Submitted, 2015 American Control Conference (ACC)
http://www.cds.caltech.edu/~murray/papers/frm15-acc.html

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 with the increasing size of the environment. A receding horizon framework was recently proposed to mitigate 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, using the current state of the environment as the initial condition. This reduces the size of the synthesized strategy, but does not provide much improvement for problems with large dynamic environments, because the large number of possible global environment strategies. Ad-hoc methods to locally restrict the environment have previously been used, at the risk of losing correctness. This paper presents a method of reducing specifications by eliminating locally redundant variables, while maintaining the correctness of 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 used in solving more and more problems 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 the problems 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 [4], [6], [10], [8]. To apply these specifications to real systems, the system and 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 have been presented on how to do so. In particular, efficient methods have been developed for the Generalized Reactivity (GR(1)) fragment of the LTL language. These methods are based on finding a strategy to 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 been used in many other areas where solving the full problem at once is too hard. The method is shown to be effective not only in terms of complexity, but also in robustness and stability. A receding horizon framework was therefore recently introduced to take advantage of these properties in the temporal logic setting [11]. The framework relies on splitting the problem into several short horizon problems by partitioning the state-space. Realizability of the global problem can be determined through symbolic checks of the specifications, 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 limits the number of events that have to be considered before the next horizon is reached, and the controller extraction can be restricted to the current state, ignoring extraction for the cases that never occur. Even though this method is effective at reducing the complexity of synthesis, it is not enough in the presence of large environments with many possible actions. Although the extraction of the controller can be restricted to the choices the system will take towards its goal, all possible actions of the environment still have to be considered at every step. This results in a computational blowup when the number of environment variables 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. 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 over-simplify 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 [2] 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 [5] use model-based diagnosis to remove

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

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

irrelevant output signals from the specification; these are 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 problems. We also 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 demonstrate the method on an autonomous car example with a dynamic environment. We also demonstrate how the reduced specifications can be used to identify cases in which several short horizon problems are practically identical. This was not possible when the problems were defined over all variables, and it enables the possibility of reusing previously resynthesized controllers.

II. Preliminaries

A. Linear Temporal Logic

Syntax: Given a set of atomic propositions AP, boolean operators for negation ($\neg$), conjunction ($\wedge$), and disjunction ($\vee$), and temporal operators next ($\bigcirc$), always ($\Box$) and eventually ($\Diamond$), an LTL formula is defined by the recursive grammar:

$$\varphi ::= \pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc\varphi \mid \Box\varphi \mid \Diamond\varphi.$$

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 according to the following rules

- $(\sigma, t) \models \neg\varphi$ iff $(\sigma, t) \not\models \varphi$
- $(\sigma, t) \models \varphi \wedge \psi$ iff $(\sigma, t) \models \varphi$ and $(\sigma, t) \models \psi$
- $(\sigma, t) \models \varphi \vee \psi$ iff $(\sigma, t) \models \neg(\neg\varphi \wedge \neg\psi)$
- $(\sigma, t) \models \bigcirc\varphi$ iff $(\sigma, t+1) \models \varphi$
- $(\sigma, t) \models \Diamond\varphi$ iff $\exists t' \geq t$ s.t $(\sigma, t') \models \varphi$
- $(\sigma, t) \models \Box\varphi$ iff $(\sigma, t) \models \neg\Diamond(\neg\varphi)$

Note that in this paper, we omit the definition of the until operator. The reader is referred to [3] for the full syntax and semantics of LTL.

B. Reactive Synthesis

We let the system and environment state be characterised by a finite number of atomic propositions (also called boolean variables). We say that $\mathcal{X}$ and $\mathcal{Y}$ are the sets of variables that the system and environment can control respectively, and let the the state of the system and environment at any time step be 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 [7], [11] where the variables $\mathcal{X}$ and $\mathcal{Y}$ are propositions that 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 $x_t \in X$ and $y_t \in Y$, provides $x_{t+1} \in X$, such that the resulting infinite sequence of truth assignments $\sigma = (x_0, y_0), (x_1, y_1), ...$ satisfies φ at time 0 (i.e. $(\sigma, 0) \models \varphi$).

Reactive synthesis for a general LTL specification is 2EXPTIME-complete [9] hard, 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 propositional formulas over $\mathcal{X}$ and $\mathcal{Y}$.

The left hand side of this expression is referred to as the *assumptions*, and the right as the *guarantees* part. It is based on this form that the synthesis problem for the receding horizon framework is defined [10].

