo and Vardi, 1999; Calvanese *et al.*, 2002; Kabanza and Thiébaux, 2005; Baier and McIlraith, 2006; Patrizi *et al.*, 2011]. However, restricted by its expressiveness, LTL cannot be used to specify relatively complex goals. Therefore, extensions of LTL [De Giacomo and Vardi, 2013; 2015] have been studied in order to make it more powerful. Unfortunately, these extensions consider only finite LTL rather than standard LTL over infinite traces.

To this end, we investigate a fragment, called $\mathcal{G}_\mu$, of linear time μ-calculus (νTL) over infinite traces [Barringer *et al.*, 1986] as a high-level goal specification language. νTL is an extension of LTL with least and greatest fixpoint operators whose expressive power is ω-regular [Emerson and Clarke, 1980]. Fixpoints not only nicely capture the non-terminating behaviors of intelligent systems, but also allow us to express a much wider range of temporally extended goals than LTL does. However, the current best time complexity for the decision problem of νTL is $2^{O(|\phi|^2 \log |\phi|)}$ [Kaivola, 1995; Bradfield *et al.*, 1996; Dax *et al.*, 2006], which prevents νTL from being used directly as goal formulas for planning. Given a formula ϕ, the logic $\mathcal{G}_\mu$ we present in this paper stipulates, for each least fixpoint subformula $\mathcal{F}$ of ϕ and each formula of the form $\phi_1 \wedge \phi_2$ in the closure of ϕ, that $\mathcal{F}$ appears at most in one conjunct of $\phi_1 \wedge \phi_2$. Despite this restriction, $\mathcal{G}_\mu$ is still very expressive. Consider the following goal: "a sweeping robot must clean a house on every Monday (whether it cleans the house on other days is not cared about)". Obviously, such a goal is not expressible in LTL. However, it can be easily specified by a simple greatest fixpoint formula $\nu X.(p_{mon} \wedge \bigcirc \bigcirc \bigcirc \bigcirc \bigcirc \bigcirc \bigcirc X)$ of $\mathcal{G}_\mu$.

In this paper, we focus on the satisfiability problem of $\mathcal{G}_\mu$. Motivated by the idea of formula progression [Bacchus and Kabanza, 1998], we define Goal Progression Form (GPF) for $\mathcal{G}_\mu$ formulas and prove that every closed formula can be transformed into this form. GPF decomposes a formula into the present and future parts. The present part is the conjunction of atomic propositions or their negations while the future part is the conjunction of elements in the closure of a given formula. Additionally, based on GPF, we introduce the notion of Goal Progression Form Graph (GPG) which can be used to describe models of a formula. In a GPG, an edge may be associated with a mark which is a subset of variables occurring in the formula and utilized to keep track of the infinite unfolding problem for least fixpoints. Further, we present a decision procedure for checking satisfiability of $\mathcal{G}_\mu$ formulas based on GPG. It is achieved, with the help of marks, by searching for a ν-path in a GPG on which no least fixpoint unfolds itself infinitely. We show that the time complexity of the proposed decision procedure is $2^{O(|\phi|)}$, which is equivalent to that of LTL [Sistla and Clarke, 1985]. This makes $\mathcal{G}_\mu$ useful in temporal planning: with $\mathcal{G}_\mu$, more goals can be specified than utilizing LTL while the complexity keeps the same with LTL.

GPGs are very useful for generating plans for $\mathcal{G}_\mu$ goals by exploiting ν-paths. Moreover, following the decision procedure mentioned above, we can easily obtain a GPG-based model checking approach for $\mathcal{G}_\mu$ which can be further applied to dealing with different kinds of planning problems.

2 Preliminaries

Let $\mathcal{P}$ be a set of atomic propositions, and $\mathcal{V}$ a set of variables. νTL formulas are constructed based on the following syntax:

$$\phi ::= p \mid \neg p \mid X \mid \phi \vee \phi \mid \phi \wedge \phi \mid \bigcirc \phi \mid \mu X.\phi \mid \nu X.\phi$$

*Corresponding authors. All authors are joint first authors.

where p ranges over $\mathcal{P}$ and X over $\mathcal{V}$.

We use σ to denote either μ or ν, and $\dot{p}$ to denote either p or $\neg p$. An occurrence of a variable X in a formula is called *free* if it does not lie in the scope of σX; it is called *bound* otherwise. A formula is called *closed* if it contains no free variables. We write $\phi[\phi'/Y]$ for the result of simultaneously substituting ϕ' for all free occurrences of the variable Y in ϕ. For each variable X in a formula, we assume that X is bound at most once. Thus, it can be seen that all formulas constructed by the syntax above are in *positive normal form* [Kozen, 1983], i.e. negations can be applied only to atomic propositions and each variable occurring in a formula is bound at most once. Then, for each bound variable X in a formula ϕ, there exists a unique fixpoint subformula $\sigma X.\varphi$ of ϕ *identified by* X. A *μ-variable* (resp. *ν-variable*) in ϕ is a variable which identifies a formula of the form $\mu X.\varphi$ (resp. $\nu X.\varphi$).

For two formulas $\sigma X.\phi$ and $\sigma Y.\phi'$ where $\sigma Y.\phi'$ is a subformula of $\sigma X.\phi$, we say Y *depends on* X, denoted by $X \lhd Y$, iff X occurs free in ϕ'. The dependency relationship among the variables in a formula is transitive.

A formula is *guarded* if, for each bound variable X in that formula, every occurrence of X is in the scope of a next ($\bigcirc$) operator. Every formula can be transformed into an equivalent one in guarded form with an exponential increase in the size of the formula in the worst case [Bruse *et al.*, 2015].

