Variable Elimination for Scalable Receding Horizon Temporal Logic Planning

Fält, Mattias; Raman, Vasumathi; Murray, Richard M.

Published in:
2015 American Control Conference (ACC)

DOI:
10.1109/ACC.2015.7171013

2015

Document Version:
Peer reviewed version (aka post-print)

Variable Elimination for Scalable Receding Horizon Temporal Logic Planning

Mattias Fält[1], Vasumathi Raman[2], Richard M. Murray[3]

***Abstract*— Correct-by-construction synthesis of high-level reactive control relies on the use of formal methods to generate controllers with provable guarantees on their behavior. While this approach has been successfully applied to a wide range of systems and environments, it scales poorly. A receding horizon framework mitigates this computational blowup, by decomposing the global control problem into several tractable subproblems. The existence of a global controller is ensured through symbolic checks of the specification, and local controllers are synthesized when needed. This reduces the size of the synthesized strategy, but still scales poorly for problems with dynamic environments because of the large number of environment strategies in each subproblem. Ad-hoc methods to locally restrict the environment come with the risk of losing correctness. We present a method for reducing the size of these subproblems by eliminating locally redundant variables, while maintaining correctness of the local (and thus global) controllers. We demonstrate the method using an autonomous car example, on problem sizes that were previously unsolvable due to the number of variables in the environment. We also demonstrate how the reduced specifications can be used to identify opportunities for reusing the synthesized local controllers.**

I. Introduction

As autonomous systems are deployed in situations of increasing complexity, it is important to be able to verify their correctness. We are interested in complex tasks where safety is critical, such as aircraft systems, autonomous cars and space missions. It is essential to have formal and unambiguous specifications of these tasks to be able to guarantee the desired system behavior.

Linear Temporal Logic (LTL) has proven effective for correct-by-construction synthesis of controllers for a wide range of applications. To apply these specifications to continuous domains, the system and its operating environment are usually represented by a discrete abstraction over which the problem can be specified. It is desirable to be able to automatically create provably correct reactive controllers from these specifications, and several methods for doing so have been presented, such as those in [5], [7], [12], [9]. In particular, efficient methods have been developed for the Generalized Reactivity (GR(1)) fragment of LTL. These methods are based on finding a winning strategy in a two player game between the system and environment. If successful, a correct-by-construction controller is generated that satisfies the specification on the discrete abstraction. A continuous controller can then be used to implement this strategy on the real system. However, as the problems grow in size, the curse of dimensionality sets in and it becomes increasingly intractable to generate these reactive strategies.

Receding horizon control has been successfully applied to many situations where solving the full problem at once is too hard. The method is shown to be effective not only in terms of computational scalability, but also robustness and stability. A receding horizon framework was therefore recently introduced to exploit these advantages in the temporal logic synthesis setting [11]. The framework relies on splitting the problem into several short horizons by partitioning the state-space. Realizability of the global problem is determined through symbolic checks of the specification, and the extraction of controllers for each of the smaller problems postponed until the respective partitions are reached.

The gain from this is twofold: the shorter problems will limit the number of events that have to be considered before the next problem is reached, and the controller extraction can be restricted to the current state, ignoring extraction for states that never occur. This method is effective at reducing the size of the synthesis problems, but is not enough in the presence of large environments with a lot of non-determinism. Although the controller extraction can be restricted to the choices the system will take towards its goal, all possible actions of the environment still have to be considered. This results in a computational blowup when the number of environment actions increases.

However, in each of the short horizon problems, usually only a small subset of the environment variables is actually interesting. This observation was previously addressed by manually restricting the parts of the environment that were deemed irrelevant in each of the short horizon problems [12]. This ad hoc method results in problems of a manageable size, but the guarantee of correctness is lost, as it is easy to accidentally restrict the environment in ways that oversimplify the problem.

Previous work on reducing specifications has resulted in tools to find *unhelpful* parts of a specification and *minimally sufficient specifications*. The authors of [3] define a notion of "helpful" signals, and iteratively remove unhelpful ones. However, their approach relies on expensive iterated realizability tests, which we circumvent in this paper. The authors in [6] use model-based diagnosis to remove irrelevant output signals from the specification; these are

[1] Mattias Fält is with the Department of Automatic Control at Lund University, SE-221 00 Lund, Sweden faldt.mattias@gmail.com

[2,3] Vasumathi Raman and Richard M. Murray are with the Department of Control and Dynamical Systems at the California Institute of Technology, Pasadena, CA vasu@caltech.edu, murray@cds.caltech.edu

This work was supported in part by TerraSwarm, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

output signals that can be set arbitrarily without affecting the unrealizability of the specification. Their approach also uses repeated realizability checks, and is unhelpful in the case of realizable specifications. Moreover, these approaches do not use any domain-specific knowledge to restrict the specifications, unlike that which we present in this paper.

In this paper, we present an algorithm that automatically identifies variables that can be ignored in each short horizon problem. We show how to modify the specification to ignore these variables without artificially restricting the environment. This enables us to solve problems that were previously unsolvable in a correct-by-construction manner. We validate the method on an autonomous car example with a dynamic environment. We also demonstrate how the reduced specifications can be used to identify when several short horizon problems are practically identical. This enables the reuse of previously synthesized controllers.