C. Receding Horizon Temporal Logic Planning

To guarantee that it will still be possible to complete the task when solving the problem using a receding horizon, and thus only looking a few steps ahead at a time, additional constraints on the execution may be required. Formally, for each of the progress properties $\psi_{g,i}$, the required parts are as follows:

- A partitioning of $X \times Y = \mathcal{W}_0^i \cup \mathcal{W}_1^i \cup ... \cup \mathcal{W}_p^i$, such that a state $(x, y) \in \mathcal{W}_0^i$ only if $(x, y) \vDash \psi_{g,i}$, and partial ordering ($\preceq_{\psi_{g,i}}$) of these partitions, such that $\mathcal{W}_0^i \preceq_{\psi_{g,i}} \mathcal{W}_j^i$ for all j.
- A mapping $\mathcal{F}^i : \{\mathcal{W}_0^i ..., \mathcal{W}_p^i\} \to \{\mathcal{W}_0^i ..., \mathcal{W}_p^i\}$ such that $\mathcal{F}^i(\mathcal{W}_j^i) \preceq_{\psi_{g,i}} \mathcal{W}_j^i$.
- A propositional formula Φ of variables from $\mathcal{X}, \mathcal{Y}$ 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 specifications was proposed in Wongpiromsarn *et al.* [11]:

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

If the specification above can be guaranteed for all partitions $\mathcal{W}_j^i$, all progress properties $\psi_{g,i}$, and for a common Φ, then a controller for the full specification (1) can be constructed by only solving these short horizon problems [10].

III. Problem

A. Complexity of receding horizon problems

The receding horizon method proposed by [11] enables solving a large problem in steps by dividing it into sub problems and synthesizing a controller for each sub problem first when that problem is encountered. This method reduces the complexity of synthesizing a controller since the initial state when entering a sub problem can be taken into consideration when synthesizing a controller for this part. This effectively reduces the number of configurations that has to be taken into consideration in each controller and the number of states in the controllers is greatly reduced by short horizons. For example, if a general problem has $|\mathcal{Y}|$ system variables and $|\mathcal{X}|$ environment variables, a controller might have up to $2^{|\mathcal{Y}||\mathcal{X}|}$ states. But if the initial position is known and the possible transitions for the environment is limited in each step by N_E and a solution can be guaranteed to be reached in N transitions, the maximum size of a controller can be bounded by $(N_E)^N$. However, if the problem is solved using the receding horizon method, with a total of N_H horizons and a maximum number of steps M in each horizon, the upper bound on the number of states in all controllers is reduced to $N_H \cdot (N_E)^M$. This is a great reduction in complexity if M is smaller than N. It might be unintuitive that the environment and its behavior would be so important to the complexity of the controller. The reason why this might be the case is because in each step, the environment may take any of N_E transitions, all of which have to be included in a controller, however, we only need to include one of the valid states that the system can go to in each step.

B. Previous approach

Although the receding horizon method is effective in reducing the total size of controllers (and therefore synthesis), there is still a problem with the complexity when N_E is large. It is often the case that several environment actions will result in the same action from the system. When the horizons are short, it could even be that some environment variables do not have to affect the system at all within several of the horizons, (this is especially true in applications in robotics and path planning where obstacles and environment actions in one location are almost unrelated to actions in another location). In previous examples, such as the one presented in Wongpiromsarn *et al.* [10], this was used to simplify each of the short horizon specifications. However, this simplification was done manually by including only the parts of the full specification that the user deemed relevant in each of the horizons. This is an *ad-hoc* approach, and the correctness of the solution can no longer be guaranteed. We therefore propose a method of reducing the number of variables in the short horizon specification while maintaining guaranteed correctness of the controller.

C. Reduction through variable elimination

The idea of the variable elimination method proposed in this paper is that only a fraction of the environment variables are relevant within each of the short horizon problems. If each of the problems can be solved by only considering N_e instead of N_E variables, the complexity is reduced to $N_H \cdot (N_e)^M \ll N_H \cdot (N_E)^M$. This reduction can move a large class of previously unsolvable correct-by-construction problems into the realm of computationally tractable problems. The complexity of controllers no longer have to increase as fast when adding problems together with partially disjoint environments.

IV. Approach

A. 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 system and environment variables respectively. We denote environment inputs by $x \in 2^{\mathcal{X}} = X$ and system outputs by $y \in 2^{\mathcal{Y}} = Y$, and a *state* is a truth assignment to both system and environment variables, i.e. (x, y).