νTL formulas are interpreted over linear time structures. A *linear time structure* over $\mathcal{P}$ is a function $\mathcal{K}\colon \mathcal{N} \to 2^{\mathcal{P}}$ where $\mathcal{N}$ denotes the set of natural numbers. The semantics of νTL formulas, relative to $\mathcal{K}$ and an environment $e : \mathcal{V} \to 2^{\mathcal{N}}$, is defined as follows:

$$
\begin{array}{rcl}
\|p\|_e^{\mathcal{K}} & := & \{i \in \mathcal{N} \mid p \in \mathcal{K}(i)\} \\
\|\neg p\|_e^{\mathcal{K}} & := & \{i \in \mathcal{N} \mid p \notin \mathcal{K}(i)\} \\
\|X\|_e^{\mathcal{K}} & := & e(X) \\
\|\varphi \vee \psi\|_e^{\mathcal{K}} & := & \|\varphi\|_e^{\mathcal{K}} \cup \|\psi\|_e^{\mathcal{K}} \\
\|\varphi \wedge \psi\|_e^{\mathcal{K}} & := & \|\varphi\|_e^{\mathcal{K}} \cap \|\psi\|_e^{\mathcal{K}} \\
\| \bigcirc \varphi\|_e^{\mathcal{K}} & := & \{i \in \mathcal{N} \mid i+1 \in \|\varphi\|_e^{\mathcal{K}}\} \\
\|\mu X.\varphi\|_e^{\mathcal{K}} & := & \bigcap\{W \subseteq \mathcal{N} \mid \|\varphi\|_{e[X \mapsto W]}^{\mathcal{K}} \subseteq W\} \\
\|\nu X.\varphi\|_e^{\mathcal{K}} & := & \bigcup\{W \subseteq \mathcal{N} \mid W \subseteq \|\varphi\|_{e[X \mapsto W]}^{\mathcal{K}}\}
\end{array}
$$

where $e[X \mapsto W]$ is the environment e' agreeing with e except for $e'(X) = W$. e is used to evaluate free variables and can be dropped when ϕ is closed.

For a given formula ϕ, we say ϕ is true at state i of a linear time structure $\mathcal{K}$, denoted by $\mathcal{K}, i \models \phi$, iff $i \in \|\phi\|_e^{\mathcal{K}}$. We say ϕ is *valid*, denoted by $\models \phi$, iff $\mathcal{K}, j \models \phi$ for all linear time structures $\mathcal{K}$ and all states j of $\mathcal{K}$; ϕ is *satisfiable* iff there exists a linear time structure $\mathcal{K}$ and a state j of $\mathcal{K}$ such that $\mathcal{K}, j \models \phi$.

Let Ord denote the class of ordinals. *Approximants* of fixpoint formulas are defined inductively by: $\mu^0 X.\phi = \bot$, $\nu^0 X.\phi = \top$, $\sigma^{\alpha+1} X.\phi = \phi[\sigma^{\alpha} X.\phi/X]$, $\mu^{\lambda} X.\phi = \bigvee_{\alpha<\lambda} \mu^{\alpha} X.\phi$ and $\nu^{\lambda} X.\phi = \bigwedge_{\alpha<\lambda} \nu^{\alpha} X.\phi$ where $\alpha, \lambda \in Ord$. In particular, λ is a limit ordinal. The following lemma [Tarski, 1955] is a standard result about approximants.

Lemma 1 ([Tarski, 1955]) *For a linear time structure $\mathcal{K}$, we say $\mathcal{K}, 0 \models \nu X.\phi$ iff $\forall \alpha \in Ord$, $\mathcal{K}, 0 \models \nu^{\alpha} X.\phi$; and $\mathcal{K}, 0 \models \mu Y.\phi$ iff $\exists \alpha \in Ord$, $\mathcal{K}, 0 \models \mu^{\alpha} Y.\phi$.*

The *closure* $CL(\phi)$ of a formula ϕ, based on [Fischer and Ladner, 1979], is the least set of formulas such that: (1) $\phi, true \in CL(\phi)$; (2) if $\varphi \vee \psi$ or $\varphi \wedge \psi \in CL(\phi)$, then $\varphi, \psi \in CL(\phi)$; (3) if $\bigcirc\varphi \in CL(\phi)$, then $\varphi \in CL(\phi)$; (4) if $\sigma X.\varphi \in CL(\phi)$, then $\varphi[\sigma X.\varphi/X] \in CL(\phi)$. It has been proved that the size of $CL(\phi)$ is linear in the size of ϕ (denoted by $|\phi|$) [Fischer and Ladner, 1979].

Given a formula ϕ, we call it a $\mathcal{G}_\mu$ formula iff for each least fixpoint subformula $\mu X.\varphi$ of ϕ we are focusing on and each $\phi_1 \wedge \phi_2 \in CL(\phi)$, $\mu X.\varphi$ appears at most in one ϕ_i ($i \in \{1, 2\}$). For example, $\nu X.\mu Y.(\bigcirc Y \vee p \wedge \bigcirc X)$ and $\mu X.(q \vee p \wedge \bigcirc X) \wedge \nu Y.(r \wedge \bigcirc\bigcirc Y)$ are $\mathcal{G}_\mu$ formulas while $\nu X.(\mu Y.(p \vee \bigcirc Y) \wedge \bigcirc X)$ and $\mu X.(p \vee \bigcirc X \wedge \bigcirc\bigcirc X)$ are not. The syntax of $\mathcal{G}_\mu$ enables us to trace conveniently the infinite unfolding problem for least fixpoints and provide efficient decision procedures.

$\mathcal{G}_\mu$ can be employed to specify a wide variety of properties. Currently, there already exists a fragment of modal μ-calculus [Kozen, 1983], called *deterministic μ-calculus* ($\mathcal{D}_\mu$), which has been used in motion planning [Karaman and Frazzoli, 2009]. The syntax of $\mathcal{D}_\mu$ is defined as follows:

$$\phi ::= p \mid \neg p \mid X \mid \phi \vee \phi \mid p \wedge \phi \mid \neg p \wedge \phi \mid \Diamond\phi \mid \mu X.\phi \mid \nu X.\phi$$

Compared with modal μ-calculus, $\mathcal{D}_\mu$ allows only the existential next-state operator $\Diamond$. Moreover, for the boolean connective $\wedge$, at least one conjunct is a proposition. Despite its succinct syntax, $\mathcal{D}_\mu$ is able to specify all ω-regular properties. However, due to the strict restriction on the boolean connective $\wedge$, many properties cannot be directly expressed in $\mathcal{D}_\mu$. We can employ $\mathcal{G}_\mu$ to overcome this weakness. For instance, the $\mathcal{G}_\mu$ formula $\mu X.(q \vee p \wedge \bigcirc X) \wedge \nu Y.(r \wedge \bigcirc\bigcirc Y)$ cannot be intuitively represented as a formula in $\mathcal{D}_\mu$. Since the restriction on the boolean connective $\wedge$ for $\mathcal{G}_\mu$ is strictly weaker than that for $\mathcal{D}_\mu$, $\mathcal{G}_\mu$ will be no less in expressive power than $\mathcal{D}_\mu$.

From now on, we confine ourselves to $\mathcal{G}_\mu$ formulas in guarded form with no $\vee$ appearing as the main operator under each next ($\bigcirc$) operator. This can be easily achieved by pushing next ($\bigcirc$) operators inwards using the equivalence $\bigcirc(\phi_1 \vee \phi_2) \equiv \bigcirc\phi_1 \vee \bigcirc\phi_2$.