II. Preliminaries

A. Linear Temporal Logic

Syntax: Given a set of atomic propositions AP, an LTL formula is defined by the recursive grammar:

$$\varphi ::= \pi \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc\varphi \mid \varphi_1 \mathsf{U} \varphi_2$$

where $\pi \in AP$, $\vee$ and $\neg$ are the Boolean operators for disjunction and negation, and $\bigcirc$ and U are the temporal operators "next" and "until". From these operators, the following operators are derived: "disjunction" $\vee$, "implication" $\Rightarrow$, "equivalence" $\Leftrightarrow$, "always" $\Box$, and "eventually" $\Diamond$.

Semantics: LTL is interpreted over sequences of truth assignments $\sigma : \mathbb{N} \to 2^{AP}$. We say that a truth assignment σ satisfies $\pi \in AP$ at time t (denoted $(\sigma, t) \models \pi$) if $\pi \in \sigma(t)$, i.e. σ assigns π to `True` at time t. We say $(\sigma, t) \not\models \pi$ if π is assigned `False` at time t, i.e. $\pi \notin \sigma(t)$. The semantics of an LTL formula is then defined recursively over the structure of the formula. For example, σ satisfies "next" φ, denoted $\sigma \models \bigcirc\varphi$, if and only if φ is true in $\sigma(t+1)$. Similarly, $\sigma \models \Box\varphi$ and $\sigma \models \Diamond\varphi$ if and only if φ is true in $\sigma(t')$ for *all* $t' \geq t$ and for *some* $t' \geq t$, respectively. The reader is referred to [4] for the full syntax and semantics of LTL.

B. Reactive Synthesis

We let the system and environment state be characterized by a finite number of atomic propositions (also called boolean variables). We denote by $\mathcal{X}$ and $\mathcal{Y}$ the sets of variables that the environment and system can control respectively. The state of the environment and system at any time step is described by a truth assignment tuple $(x, y) \in X \times Y$ where $X = 2^{\mathcal{X}}$ and $Y = 2^{\mathcal{Y}}$. These states can be the result of discretization, as described in [8], [11], where the variables in $\mathcal{X}$ and $\mathcal{Y}$ abstract a continuous state space.

Given an LTL specification φ, the *reactive synthesis* problem is to find a finite-state strategy for the system that, at each time t, given $x_t, x_{t+1} \in X$ and $y_t \in Y$, provides $y_{t+1} \in Y$, such that the resulting infinite sequence of truth assignments $\sigma = (x_0, y_0), (x_1, y_1), \ldots$ satisfies φ at time 0 (i.e. $(\sigma, 0) \models \varphi$). If such a strategy exists, the specification is called *realizable*. Reactive synthesis for general LTL specifications is 2EXPTIME-complete [10], but [1] presents a tractable algorithm for the Generalized Reactivity(1) (GR(1)) fragment, which consists of specifications of the form

$$\Big(\psi_{init} \wedge \Box\psi_e \wedge \bigwedge_{i \in I_f} \Box\Diamond\psi_{f,i}\Big) \Longrightarrow \Big(\bigwedge_{i \in I_s} \Box\psi_{s,i} \wedge \bigwedge_{i \in I_g} \Box\Diamond\psi_{g,i}\Big), \tag{1}$$

where:

- ψ_e is a propositional formula over $\mathcal{X}, \mathcal{Y}$ and $\bigcirc\mathcal{X}$, where $\bigcirc AP = \{\bigcirc\pi \mid \pi \in AP\}$
- $\psi_{s,i}$ is a propositional formula over $\mathcal{X}, \mathcal{Y}, \bigcirc\mathcal{X}$ and $\bigcirc\mathcal{Y}$
- ψ_{init}, $\psi_{f,i}$ and $\psi_{g,i}$ are formulas over $\mathcal{X}$ and $\mathcal{Y}$.

The antecedent of this implication is referred to as the *assumptions*, and the consequent as the *guarantees*. It is based on this form of specification that the synthesis problem for the receding horizon framework was defined in [12].

C. Receding Horizon Temporal Logic Planning

To guarantee that it will still be possible to complete the task using a receding horizon, additional constraints are required on the execution. Formally, for each of the progress properties $\psi_{g,i}$, we require:

- A partitioning of $X \times Y = \mathcal{W}_0^i \cup \mathcal{W}_1^i \cup \ldots \cup \mathcal{W}_p^i$, such that $(x, y) \in \mathcal{W}_0^i$ only if $(x, y) \vDash \psi_{g,i}$.
- A partial order $(\preceq_{\psi_{g,i}})$ on partitions with $\mathcal{W}_0^i \preceq_{\psi_{g,i}} \mathcal{W}_j^i$ for all j.
- A mapping $\mathcal{F}^i : \{\mathcal{W}_0^i \ldots, \mathcal{W}_p^i\} \to \{\mathcal{W}_0^i \ldots, \mathcal{W}_p^i\}$ such that $\mathcal{F}^i(\mathcal{W}_j^i) \preceq_{\psi_{g,i}} \mathcal{W}_j^i$.
- A propositional formula Φ (called the *invariant*) over $\mathcal{X} \cup \mathcal{Y}$ such that $\psi_{init} \Longrightarrow \Phi$ is True.