3 GPF of $\mathcal{G}_\mu$ Formulas

In this section, we define GPF of $\mathcal{G}_\mu$ formulas and prove that every closed formula can be transformed into this form.

Definition 1 *Let ϕ be a closed formula, $\mathcal{P}_\phi$ the set of atomic propositions appearing in ϕ.* GPF *of ϕ is defined by: $\phi \equiv \bigvee_{i=1}^{n}(\phi_{p_i} \wedge \bigcirc\phi_{f_i})$, where $\phi_{p_i} \equiv \bigwedge_{h=1}^{n_1} \dot{q}_{ih}$, $q_{ih} \in \mathcal{P}_\phi$ for each h, and $\phi_{f_i} \equiv \bigwedge_{m=1}^{n_2} \phi_{im}$, $\phi_{im} \in CL(\phi)$ for each m.*

Intuitively, in a GPF, ϕ_{p_i} represents the present part while ϕ_{f_i} represents the future one.

Theorem 2 *Every closed formula φ can be transformed into GPF.*

Proof. Let $Conj(\psi)$ represent the set of all conjuncts in ψ. The proof proceeds by induction on the structure of φ.

- **Base case:**

– φ is p (or $\neg p$): p (or $\neg p$) can be written as: $p \equiv p \wedge \bigcirc true$ (or $\neg p \equiv \neg p \wedge \bigcirc true$). Therefore, φ can be transformed into GPF in these two cases.

- **Induction:**

– φ is $\bigcirc\phi$: $\bigcirc\phi$ can be written as: $\bigcirc\phi \equiv \bigvee_i (true \wedge \bigcirc\phi_i)$. For each $\phi_c \in Conj(\phi_i)$, we have $\phi_c \in CL(\varphi)$ since $\phi \in CL(\varphi)$. Hence, φ can be transformed into GPF in this case.

– φ is $\phi_1 \vee \phi_2$: by induction hypothesis, both ϕ_1 and ϕ_2 can be transformed into GPF: $\phi_1 \equiv \bigvee_{i=1}^{n}(\phi_{1p_i} \wedge \bigcirc\phi_{1f_i})$, $\phi_2 \equiv \bigvee_{j=1}^{m}(\phi_{2p_j} \wedge \bigcirc\phi_{2f_j})$ where $\phi_{1c} \in Conj(\phi_{1f_i})$ and $\phi_{1c} \in CL(\phi_1)$, $\phi_{2c} \in Conj(\phi_{2f_j})$ and $\phi_{2c} \in CL(\phi_2)$, for each i and j. Then, we have $\varphi \equiv \phi_1 \vee \phi_2 \equiv \bigvee_{i=1}^{n}(\phi_{1p_i} \wedge \bigcirc\phi_{1f_i}) \vee \bigvee_{j=1}^{m}(\phi_{2p_j} \wedge \bigcirc\phi_{2f_j})$. Since $\phi_1 \vee \phi_2 \in CL(\varphi)$, we have $\phi_1, \phi_2 \in CL(\varphi)$. For each $\phi_{1c} \in Conj(\phi_{1f_i})$, by induction hypothesis, we have $\phi_{1c} \in CL(\phi_1)$. Therefore, $\phi_{1c} \in CL(\varphi)$. Similarly, we can obtain that each $\phi_{2c} \in CL(\varphi)$. Thus, φ can be transformed into GPF in this case.

– φ is $\phi_1 \wedge \phi_2$: by induction hypothesis, both ϕ_1 and ϕ_2 can be transformed into GPF: $\phi_1 \equiv \bigvee_{i=1}^{n}(\phi_{1p_i} \wedge \bigcirc\phi_{1f_i})$, $\phi_2 \equiv \bigvee_{j=1}^{m}(\phi_{2p_j} \wedge \bigcirc\phi_{2f_j})$ where $\phi_{1c} \in Conj(\phi_{1f_i})$ and $\phi_{1c} \in CL(\phi_1)$, $\phi_{2c} \in Conj(\phi_{2f_j})$ and $\phi_{2c} \in CL(\phi_2)$, for each i and j. Then φ can be further converted into: $\varphi \equiv \phi_1 \wedge \phi_2 \equiv (\bigvee_{i=1}^{n}(\phi_{1p_i} \wedge \bigcirc\phi_{1f_i})) \wedge (\bigvee_{j=1}^{m}(\phi_{2p_j} \wedge \bigcirc\phi_{2f_j})) \equiv \bigvee_{i=1}^{n}\bigvee_{j=1}^{m}(\phi_{1p_i} \wedge \phi_{2p_j} \wedge \bigcirc(\phi_{1f_i} \wedge \phi_{2f_j}))$. Since $\phi_1 \wedge \phi_2 \in CL(\varphi)$, we have $\phi_1, \phi_2 \in CL(\varphi)$. For each $\phi_{1c} \in Conj(\phi_{1f_i})$, by induction hypothesis, we have $\phi_{1c} \in CL(\phi_1)$. Hence, $\phi_{1c} \in CL(\varphi)$. Similarly, we can obtain that each $\phi_{2c} \in CL(\varphi)$. Therefore, all conjuncts behind the next ($\bigcirc$) operators in φ belong to $CL(\varphi)$ and φ can be transformed into GPF in this case.

– φ is $\mu X.\phi$: to transform $\mu X.\phi$ into GPF, we first need to unfold it using the equivalence $\mu X.\phi \equiv \phi[\mu X.\phi/X]$. That is to say, we can treat the free variable X occurring in ϕ as an atomic proposition when transforming ϕ into GPF since X will finally be replaced by $\mu X.\phi$. As a result, by induction hypothesis, ϕ can be transformed into GPF: $\phi \equiv \bigvee_{i=1}^{n}(\phi_{p_i} \wedge \bigcirc\phi_{f_i})$. For each $\phi_c \in Conj(\phi_{f_i})$, by induction hypothesis, we have $\phi_c \in CL(\phi)$. Further, by substituting $\mu X.\phi$ for all the free occurrences of X in ϕ, we have that $\varphi \equiv \phi[\mu X.\phi/X] \equiv \bigvee_{i=1}^{n}(\phi_{p_i} \wedge \bigcirc\phi_{f_i}[\mu X.\phi/X])$. Since $\phi_c \in CL(\phi)$ for each ϕ_c, we can easily obtain that $\phi_c[\mu X.\phi/X] \in CL(\phi[\mu X.\phi/X])$ after the substitution. Since $\phi[\mu X.\phi/X] \in CL(\varphi)$, we have $\phi_c[\mu X.\phi/X] \in CL(\varphi)$. Therefore, φ can be transformed into GPF in this case.

– φ is $\nu X.\phi$: this case can be proved similarly to the case when φ is $\mu X.\phi$. □

Following Theorem 2, we present algorithm *GPFTr* in the following for transforming a closed formula ϕ into GPF.

GPFTr uses the function *AND* to deal with the boolean connective $\wedge$. It can be seen that the inputs, ψ and φ, for *AND* are both in GPF. Suppose ψ is of the form $\bigvee_i(\psi_i \wedge \bigcirc\psi'_i)$ while φ of the form $\bigvee_j(\varphi_j \wedge \bigcirc\varphi'_j)$. *AND* finally returns $\bigvee_i\bigvee_j(\psi_i \wedge \varphi_j \wedge \bigcirc(\psi'_i \wedge \varphi'_j))$.

Algorithm 1 GPFTr(ϕ)

case
  ϕ is $true$: **return** $true \wedge \bigcirc true$
  ϕ is $false$: **return** $false$
  ϕ is ϕ_p where $\phi_p \equiv \bigwedge_{h=1}^{n} \dot{q}_h$: **return** $\phi_p \wedge \bigcirc true$
  ϕ is $\phi_p \wedge \bigcirc\varphi$: **return** $\bigvee_i(\phi_p \wedge \bigcirc\varphi_i)$
  ϕ is $\bigcirc\varphi$: **return** $\bigvee_i(true \wedge \bigcirc\varphi_i)$
  ϕ is $\phi_1 \vee \phi_2$: **return** GPFTr(ϕ_1) $\vee$ GPFTr(ϕ_2)
  ϕ is $\phi_1 \wedge \phi_2$: **return** AND(GPFTr(ϕ_1), GPFTr(ϕ_2))
  ϕ is $\sigma X.\varphi$: **return** GPFTr($\varphi[\sigma X.\varphi/X]$)
end case

Theorem 3 *Transforming a formula ϕ into GPF by algorithm GPFTr can be completed in $2^{O(|\phi|)}$.*

Proof. The proof proceeds by induction on the structure of ϕ.

- **Base case:**

– ϕ is $true$, $false$, ϕ_p, $\phi_p \wedge \bigcirc\varphi$ (where ϕ_p is of the form $\bigwedge_{h=1}^{n} \dot{q}_h$), or $\bigcirc\varphi$: the theorem holds obviously in these cases.

- **Induction:**

– ϕ is $\phi_1 \vee \phi_2$: by induction hypothesis, *GPFTr*(ϕ_1) and *GPFTr*(ϕ_2) can be done in $2^{O(|\phi_1|)}$ and $2^{O(|\phi_2|)}$, respectively. Therefore, *GPFTr*(ϕ) can be finished in $2^{O(|\phi_1|)} + 2^{O(|\phi_2|)}$, namely $2^{O(|\phi|)}$.

– $\phi = \phi_1 \wedge \phi_2$: by induction hypothesis, *GPFTr*(ϕ_1) and *GPFTr*(ϕ_2) can be completed in $2^{O(|\phi_1|)}$ and $2^{O(|\phi_2|)}$, respectively. After the GPF transformation, the number of disjuncts in ϕ_1 (resp. ϕ_2) is bounded by $2^{O(|\phi_1|)}$ (resp. $2^{O(|\phi_2|)}$). Hence, the function *AND* can be accomplished in $2^{O(|\phi_1|+|\phi_2|)}$. It follows that *GPFTr*(ϕ) can be done in $2^{O(|\phi_1|)} + 2^{O(|\phi_2|)} + 2^{O(|\phi_1|+|\phi_2|)}$, namely $2^{O(|\phi|)}$.

– $\phi = \sigma X.\varphi$: regarding X as an atomic proposition, φ can be transformed into GPF by algorithm *GPFTr*, which can be accomplished, by induction hypothesis, in $2^{O(|\varphi|)}$. Subsequently, by substituting $\sigma X.\varphi$ for all free occurrences of X in φ, we have that $\varphi[\sigma X.\varphi/X]$ can be transformed into GPF by algorithm *GPFTr* in $2^{O(|\varphi|)}$. Therefore, *GPFTr*($\sigma X.\varphi$) can be done in $2^{O(|\phi|)}$. □

4 Goal Progression Form Graph

The GPG, G_ϕ, of a formula ϕ is a tuple (N_ϕ, E_ϕ, n_0) where N_ϕ is a set of nodes, E_ϕ a set of directed edges, and n_0 the root node. Each node in N_ϕ is specified by the conjunction of formulas in $CL(\phi)$ while each edge in E_ϕ is identified by a triple (ϕ_0, ϕ_e, ϕ_1), where ϕ_0, $\phi_1 \in N_\phi$ and ϕ_e is the label of the edge from ϕ_0 to ϕ_1. An edge may be associated with a mark which is a subset of variables occurring in ϕ.

Definition 2 *Given a closed formula ϕ. N_ϕ and E_ϕ are inductively defined by: (1) $n_0 = \phi \in N_\phi$; (2) for all $\varphi \in N_\phi \backslash \{false\}$, if $\varphi \equiv \bigvee_{i=1}^{k}(\varphi_{p_i} \wedge \bigcirc\varphi_{f_i})$, then $\varphi_{f_i} \in N_\phi$, $(\varphi, \varphi_{p_i}, \varphi_{f_i}) \in E_\phi$ for each i ($1 \le i \le k$).*

In a GPG, the root node is denoted by a double circle while each of other nodes by a single circle. Each edge is denoted by a directed arc with a label and also possibly a mark that connects two nodes. To simplify notations, we usually use variables to represent the corresponding fixpoint formulas occurring in a node. An example of GPG for formula $\mu X.(p \vee \bigcirc X) \vee \nu Y.(q \wedge \bigcirc Y)$ is depicted in Figure 1. There are four nodes in the GPG where n_0 is the root node. (n_0, q, n_3) is an edge with the label being q and the mark being $\{Y\}$ while (n_0, p, n_1) is an edge with the label being p and no mark.