The partial ordering of the partitions represents a measure of closeness to the progress property $\psi_{g,i}$, while the mapping $\mathcal{F}^i$ decides where to set the short horizon goal while ensuring that the system gets closer to fulfilling its progress property $\psi_{g,i}$. Lastly, the invariant represents the additional constraints required to ensure realizability when switching between short horizon problems. Formally, the following sufficient short horizon specification was proposed in [12]:

$$\begin{aligned} \Psi_j^i \;\doteq\; & \Big((\nu \in \mathcal{W}_j^i) \wedge \Phi \wedge \Box\psi_e \wedge \bigwedge_{k \in I_f} \Box\Diamond\psi_{f,k}\Big) \\ & \Longrightarrow \Big(\bigwedge_{k \in I_s} \Box\psi_{s,k} \wedge \Box\Phi \wedge \Diamond\left(\nu \in \mathcal{F}^i\left(\mathcal{W}_j^i\right)\right)\Big). \end{aligned} \tag{2}$$

If Φ can be constructed such that (2) is realizable for all $\psi_{g,i}, \mathcal{W}_j^i$, a controller for (1) can be constructed via reactive synthesis on these short horizon problems [12].

III. Problem

A. Scalability of receding horizon synthesis

The receding horizon method proposed by [11] enables solving a large problem in steps by dividing it into sub problems, and extracting a controller for each sub problem only when that problem is encountered. This method reduces

the running time for synthesizing a controller, since the initial state of a sub problem restricts the explored state space when extracting a controller for this part. A detailed discussion on this aspect can be found in [12].

Although the receding horizon method is effective in reducing the total size of controllers (and therefore synthesis time), it still scales poorly when the number of environment variables is large. In many states, the system behavior is independent of some of the environment actions. When the horizons are short, it is often the case that some environment variables do not affect the system within an entire short horizon. This is often true in robotics and path planning applications, where obstacles and environment actions in one part of the workspace are almost unrelated to those in another. In the examples presented in [12], this property was exploited to simplify each of the short horizon specifications. However, the simplification was achieved manually, by including only parts of the full specification that the user deemed relevant for each horizon. This is an ad hoc approach, and the correctness of the solution can no longer be guaranteed. We would therefore like a method of automatically reducing the number of variables in the short horizon specification while maintaining guarantees.

The variable elimination method proposed in this paper is motivated by the observation that only a fraction of environment variables are relevant within each short horizon problem, reducing the time taken to synthesize each solution. As we show in Section VI. this reduction moves many previously unsolvable correct-by-construction synthesis problems into the realm of computational tractability.

IV. Variable Reduction

Let $\mathcal{X} = \{e_1, e_2, ..., e_{n_1}\}$, $\mathcal{Y} = \{s_1, s_2, ..., s_{n_2}\}$ be the set of environment and system variables, respectively. We denote environment inputs by $x \in X = 2^{\mathcal{X}}$ and system outputs by $y \in Y = 2^{\mathcal{Y}}$, and a *state* is a tuple (x, y). For simplicity of notation, we suppress the index i that indicates the current horizon and rewrite (2) as follows:

$$\begin{aligned}\phi_j &= \varphi^e \Rightarrow \varphi^s \\ &= (\varphi_{init} \wedge \Phi \wedge \square\varphi_s^e \wedge \square\diamond\varphi_l^e) \Rightarrow \left(\square\Phi \wedge \square\varphi_s^s \wedge \diamond\varphi_p^s\right),\end{aligned} \tag{3}$$

where φ_s^s is a propositional formula over $\mathcal{X}, \mathcal{Y}, \bigcirc\mathcal{X}$ and $\bigcirc\mathcal{Y}$; φ_l^e is over $\mathcal{X}, \mathcal{Y}$ and $\bigcirc\mathcal{X}$; and $\varphi_l^e, \hat{\varphi}_{init}, \Phi$ and φ_p^s are over $\mathcal{X}$ and $\mathcal{Y}$.

Let $\mathcal{W} = \{X \times Y_j\}_{j\in[1,k]}$ where $\{Y_1, ..., Y_k\}$ is a partitioning of Y. Define $\mathcal{R} : \mathcal{W} \to 2^{\mathcal{W}}$ and $\mathcal{F} : \mathcal{W} \to \mathcal{W}$, where $\mathcal{W}_j, \mathcal{F}(\mathcal{W}_j) \in \mathcal{R}(\mathcal{W}_j)$ for all $\mathcal{W}_j \in \mathcal{W}$. $\mathcal{R}(\mathcal{W}_j)$ is the *plan set* for $\mathcal{W}_j$, and contains subsets of $\mathcal{W}$ that are relevant when constructing a plan over set $\mathcal{W}_j \in \mathcal{W}$. This *plan set* restricts the short horizon solution, and should be chosen as the smallest set such that the specifications can still be satisfied. Finding a restriction is often not hard, and an example is given in section VI-A.

For the rest of this section, we will view propositional formulas as functions mapping truth assignments to the variables on which they are defined to a value in $\{0, 1\}$ (denoting the corresponding truth value of the formula); we call these *Boolean functions*. For example, overloading notation, the formula Φ can be viewed as a function $\Phi : X \times Y \to \{0, 1\}$.

Definition 1: Let a Boolean variable x_i be in the *support* of a Boolean function $f(x_1, ..., x_n)$, iff

$$f(x_1, ..., x_i = 0, ..., x_n) \neq f(x_1, ..., x_i = 1, ..., x_n).$$

Definition 2: The *existential abstraction* of Boolean function f with respect to Boolean variable x_i is defined as

$$\exists_{x_i} f = f|_{x_i=0} \vee f|_{x_i=1}.$$

Existential abstraction with respect to sets of Boolean variables is defined naturally via iteration over the set.