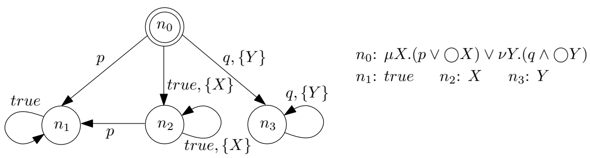

Figure 1: An example of GPG

A *path* Π in a GPG is an infinitely alternate sequence of nodes and edges departing from the root node. Let $Atom(\bigwedge_{i=1}^{m} \dot{q}_i)$ denote the set of atomic propositions or their negations appearing in formula $\bigwedge_{i=1}^{m} \dot{q}_i$. Given a path $\Pi = \phi_0, \phi_{e0}, \phi_1, \phi_{e1}, \ldots$ in a GPG, we can obtain its corresponding linear time structure $Atom(\phi_{e0}), Atom(\phi_{e1}), \ldots$. For example, in Figure 1, the path $n_0, true, n_2, p, (n_1, true)^\omega$ corresponds to the linear time structure $\{true\}\{p\}\{true\}^\omega$.

From Figure 1 we can see that there may exist a path in a GPG, e.g. $n_0, true, (n_2, true)^\omega$, which arises from the infinite unfolding of a least fixpoint. Thus, marks are essential in a GPG to keep track of the infinite unfolding problem for least fixpoints when constructing the GPG.

Definition 3 *Given a GPG* G_ϕ *and a node* $\phi_m \in N_\phi$ *where* $\phi_m \equiv \bigvee_{i=1}^{k}(\phi_{p_i} \wedge \bigcirc \phi_{f_i})$. *The* mark *of an edge* $(\phi_m, \phi_{p_i}, \phi_{f_i})$ $(1 \leq i \leq k)$ *is a set of variables* M_v *such that for each* $X \in M_v$*, its corresponding fixpoint formula* $\sigma X.\phi_X$ *appears as a subformula of* ϕ_{f_i} *and has been unfolded by itself in the GPF transformation process.*

When transforming a formula into GPF, the occurrence of a fixpoint formula $\sigma X.\phi_X$ in the future part may be caused by the unfolding of either (I) itself, or (II) another fixpoint formula. For example, as shown in Figure 2, when the node n_0 is transformed into GPF: $n_0 \equiv true \wedge \bigcirc n_1$, the occurrence of $\nu Y.(p \wedge \bigcirc Y)$ in n_1 is due to the unfolding of $\mu X. \bigcirc (\bigcirc \nu Y.(p \wedge \bigcirc Y) \wedge X)$, hence Y does not exist in the mark of the edge $(n_0, true, n_1)$.

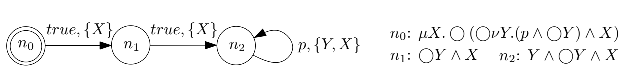

Figure 2: GPG of $\mu X. \bigcirc (\bigcirc \nu Y.(p \wedge \bigcirc Y) \wedge X)$

Note that the cases I and II above may occur simultaneously for a greatest fixpoint formula $\nu Z.\phi_Z$, we will add Z to the corresponding mark in such a situation, e.g. the occurrence of Y in the mark of the edge (n_2, p, n_2) in Figure 2. However, for a least fixpoint formula, the cases I and II cannot happen simultaneously due to the syntactic restriction of $\mathcal{G}_\mu$. Actually, this restriction makes, for a given formula ϕ, each $\mu X.\varphi \in CL(\phi)$ occur at most in one conjunct of each node in G_ϕ. This further facilitates the tracing of the infinite unfolding problem for least fixpoints.

Given a closed formula ϕ, the whole process of constructing G_ϕ is presented in Algorithm 2.

Algorithm 2 GPGCON(ϕ)

```
1: n_0 = φ, N_φ = {n_0}, E_φ = ∅
2: while there exists an unhandled φ ∈ N_φ \ {false} do
3:     φ = GPFTr(φ)   /*suppose φ = ⋁_{i=1}^{k}(φ_{p_i} ∧ ○φ_{f_i})*/
4:     for i = 1 to k do
5:         E_φ = E_φ ∪ {(φ, φ_{p_i}, φ_{f_i})}
6:         N_φ = N_φ ∪ {φ_{f_i}}
7:         MARK((φ, φ_{p_i}, φ_{f_i}))
8:     end for
9: end while
10: for all φ ∈ N_φ with no outgoing edge do
11:     N_φ = N_φ \ {φ}
12:     E_φ = E_φ \ ⋃_i{(φ_i, φ_e, φ)}
13: end for
14: return G_φ
```

The algorithm repeatedly converts an unhandled formula $\varphi \in N_\phi$ into GPF and then adds the corresponding nodes and edges to N_ϕ and E_ϕ, respectively, until all formulas in N_ϕ have been handled. Function *MARK* is utilized to mark an edge with a subset of variables occurring in ϕ by distinguishing appropriate fixpoint formulas from all fixpoint formulas contained in the future part of a certain GPF. Given an edge $(\varphi, \varphi_{p_i}, \varphi_{f_i})$, *MARK* checks each conjunct φ_c of φ_{f_i}. If φ_c is of the form $\bigcirc^n \sigma X.\varphi_X$ $(n \geq 0)$ and $\sigma X.\varphi_X$ has been unfolded by itself in the GPF transformation process, X will be added to the mark of the edge. Here $\bigcirc^n$ represents the consecutive occurrence of next ($\bigcirc$) operators for n times. Additionally, throughout the construction of G_ϕ, a *false* node (e.g. $q \wedge \neg q$) may be generated which corresponds to an inconsistent subset of $CL(\phi)$. Such kind of nodes have no successor and are redundant. We use the *for loop* in Line 10 of the algorithm to remove these nodes and the relative edges.

In the GPG G_ϕ of a formula ϕ, since each node in N_ϕ is the conjunction of formulas in $CL(\phi)$, the following corollary is easily obtained.

Corollary 4 *For any closed formula* ϕ*, both the number of nodes and the number of edges in* G_ϕ *are bounded by* $2^{O(|\phi|)}$.

Theorem 5 *Constructing the GPG of a formula* ϕ *by algorithm GPGCON can be done in* $2^{O(|\phi|)}$.