Definition 3: Let the *supporting set* of a Boolean function f be the set of variables in the support of f, and the *non-supporting set* be the set variables not in the support of f. We denote this as $S(f)$ and $NS(f)$ respectively.
We define

$$Y_{\mathcal{R}(\mathcal{W}_j)} = \bigcup_{is.t.\mathcal{W}_i \in \mathcal{R}(\mathcal{W}_j)} Y_i,$$

and denote a function f restricted to $X \times Y_{\mathcal{R}(\mathcal{W}_j)}$ by $\tilde{f} = f|_{X\times Y_{\mathcal{R}(\mathcal{W}_j)}}$ (where the index j is implied from the context).

We will restrict the specification to relevant variables on the plan set for the current horizon. After restricting the specifications to the current plan set $\mathcal{R}(\mathcal{W}_j)$, we define sets

$$\begin{aligned}\mathcal{X}_{ns}^{s,s} &= \left\{e_i \in \mathcal{X} \,\middle|\, e_i, e_i' \in NS\left(\tilde{\varphi}_s^s \wedge \tilde{\Phi}\right)\right\}, \\ \mathcal{X}_{ns}^{s,p} &= \left\{e_i \in \mathcal{X} \,\middle|\, e_i \in NS\left(\tilde{\varphi}_p^s\right)\right\}\end{aligned}$$

of variables that do not affect the two guarantee portions of the specification; we can therefore eliminate the variables in $\mathcal{X}_{ns} = \mathcal{X}_{ns}^{s,s} \cap \mathcal{X}_{ns}^{s,p}$. Given f, let $\hat{f} = \exists_{\mathcal{X}_{ns}} \tilde{f}$, and denote by X_s its domain, i.e. the powerset of variables in $\mathcal{X}\backslash(\mathcal{X}_{ns})$. It is clear that existentially conditioning non-supporting variables does not change a function in the sense that $f(\mathcal{X}_s, \mathcal{X}_{ns}) \Leftrightarrow \exists_{\mathcal{X}_{ns}} f(\mathcal{X}_s)$.

Definition 4: The *reduced short horizon specification* is

$$\begin{aligned}\hat{\phi}_j &\doteq \hat{\varphi}^e \Rightarrow \hat{\varphi}^s \\ &\doteq \left(\hat{\varphi}_{init} \wedge \hat{\Phi} \wedge \square\hat{\varphi}_s^e \wedge \square\diamond\hat{\varphi}_l^e\right) \Rightarrow \left(\square\hat{\Phi} \wedge \square\hat{\varphi}_s^s \wedge \diamond\hat{\varphi}_p^s\right),\end{aligned} \tag{4}$$

where $\varphi_p^s = (\nu \in \mathcal{F}(\mathcal{W}_j))$, $\varphi_{init} = (\nu \in \mathcal{W}_j)$.

Lemma 1: For any infinite sequence of states $\sigma = \sigma_0\sigma_1...$ with $\sigma_i = ((x_{s,i}, x_{ns,i}), y_i) \in (X_s \times X_{ns}) \times Y_{\mathcal{R}(\mathcal{W}_j)}$, define $\sigma_s = \sigma_{s,0}\sigma_{s,1}...$ with $\sigma_{s,i} = (x_{s,i}, y_i)$. Then

$$\begin{aligned}\sigma \vDash (\varphi_{init} \wedge \Phi \wedge \square\varphi_s^e \wedge \square\diamond\varphi_l^e) \\ \Longrightarrow \sigma_s \vDash \left(\hat{\varphi}_{init} \wedge \hat{\Phi} \wedge \square\hat{\varphi}_s^e \wedge \diamond\hat{\varphi}_l^e\right)\end{aligned}$$

Proof: By definition, $\hat{\Phi} = \exists_{\mathcal{X}_s} \tilde{\Phi}$ with $\tilde{\Phi} = \Phi$ on the subspace $X \times Y_{\mathcal{R}(\mathcal{W}_j)}$. If $\Phi(\sigma_i) = \Phi((x_{s,i}, x_{ns,i}), y_i) = 1$ then it follows from the definition of existential abstraction that $\hat{\Phi}(\sigma_{s,i}) = \hat{\Phi}(x_{s,i}, y_i) = 1$. Analogously, the same applies to φ_{init}, φ_s^e and φ_l^e. Thus, since any state σ_i or transition (σ_i, σ_{i+1}) that satisfies $\varphi_{init}, \Phi, \varphi_s^e, \varphi_l^e$ results in the state $\sigma_{s,i}$ or transition $(\sigma_{s,i}, \sigma_{s,i+1})$ that satisfies $\hat{\varphi}_{init}, \hat{\Phi}, \hat{\varphi}_s^e, \hat{\varphi}_l^e$, the implication for the sequences follows. ■

Lemma 2: Given $\sigma_s = \sigma_{s,0}\sigma_{s,1}...$, with $\sigma_{s,i} = (x_{s,i}, y_i) \in X_s \times Y_{\mathcal{R}(\mathcal{W}_j)}$, for any $\sigma = \sigma_0\sigma_1...$ with $\sigma_i = ((x_{s,i}, x_{ns,i}), y_i) \in X \times Y$,

$$\sigma_s \vDash \left(\square\hat{\Phi} \wedge \square\hat{\varphi}_s^s \wedge \square\diamond\hat{\varphi}_p^s\right) \implies \sigma \vDash \left(\square\Phi \wedge \square\varphi \wedge \diamond\varphi_p^s\right).$$