Proof. By Corollary 4, the number of iterations of the *while loop* is bounded by $2^{O(|\phi|)}$. In each iteration, algorithm *GPFTr* is called first, which can be done in $2^{O(|\phi|)}$ by Theorem 3. Subsequently, after the GPF transformation, we have that the number of iterations of the *for loop* in Line 4 is bounded by $2^{O(|\phi|)}$. Function *MARK* checks if a fixpoint formula, which has been unfolded by itself in a GPF transformation process, exists in the future part of the GPF and can be completed in $O(|\phi|)$. Therefore, the *while loop* can be finished in $2^{O(|\phi|)}$. Further, it is obvious that eliminating redundant nodes and the relative edges can be finished in $2^{O(|\phi|)}$. Consequently, *GPGCON* can be done in $2^{O(|\phi|)}$. $\square$

5 A Decision Procedure Based on GPG

In this section we show how to find a model for a given formula ϕ from G_ϕ. According to the theory of eventually periodic models [Banieqbal and Barringer, 1989], we restrict ourselves here only to the paths ending with loops in G_ϕ. Let Π be a path in G_ϕ, for convenience, we use *LES*(Π) to denote the set of edges appearing in the loop part of Π, *Mark*(e) the mark of an edge e, and *LMS*(Π) the set of all the μ-variables occurring in each *Mark*(e_l) where $e_l \in$ *LES*(Π).

In the following, we present the notion of *ν-paths* which will play a vital role in obtaining the GPG-based decision procedure for $\mathcal{G}_\mu$.

Definition 4 *Given a GPG G_ϕ and a path Π in G_ϕ. We call Π a ν-path iff for each $X \in$ LMS(Π), an edge $e \in$ LES(Π) can be found such that $X \notin$ Mark(e) and there exists no $X' \in$ Mark(e) where $X \lhd X'$.*

We consider the following two paths in Figure 1 to show what a ν-path is: (1) Π_1: $n_0, q, (n_3, q)^\omega$. Π_1 is a ν-path since *LMS*$(\Pi_1) = \emptyset$; (2) Π_2: $n_0, true, (n_2, true)^\omega$. We have that *LES*$(\Pi_2) = \{(n_2, true, n_2)\}$ and *LMS*$(\Pi_2) = \{X\}$. For the only μ-variable $X \in$ *LMS*(Π_2), we cannot find an edge from *LES*(Π_2) whose mark does not contain X. Therefore, Π_2 is not a ν-path.

Regarding the notion of ν-paths, the following theorem is formalized.

Theorem 6 *A closed formula ϕ is satisfiable iff a ν-path can be found in G_ϕ.*

Proof. ($\Rightarrow$) Suppose ϕ is satisfiable and no ν-path exists in G_ϕ. In this case, for any path $\Pi_1 = \phi_0, \phi_{e0}, \phi_1, \phi_{e1}, \ldots, (\phi_k, \phi_{ek}, \phi_{k+1}, \phi_{e(k+1)}, \ldots, \phi_l, \phi_{el})^\omega$ in G_ϕ, there exists at least one $X \in$ *LMS*(Π_1) such that for each edge $e_1 \in$ *LES*(Π_1), either $X \in$ *Mark*(e_1) or $X' \in$ *Mark*(e_1), where $X \lhd X'$. As a result, we can obtain the following sequence of variables $\mathcal{X}$ according to the sequence of marks in the loop part of Π_1: $X_1, X_2, X_3, \ldots, X_{l-k+1}$, where each $X_i \in$ *Mark*$((\phi_{i+k-1}, \phi_{e(i+k-1)}, \phi_{i+k}))$ $(1 \le i \le l-k+1)$ is either X itself or a variable depending on X. Let ϕ_{X_i} denote the conjunct in ϕ_{i+k} corresponding to X_i, then $\mathcal{X}$ determines a sequence of formulas Φ_X: $\phi_{X_1}, \phi_{X_2}, \phi_{X_3}, \ldots, \phi_{X_{l-k+1}}$. For each variable X_j on $\mathcal{X}$ that depends on X, we have that $\mu X.\phi_X$ appears as a subformula of ϕ_{X_j}, where $\mu X.\phi_X \in CL(\phi)$ represents the fixpoint formula corresponding to X. Suppose Π_1 characterizes a model $\mathcal{K}$. There must exist a state t_1 of $\mathcal{K}$ where $\mu X.\phi_X$ can be satisfied. By Lemma 1, there exists an ordinal m such that $\mathcal{K}, t_1 \models \mu^m X.\phi_X$. Pushing this satisfaction further down the sequence Φ_X, we will eventually reach a state t_2 of $\mathcal{K}$ such that $\mathcal{K}, t_2 \models \mu^0 X.\phi_X$, which is impossible. Therefore, we can see that Π_1 does not characterize a model of ϕ. This contradicts the premise that ϕ is satisfiable. Therefore, if ϕ is satisfiable, there exists at least one ν-path in G_ϕ.

($\Leftarrow$) Let $\Pi_2 = \phi_0, \phi_{e0}, \phi_1, \phi_{e1}, \ldots, (\phi_k, \phi_{ek}, \phi_{k+1}, \phi_{e(k+1)}, \ldots, \phi_l, \phi_{el})^\omega$ be a ν-path in G_ϕ. When *LMS*(Π_2) is empty, the infinite unfolding problem for least fixpoints will not be involved on Π_2. Consequently, Π_2 characterizes a model of ϕ in this case.

When *LMS*(Π_2) is not empty, we have that for each $Y \in$ *LMS*(Π_2), an edge $e_2 \in$ *LES*(Π_2) can be found such that $Y \notin$ *Mark*(e_2) and there exists no $Y' \in$ *Mark*(e_2) where $Y \lhd Y'$. Subsequently, we can obtain the following sequence of variables $\mathcal{Y}$ according to the sequence of marks in the loop part of Π_2: $Y_1, Y_2, \ldots, Y_j, Y_{j+1}, \ldots, Y_{l-k+1}$, where each $Y_i \in$ *Mark*$((\phi_{i+k-1}, \phi_{e(i+k-1)}, \phi_{i+k}))$ $(1 \le i \le l-k+1)$, and Y_j is neither Y nor a variable depending on Y while any other variable on $\mathcal{Y}$ is the opposite. Let ϕ_{Y_i} denote the conjunct in ϕ_{i+k} corresponding to Y_i, then $\mathcal{Y}$ determines a sequence of formulas Φ_Y: $\phi_{Y_1}, \phi_{Y_2}, \phi_{Y_3}, \ldots, \phi_{Y_{l-k+1}}$. Further, we have that $\mu Y.\phi_Y$ does not appear as a subformula of ϕ_{Y_j}, where $\mu Y.\phi_Y \in CL(\phi)$ represents the fixpoint formula corresponding to Y. Similarly, we can obtain that Π_2 characterizes a model of ϕ using the notion of approximants. It follows that when there exists a ν-path in G_ϕ, ϕ is satisfiable. $\square$

Consequently, we reduce the satisfiability problem of a formula to a ν-path searching problem from its GPG. In the following, we present algorithm *NuSearch* used to find a ν-path from a GPG.