Proof: Given a state $\sigma_{s,i} = (x_{s,i}, y_i) \vDash \hat{\varphi}_p^s$, from the definition of *existential abstraction* there has to exist $x_{ns,i} \in X_{ns}$ such that $((x_{s,i}, x_{ns,i}), y_i) \vDash \tilde{\varphi}_p^s$. But since $x_{ns,i} \in X_{ns}$, all corresponding variables must be in the *non-supporting set* $\mathcal{X}_{ns}^{s,p}$, and thus $\hat{\varphi}_p^s((x_{s,i}, x_{ns,i}), y_i) = \tilde{\varphi}_p^s((x_{s,i}, x_{ns,i}^{'}), y_i)$ for any $x_{ns,i}^{'} \in X_{ns}$. Since $\tilde{\varphi}_p^s = \varphi_p^s$ on $X \times Y_{\mathcal{R}(\mathcal{W}_j)}$, we have shown that $(x_{s,i}, y_i) \vDash \hat{\varphi}_p^s \Rightarrow ((x_{s,i}, x_{ns,i}), y_i) \vDash \varphi_p^s$ for any $x_{ns,i}$. The same argument can be made for the conjunction $\hat{\varphi}_s^s \wedge \hat{\Phi}$, and the implication is therefore true for the sequences σ_s and σ. ∎

Theorem 1: Realizability of the reduced short horizon specification (4) implies realizability of the short horizon specification (3). Moreover, a strategy for the reduced short horizon specification can be refined to a strategy for the short horizon specification.

Proof: If (4) is *realizable* then there exists a strategy $g : X_s^2 \times Y_{\mathcal{R}(\mathcal{W}_j)} \to Y_{\mathcal{R}(\mathcal{W}_j)}$ (or $g_i : X_s \times Y_{\mathcal{R}(\mathcal{W}_j)} \to Y_{\mathcal{R}(\mathcal{W}_j)}$ for the initial state) that given inputs x will generate an infinite sequence of states $\sigma_s = ((x_{s,0}, y_0), (x_{s,1}, y_1), ...$ with $y_{i+1} = g(x_{s,i}, x_{s,i+1}, y_i)$ such that σ_s satisfies equation (4). We now show how this strategy can be refined to one that satisfies equation (3). For an arbitrary set of states $x_i = (x_{s,i}, x_{ns,i}) \in X_s \times X_{ns} = X$, $x_{i+1} \in X$, $y_i \in Y$, define the strategy $h : X^2 \times Y \to Y_{\mathcal{R}(\mathcal{W}_j)}$ as

$$h(x_i, x_{i+1}, y_i) = g(x_{s,i}, x_{s,i+1}, y_i)$$

if $y_i \in Y_{\mathcal{R}(\mathcal{W}_j)}$, and arbitrarily otherwise. This strategy will generate a sequence $\sigma = \sigma_0\sigma_1...$ of states. Assume first that the sequence $\sigma \nvDash \varphi^e$, then $\varphi^e \Rightarrow \varphi^s$ imposes no restrictions and $\sigma \vDash \varphi^e \Rightarrow \varphi^s$. If $\sigma \vDash \varphi^e$, then since $\varphi_{init} = (\nu \in \mathcal{W}_j^i)$ we have that $y_0 \in Y_{\mathcal{R}(\mathcal{W}_j)}$ and since h is a function to $Y_{\mathcal{R}(\mathcal{W}_j)}$, we conclude that $y_i \in Y_{\mathcal{R}(\mathcal{W}_j)}$ for all i. Through Lemma 1 it follows that $\sigma_s \vDash \hat{\varphi}^e$, and because g is a strategy for the reduced short horizon specification, it follows that $\sigma_s \vDash \hat{\varphi}^s$. It is therefore clear from Lemma 2 that $\sigma \vDash \varphi^s$, and we have thus shown that $\sigma \vDash \varphi^e \Rightarrow \varphi^s$. Thus h is a strategy for (3). ∎

V. Application to Problem Classification

Short horizon problems often have very similar structure, and it is useful to identify and leverage this property. Using the method proposed in Section IV, each of the problems previously defined on $X \times Y$ will now be defined on the smaller subset $X_s \times Y_{\mathcal{R}(\mathcal{W}_j)}$, and over fewer variables $\mathcal{S} = \mathcal{X}_s \cup \mathcal{Y}$. We denote the set of Boolean functions that characterizes $\hat{\phi}_i$ by $f_j^i \in F^i = \left\{(\hat{\varphi}_{init} \wedge \hat{\Phi}), \hat{\varphi}_s^e, \hat{\varphi}_l^e, (\hat{\Phi} \wedge \hat{\varphi}_s^s), \hat{\varphi}_p^s\right\}$.

Given two problems $\hat{\phi}_1, \hat{\phi}_2$ defined on sets of variables $\mathcal{S}_1, \mathcal{S}_2$ respectively, if a mapping $\mathcal{M} : \mathcal{S}_1 \to \mathcal{S}_2$ exists such that $f_j^1 \circ \mathcal{M} = f_j^2$ for all $j \in \{1, ..., 5\}$, then a controller g_1 for $\hat{\phi}_1$ can be used as a controller $g_2 = g_1 \circ \mathcal{M}$ for $\hat{\phi}_2$.

```
1: procedure ClassifyProbs({φ̂_1, φ̂_2, ..., φ̂_n})
2:     E ← {}                                          ▷ Equivalence classes
3:     for 1 ≤ this ≤ n do                             ▷ Iterate over horizons
4:         c_1, ..., c_n ← VariableClassification(φ̂_this)
5:         for eq ∈ E do
6:             φ̂_other ← First(eq)
7:             if |c_i(φ̂_this)| ≠ |c_i(φ̂_other)| for any i then
8:                 continue
9:             end if
10:            for p ∈ Π_{c_1} × Π_{c_2} × ...Π_{c_n} do
11:                for f_j^{φ̂_this} ∈ F^{φ̂_this} do
12:                    if f_j^{φ̂_this} ≠ f_j^{φ̂_other} ∘ M_p then
13:                        Next p at line 10
14:                    end if
15:                end for
16:                Add φ̂_this to eq                    ▷ Mapping M_p works
17:                Next φ̂_this at line 3
18:            end for
19:                                                    ▷ No mapping exists for this eq
20:        end for
21:                                                    ▷ φ̂_this does not belong to any class eq
22:        Add new eq = {φ̂_this} to E
23:    end for
24: end procedure

25: procedure VariableClassification(φ^i)
26:    c_0, ..., c_k ← {}, {}, ..., {}
27:    for variable v ∈ S(φ_i) do
28:        val ← 0
29:        C = (S(f_1^i), S(f_2^i), ..., , S(f_5^i))    ▷ set sequence
30:        for 1 ≤ j ≤ |C| do
31:            if v ∈ C(j) then
32:                val ← val + 2^j                     ▷ encode that v ∈ C(j)
33:            end if
34:        end for
35:        Add v to class c_val                        ▷ v' ∈ c_val have same val
36:    end for
37:    return c_0, c_1..., c_k
38: end procedure
```

Fig. 1. Algorithm for Problem Classification. The algorithm starts by classifying the variables in each $\hat{\phi}_i$ by looking at which of the functions $f_j^i \in F^i$ they support. The algorithm then only considers bijections between problems with the same number of variables in each class. In this case there will be $2^{|F^i|} = 32$ variable classes.

Finding such a mapping $\mathcal{M}$ between variables is in general NP-hard. The algorithm in Fig. 1 presents a method that searches for such a mapping over the set of all bijections $\mathcal{M} : \mathcal{S}_1 \to \mathcal{S}_2$ (assuming $|\mathcal{S}_1| = |\mathcal{S}_2|$), while trying to minimize the number of bijections tested.

Given a set of variables $\mathcal{S} = \{s_1, s_2, ..., s_n\}$, we denote by $\Pi_{\mathcal{S}}$ the set of all permutations over $\mathcal{S}$ (corresponding to permutations over the set of integers in $[1, n]$. Given $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$, we abuse notation slightly by writing $\Pi_{\mathcal{S}_1} \times \Pi_{\mathcal{S}_2} = \{p_1p_2 \mid p_1 \in \Pi_{\mathcal{S}_1}, p_2 \in \Pi_{\mathcal{S}_2}\}$. Given $\mathcal{S}_1 = \{s_1^1, s_2^1, ..., s_n^1\}$ and $\mathcal{S}_2 = \{s_1^2, s_2^2, ..., s_n^2\}$, each permutation $p \in \Pi_{\mathcal{S}_1}$ defines a mapping $\mathcal{M}_p$ as $\mathcal{M}_p(s_i^1) = s_p^2$.

We incrementally construct a set $E = \{eq_1, eq_2, ..., eq_n\}$ of equivalence classes of short horizon problems such that for all $\hat{\phi}_1, \hat{\phi}_2 \in eq_i$, there is a permutation $p \in \Pi_{\mathcal{S}_1}$ such that $\hat{\phi}_2 = \hat{\phi}_1 \circ \mathcal{M}_p$. We make several optimizations:

- We compute supporting sets only once for each function $f_j^i \in F^i$, categorizing each variable into one of $2^{|F^i|} =$

$2^5 = 32$ classes c_i based on where it is present. We can immediately quit if the specifications do not have an equal number of variables in each class.

- Instead of considering all $|\mathcal{X}_s|! \cdot |\mathcal{Y}|!$ permutations of all variables, we consider only permutations of variables within each class (Π_{c_i}).
- We compare two Boolean functions using their Binary Decision Diagrams (BDDs) [2], which are unique for a fixed variable ordering. When computing $f^i_j \circ \mathcal{M}_p$, the BDD (or a hash of the BDD) can be saved for each permutation p and each $f^i_j \in F$ for a single representative function $\hat{\phi}_i \in eq_i$ of every equivalence class $eq_i \in E$. It is then sufficient to compare the BDD of f^i_j to all the previously-saved BDDs of $f^i_j \circ \mathcal{M}_p$ when looking for a mapping between $\hat{\phi}_i$ and $\hat{\phi}_k$. This greatly reduces the running time, since two BDDs can be compared in time linear in the number of variables.
- Specifications are usually built via an automated process. Equivalent specifications therefore usually have corresponding variables in the same locations in the specification, and the first variable in the first specification usually corresponds to the first variable in the second specification, and so on. This fact can be used to greatly reduce the number of mappings tried.
- This method can be extended to categorize variables based on other characteristics, such as if a variable is *essential* for a function (i.e. if $f \Leftrightarrow f \wedge e_i$).