Algorithm 3 NuSearch(n_0)

```
NS.push_back(n0)
for each edge e in Gφ do
  if src[e] = n0 and visit[e] = 0 then
    ES.push_back(e)
    visit[e] = 1
    if LOOP(tgt[e], pos) then
      TES.assign(ES.begin() + pos, ES.end())
      if TES corresponds to a ν-path then
        return satisfiable
      end if
      ES.pop_back()
    else
      NuSearch(tgt[e])
    end if
  end if
end for
if ES.size() > 0 then
  ES.pop_back()
end if
NS.pop_back()
```

Given a GPG G_ϕ, the algorithm first takes the root node n_0 of G_ϕ as input and tries to build a ν-path. Two global variables, *ES* and *NS*, are used in the algorithm. *ES* is a vector which stores the sequence of edges aiming to construct a path ending with a loop. *NS* is also a vector storing the sequence of nodes corresponding to *ES*. In addition, *src*[] and *tgt*[] are employed to obtain the source and target nodes of an edge, respectively. *visit*$[e]$ = 1 (or. 0) indicates that an edge e has (or. has not) been visited. For each edge e in G_ϕ, *visit*$[e]$ is initialized to 0. *LOOP* is a simple boolean function which determines whether a node u exists in *NS* and obtains, if so, the position *pos* of u in *NS*.

In algorithm *NuSearch*, n_0 is added to *NS* first. After that, for each unvisited edge e in G_ϕ whose source node is n_0, the algorithm adds it to *ES* and assigns *visit*$[e]$ to 1. Then, it determines whether *tgt*$[e]$ exists in *NS* by means of function *LOOP*. If the output of *LOOP* is $true$, there exists a loop in *ES* and we use *TES* to store the loop of *ES*. Further, if *TES* corresponds to a ν-path, the given formula is satisfiable and the algorithm terminates; otherwise, the last edge in *ES* is removed and a new *for loop* begins in order to search for another unvisited edge from G_ϕ whose source node is n_0 to establish a new path. In case the output of *LOOP* is $false$, which means the current *ES* cannot construct a path ending with a loop, the algorithm calls itself and tries to build new paths from the node *tgt*$[e]$. If the conditional statement in Line 3 is never satisfied, i.e. any edge in G_ϕ with n_0 being its source node has been visited, n_0 is removed from *NS*. Note that if the size of *ES* is greater than 0 when the *for loop* terminates, we need to remove the last edge in *ES* generated by the next level of recursion.

Theorem 7 *For the GPG G_ϕ of a closed formula ϕ, algorithm NuSearch can be completed in $2^{O(|\phi|)}$.*

Proof. By Corollary 4, we have that both the number of nodes $|N_\phi|$ and the number of edges $|E_\phi|$ in G_ϕ are bounded by $2^{O(|\phi|)}$. Since each edge in G_ϕ is handled exactly once, the total number of recursive calls for *NuSearch* is bounded by $2^{O(|\phi|)}$. Moreover, the number of iterations of the *for loop* is also bounded by $2^{O(|\phi|)}$. Subsequently, function *LOOP* checks whether a node exists in *NS*, which can apparently be done in $2^{O(|\phi|)}$. Further, since the size of *TES* is in $2^{O(|\phi|)}$, by maintaining, for each μ-variable X occurring in ϕ, a list of variables depending on X, we can determine whether *TES* corresponds to a ν-path in $2^{O(|\phi|)}$. Therefore, algorithm *NuSearch* can be completed in $2^{O(|\phi|)}$. □

As a consequence of Theorems 5 and 7, we obtain the following theorem.

Theorem 8 *For a given closed formula ϕ, the GPG-based decision procedure can be done in $2^{O(|\phi|)}$.*

6 Related Work

GPG is a useful formalism for describing the models satisfying a formula. Therefore, it can be employed to generate plans for $\mathcal{G}_\mu$ goals. More precisely, it is ν-paths in a GPG that characterize such plans. Representing $\mathcal{G}_\mu$ goals as GPGs is similar to the compilation approaches [Rintanen, 2000; Cresswell and Coddington, 2004; Kabanza and Thiébaux, 2005; Edelkamp, 2006; Baier and McIlraith, 2006; Patrizi *et al.*, 2011] to planning for LTL goals which exploit the relationship between LTL and finite-state automata (FSA). The compilation approaches are particularly useful when there exists no search control knowledge. Recently, a novel method [Torres and Baier, 2015] to compile away finite LTL goals running in polynomial time is proposed. The method exploits alternating automata instead of FSA. However, all the abovementioned methods consider LTL goals, which are less expressive than $\mathcal{G}_\mu$ goals presented in this paper. In addition, although LTL has been extended in [De Giacomo and Vardi, 2013; 2015] in order to express a wider variety of goals, these extensions, unfortunately, consider only LTL over finite traces, which can be more easily handled than standard LTL over infinite traces. In contrast, our logic $\mathcal{G}_\mu$ focuses on infinite traces.

The decision problems of νTL have been extensively studied. In [Vardi, 1988], Vardi first adapts the classical automata theoretic decision procedure for modal μ-calculus [Streett and Emerson, 1984] to νTL with past operators, yielding an algorithm running in $2^{O(|\phi|^4)}$. Later, Banieqbal and Barringer [Banieqbal and Barringer, 1989] demonstrate that the satisfiability problem of a formula can be reduced to a good path searching problem from a graph. This method is equivalent in time complexity to Vardi's but runs in exponential space. In [Stirling and Walker, 1990], the first tableau characterization for νTL's decision problems is presented without mentioning complexity issues. After that, based on the work in [Kaivola, 1995], the tableau system is improved by simplifying the success conditions for a tableau [Bradfield *et al.*, 1996]. The algorithm obtained runs in $2^{O(|\phi|^2 \log |\phi|)}$. Further, a proof system for checking validity of νTL formulas is proposed in [Dax *et al.*, 2006] which runs in $2^{O(|\phi|^2 \log |\phi|)}$ and has been implemented in OCAML. However, the complexities of these decision procedures are too high, which hinders the use of νTL goals in planning. Therefore, we focus on an expressive fragment $\mathcal{G}_\mu$ of νTL in this paper and provide a better GPG-based decision procedure running in $2^{O(|\phi|)}$. This makes $\mathcal{G}_\mu$ a compelling goal specification language.

It is worth pointing out that the idea of breaking a formula into the present and future parts has also been considered in [Duan *et al.*, 2008; Duan and Tian, 2014; Duan *et al.*, 2016] to solve the decidability problem of Propositional Projection Temporal Logic (PPTL).

7 Conclusion

In this paper, we have investigated an expressive fragment $\mathcal{G}_\mu$ of νTL and presented GPF and GPG for $\mathcal{G}_\mu$ formulas. Also, we have proposed a simple GPG-based decision procedure for checking satisfiability of $\mathcal{G}_\mu$ formulas running in $2^{O(|\phi|)}$, which makes $\mathcal{G}_\mu$ a compelling alternative for specifying a richer class of goals in planning compared with LTL.

In the future, we are going to implement the proposed decision procedure. We also plan to develop a GPG-based model checker to solve different kinds of planning problems.

Acknowledgments

The authors would like to thank all the anonymous reviewers for their valuable comments on this paper. This research is supported by the National Natural Science Foundation of China Grant Nos. 61133001, 61322202, 61420106004, and 91418201.

References

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

[Baier and McIlraith, 2006] Jorge A. Baier and Sheila A. McIlraith. Planning with first-order temporally extended goals using heuristic search. In *AAAI*, pages 788–795. AAAI Press, 2006.

[Banieqbal and Barringer, 1989] Behnam Banieqbal and Howard Barringer. Temporal logic with fixed points. In *Temporal Logic in Specification*, volume 398 of LNCS, pages 62–74. Springer, 1989.

[Barringer *et al.*, 1986] Howard Barringer, Ruurd Kuiper, and Amir Pnueli. A really abstract concurrent model and its temporal logic. In *POPL*, pages 173–183. ACM Press, 1986.

[Bradfield *et al.*, 1996] Julian Bradfield, Javier Esparza, and Angelika Mader. An effective tableau system for the linear time μ-calculus. In *ICALP*, volume 1099 of LNCS, pages 98–109. Springer, 1996.

[Bruse *et al.*, 2015] Florian Bruse, Oliver Friedmann, and Martin Lange. On guarded transformation in the modal μ-calculus. *Logic Journal of the IGPL*, 23(2):194–216, 2015.

[Calvanese *et al.*, 2002] Diego Calvanese, Giuseppe De Giacomo, and Moshe Y. Vardi. Reasoning about actions and planning in LTL action theories. In *KR*, pages 593–602. Morgan Kaufmann, 2002.

[Cresswell and Coddington, 2004] Stephen Cresswell and Alexandra M. Coddington. Compilation of LTL goal formulas into PDDL. In *ECAI*, pages 985–986. IOS Press, 2004.

[Dax *et al.*, 2006] Christian Dax, Martin Hofmann, and Martin Lange. A proof system for the linear time μ-calculus. In *FSTTCS*, volume 4337 of LNCS, pages 274–285. Springer, 2006.

[De Giacomo and Vardi, 1999] Giuseppe De Giacomo and Moshe Y. Vardi. Automata-theoretic approach to planning for temporally extended goals. In *ECP*, volume 1809 of LNAI, pages 226–238. Springer, 1999.

[De Giacomo and Vardi, 2013] Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*, pages 854–860. AAAI Press, 2013.

[De Giacomo and Vardi, 2015] Giuseppe De Giacomo and Moshe Y. Vardi. Synthesis for LTL and LDL on finite traces. In *IJCAI*, pages 1558–1564. AAAI Press, 2015.

[Duan and Tian, 2014] Zhenhua Duan and Cong Tian. A practical decision procedure for propositional projection temporal logic with infinite models. *Theoretical Computer Science*, 554:169–190, 2014.

[Duan *et al.*, 2008] Zhenhua Duan, Cong Tian, and Li Zhang. A decision procedure for propositional projection temporal logic with infinite models. *Acta Informatica*, 45(1):43–78, 2008.

[Duan *et al.*, 2016] Zhenhua Duan, Cong Tian, and Nan Zhang. A canonical form based decision procedure and model checking approach for propositional projection temporal logic. *Theoretical Computer Science*, 609:544–560, 2016.

[Edelkamp, 2006] Stefan Edelkamp. On the compilation of plan constraints and preferences. In *ICAPS*, pages 374–377. AAAI Press, 2006.

[Emerson and Clarke, 1980] E. Allen Emerson and Edmund M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *ICALP*, volume 85 of LNCS, pages 169–181. Springer, 1980.

[Fischer and Ladner, 1979] Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979.

[Kabanza and Thiébaux, 2005] Froduald Kabanza and Sylvie Thiébaux. Search control in planning for temporally extended goals. In *ICAPS*, pages 130–139. AAAI Press, 2005.

[Kaivola, 1995] Roope Kaivola. A simple decision method for the linear time mu-calculus. In *Structures in Concurrency Theory*, pages 190–204. Springer, 1995.

[Karaman and Frazzoli, 2009] Sertac Karaman and Emilio Frazzoli. Sampling-based motion planning with deterministic μ-calculus specifications. In *CDC*, pages 2222–2229. IEEE Press, 2009.

[Kozen, 1983] Dexter Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27(3):333–354, 1983.

[Patrizi *et al.*, 2011] Fabio Patrizi, Nir Lipoveztky, Giuseppe De Giacomo, and Hector Geffner. Computing infinite plans for LTL goals using a classical planner. In *IJCAI*, pages 2003–2008. AAAI Press, 2011.

[Pnueli, 1977] Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE Press, 1977.

[Rintanen, 2000] Jussi Rintanen. Incorporation of temporal logic control into plan operators. In *ECAI*, pages 526–530. IOS Press, 2000.

[Sistla and Clarke, 1985] A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

[Stirling and Walker, 1990] Colin Stirling and David Walker. CCS, liveness, and local model checking in the linear time mu-calculus. In *Automatic Verification Methods for Finite State Systems*, volume 407 of LNCS, pages 166–178. Springer, 1990.

[Streett and Emerson, 1984] Robert S. Streett and E. Allen Emerson. The propositional mu-calculus is elementary. In *ICALP*, volume 172 of LNCS, pages 465–472. Springer, 1984.

[Tarski, 1955] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309, 1955.

[Torres and Baier, 2015] Jorge Torres and Jorge A. Baier. Polynomial-time reformulations of LTL temporally extended goals into final-state goals. In *IJCAI*, pages 1696–1703. AAAI Press, 2015.

[Vardi, 1988] Moshe Y. Vardi. A temporal fixpoint calculus. In *POPL*, pages 250–259. ACM Press, 1988.