VI. Results

The proposed methods were implemented and tested in the Temporal Logic Planning Toolbox (TuLiP) [13]. We present a simple demonstrative problem inspired by the road example previously used for the receding horizon framework [11].

A. Example

We consider a discretized road with a bend, as depicted in Fig. 2. We let the discrete length (in "cells") before and after the bend be n and m, and index the road segments (each of which are 2 cells wide) by i, corresponding to their distance from the starting cell. We denote by $O_{i,l}$ and $O_{i,r}$ the presence of an obstacle in the left and right cells of segment i, respectively, and the location of the car in road segment i by $Y_{i,l}$ and $Y_{i,r}$. There might be obstacles at any location, with a few restrictions:

- they may not block the road completely:

$$\bigwedge_{i \in I} \neg(O_{i,l} \wedge O_{i,r}) \wedge \neg(O_{i,l} \wedge O_{i+1,r}) \wedge \neg(O_{i+1,l} \wedge O_{i,r}),$$

- they may not appear or disappear while the car is nearby:

$$\bigwedge_{\substack{i \in I \\ k \in [i-1, i+1]}} (Y_{i,l} \vee Y_{i,r}) \Rightarrow ((O_{k,l} \Leftrightarrow \bigcirc O_{k,l}) \wedge (O_{k,r} \Leftrightarrow \bigcirc O_{k,r})),$$

- they cannot block the turn: $\neg(O_{n-1,r} \wedge O_{n,r})$.

The goal for the car is to get to $\psi_g = Y_{n+m,l} \vee Y_{n+m,r}$ from $\psi_{init} = Y_{0,l} \vee Y_{0,r}$. It may move to adjacent cells but not diagonally (this assumption is implicit in the rest of the paper), and may never be in the same cell as an obstacle:

$$\bigwedge_{i \in I} \neg((Y_{i,l} \wedge O_{i,l}) \vee (Y_{i,r} \wedge O_{i,r})).$$

This global specification is thus defined over the variables $\mathcal{X} = \cup_{i \in I}(O_{i,l} \cup O_{i,r})$ and $\mathcal{Y} = \cup_{i \in I}(Y_{i,l} \cup Y_{i,r})$ with implicit mutual exclusion over the system variables. For the receding horizon framework we choose the partitioning $\mathcal{W}_i = 2^{\mathcal{X}} \times 2^{\{Y_{i,l}, Y_{i,r}\}}$ and the mapping $\mathcal{F}(\mathcal{W}_i) = \mathcal{W}_{\min(i+h, max(I))}$ where h is the horizon (as a number of discrete cells). Lastly, we choose $\mathcal{R}(\mathcal{W}_i) = \{\mathcal{W}_i, \mathcal{W}_{i+1}, ..., \mathcal{F}(\mathcal{W}_i)\}$. Fig. 2 illustrates these sets.

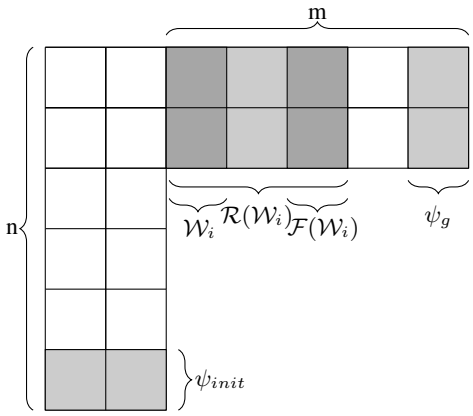

Fig. 2. Example illustrating the road example of size n=6, m=7. $\psi_{init}$and ψ_g represents the initial condition and goal of the full specification. The figure also illustrates $\mathcal{W}_i, \mathcal{F}(\mathcal{W}_i), \mathcal{R}(\mathcal{W}_i)$ for a specific short horizon problem with horizon 2.

TABLE I
Results for the double road without receding horizon framework

length	states	storage for extraction (array size)
3	163	<50000
4	698	>50000
5	2827	>400000
6	?	>3200000

B. Regular road, no receding horizon

For comparison, we ran the GR(1) synthesis algorithm on this example without using the receding horizon framework. This problem quickly grows beyond what is possible to solve because of the large number of available environment transitions in each step. Table I shows the number of states in the synthesized controllers for different road lengths, and the approximate array sizes needed to store intermediate fixpoints for strategy extraction. The solver crashed because of excessive memory usage in the last example.

C. Double road, horizon 1

The receding horizon framework reduces the number of states in the controllers significantly, as seen in table II. t_{synth} is the total time (in seconds) for synthesizing all controllers, which seems to increase linearly with the length. Here t_{reduce} is the time spent reducing the short horizon

TABLE II

RESULTS FOR THE DOUBLE ROAD WITH HORIZON 1

n	m	t_{synth}	t_{reduce}	$state_{max}$	$state_{med}$	$state_{tot}$
3	3	11s	22s	230	45	410
4	4	13s	47s	237	45	505
5	5	14s	84s	197	45	552
2	9	15s	109s	197	45	597
2	10	15s	109s	197	45	642
10	10	22s	568s	231	45	1012

TABLE III

RESULTS FOR THE DOUBLE ROAD WITH HORIZON 2

n	m	t_{synth}	t_{reduce}	$state_{max}$	$state_{med}$	$state_{tot}$
5	5	43s	89s	303	303	1827
2	9	51s	112s	303	303	2179
10	10	105s	571s	303	303	4440

problems using the methods in this paper, and $state_{max}$, $state_{med}$ and $state_{tot}$ are the maximum, median and sum of the number of states over all short horizon problems. The number of states in each short horizon problem is constant after reduction, as expected.

D. Double road, horizon 2

Table III shows results for horizon $h = 2$. The number of states quickly increases with the horizon, as expected. However, the time taken using variable elimination grows much more slowly than the time taken without, and we are able to solve much larger problems.

E. Problem classification

By using the algorithm in Fig. 1, we were able to classify most of the short horizon problems into a small number of equivalence classes, and can therefore check realizability or extract a controller for just one problem per class. The method produced 5 different classes in the case of $h = 1$, one for the first partition, one for the last, two for the partitions on the turn, and one for all other problems. An illustration of the classification is shown in Figure 3. The resulting reduced short horizon specification for the majority of the specifications can be described by the following formulas

$$\begin{aligned}
\hat{\varphi}_{init} \wedge \hat{\Phi} &= (Y_{k,l} \vee Y_{k,r}) \wedge \neg C_k \\
\hat{\varphi}^e_s &= \bigwedge_{i \in \{k,k+1\}} (O_{k,l} \Leftrightarrow \bigcirc O_{k,l}) \wedge (O_{k,r} \Leftrightarrow \bigcirc O_{k,r}) \\
\hat{\varphi}^e_l &= \texttt{True} \\
\hat{\varphi}^s_s \wedge \hat{\Phi} &= \neg C_k \wedge \neg C_{k+1} \\
\hat{\varphi}^s_l &= Y_{k+1,l} \vee Y_{k+1,r},
\end{aligned}$$

where k is the starting partition and $C_i = (Y_{i,l} \wedge O_{i,l}) \vee (Y_{i,r} \wedge O_{i,r})$ denotes a crash in partition i. We have thus reduced the set of environment variables $\mathcal{X}$ to the set $\mathcal{X}_s = \{O_{k,l}, O_{k,r}, O_{k+1,l}, O_{k+1,r}\}$. The mappings between two specifications at index i and j is simply $\mathcal{M}(O_{i,l}) = O_{j,l}$ and analogously for the other variables $O_{i,r}, Y_{i,l}, Y_{i,r}$.

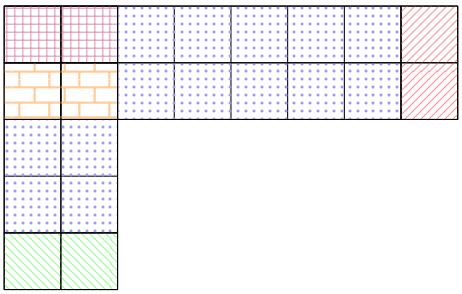

Fig. 3. Illustration of the five different categories of problems identified. Each horizon is identified as equivalent to the others except for the horizons in the start, turn and end.

VII. CONCLUSION

We have developed a method that greatly reduces the size of the short horizon problems in a receding horizon framework for temporal logic synthesis. This improves scalability without compromising the correctness of controllers. We have also shown how it is possible to identify and reuse controllers for several short horizon problems. Future work will improve efficiency of the problem classification method using domain-specific heuristics, and better study the tradeoffs involved in performing this procedure instead of synthesizing each horizon separately.

REFERENCES

[1] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. *Journal of Computer and System Sciences*, 78(3):911 – 938, 2012. In Commemoration of Amir Pnueli.

[2] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, C-35(8):677–691, Aug 1986.

[3] Alessandro Cimatti, Marco Roveri, Viktor Schuppan, and Andrei Tchaltsev. Diagnostic information for realizability. In *VMCAI*, pages 52–67, Berlin, Heidelberg, 2008. Springer-Verlag.

[4] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 1999.

[5] Marius Kloetzer and Calin Belta. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Transaction on Automatic Control*, 53(1):287–297, 2008.

[6] Robert Könighofer, Georg Hofferek, and Roderick Bloem. Debugging unrealizable specifications with model-based diagnosis. In *Haifa Verification Conference*, pages 29–45, 2010.

[7] Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, 2009.

[8] Jun Liu and Necmiye Ozay. Abstraction, discretization, and robustness in temporal logic control of dynamical systems. In *HSCC*, pages 293–302, New York, NY, USA, 2014. ACM.

[9] Pierluigi Nuzzo, Huan Xu, Necmiye Ozay, John B. Finn, Alberto L. Sangiovanni-Vincentelli, Richard M. Murray, Alexandre Donzé, and Sanjit A. Seshia. A contract-based methodology for aircraft electric power system design. *Access, IEEE*, PP(99):1–1, 2013.

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

[11] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M. Murray. Receding horizon temporal logic planning for dynamical systems. In *CDC*, pages 5997–6004, Dec 2009.

[12] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M. Murray. Receding horizon temporal logic planning. *Automatic Control, IEEE Transactions on*, 57(11):2817–2830, Nov 2012.

[13] Tichakorn Wongpiromsarn, Ufuk Topcu, Necmiye Ozay, Huan Xu, and Richard M. Murray. Tulip: A software toolbox for receding horizon temporal logic planning. In *HSCC*, pages 313–314, New York, NY, USA, 2011. ACM.