Let the short horizon specification be

$$\begin{aligned}\phi_j^i &= \varphi^e \to \varphi^s \\ &= (\psi_{init} \wedge \Phi \wedge \Box\varphi_s^e \wedge \Diamond\varphi_l^e) \to \left(\Box\Phi \wedge \Box\varphi_s^s \wedge \Diamond\varphi_p^s\right),\end{aligned} \tag{3}$$

where the specification φ_s^s is of the form

$$\varphi(x, x', y, y') : X^2 \times Y^2 \to \{0, 1\},$$

φ_s^e is of the form

$$\varphi(X, X', Y) : X^2 \times Y \to \{0, 1\},$$

and φ_l^e, Φ, $\hat{\varphi}_{init}$ are of the form

$$\varphi(X, Y) : X \times Y \to \{0, 1\}.$$

Let $\mathcal{R} : \mathcal{W} \to 2^{\mathcal{W}}$ and $\mathcal{F} : \mathcal{W} \to \mathcal{W}$, where $\mathcal{W}, \mathcal{F}(\mathcal{W}) \in \mathcal{R}(\mathcal{W})$ and $\mathcal{W} = \{X \times Y_j\}_{j\in[1,k]}$ where $\{Y_1, ..., Y_k\}$ is some partitioning of the space Y.

Definition 1: Let a Boolean variable x_i be in the *support* of a 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 a 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 through iteration over variables in the set.

Definition 3: Let the *supporting set* of a 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.

Definition 4: We let $Y_{\mathcal{R}(\mathcal{W}_j)}$ be the domain $\bigcup Y_i$ for i such that $\mathcal{W}_i \in \mathcal{R}(\mathcal{W}_j)$, and denote a function f restricted to $X \times Y_{\mathcal{R}(\mathcal{W}_j)}$ as $\tilde{f} = f|_{X\times Y_{\mathcal{R}(\mathcal{W}_j)}}$ (where the index j is understood from the context).

We want to reduce the specification to only include relevant variables on the current horizon. After restricting the specifications to the current *plan set* $\mathcal{R}(\mathcal{W}_j)$, we define the 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 parts of the specification. We let the intersection of these sets be $\mathcal{X}_{ns} = \mathcal{X}_{ns}^{s,s} \cap \mathcal{X}_{ns}^{s,p}$. These are the variables that we want to remove from the specifications. We then denote a function $\tilde{f}$ existentially conditioned with respect to $\mathcal{X}_{ns}$ as $\hat{f} = \exists_{\mathcal{X}_{ns}} \tilde{f}$ and let the resulting space, the powerset of variables in $\mathcal{X}\backslash(\mathcal{X}_{us})$, be denoted X_s.

It is clear that existentially conditioning non-supporting variables does not change a function in the sense that $f(\mathcal{X}_s, \mathcal{X}_{ns}) \Leftrightarrow \exists_{\mathcal{X}_{ns}} f(\mathcal{X}_s)$. We can therefore conclude that $\hat{\varphi}_s^s = \varphi_s^s$, $\hat{\varphi}_p^s = \varphi_p^s$ and $\hat{\Phi} = \Phi$ on the subspaces $X_s^2 \times Y_{\mathcal{R}(\mathcal{W}_j)}^2$ and $X_s \times Y_{\mathcal{R}(\mathcal{W}_j)}$ respectively.

Definition 5: We define the *reduced short horizon problem* as

$$\begin{aligned}\hat{\phi}_j^i &= \hat{\varphi}^e \rightarrow \hat{\varphi}^s \qquad (4)\\ &= \left(\hat{\varphi}_{init} \wedge \hat{\Phi} \wedge \Box\hat{\varphi}_s^e \wedge \Diamond\hat{\varphi}_l^e\right) \rightarrow \left(\Box\hat{\Phi} \wedge \Box\hat{\varphi}_s^s \wedge \Diamond\hat{\varphi}_p^s\right),\end{aligned}$$

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

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

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

Proof: From the definition of $\hat{\Phi}$ we have that $\hat{\Phi} = \exists_{\mathcal{X}_s} \tilde{\Phi}$ 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, 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 an infinite sequence of states $\sigma_s = \sigma_{s,0}\sigma_{s,1}...$, with $\sigma_{s,i} = (x_{s,i}, y_i) \in X_s \times Y_{\mathcal{R}(\mathcal{W}_j)}$

$$\begin{aligned}&\sigma_s \vDash \left(\Box\hat{\Phi} \wedge \Box\hat{\varphi}_s^s \wedge \Diamond\hat{\varphi}_p^s\right)\\ &\quad\Longrightarrow \sigma \vDash \left(\Box\Phi \wedge \Box\varphi \wedge \Diamond\varphi_p^s\right)\end{aligned}$$

for any sequence $\sigma = \sigma_0\sigma_1...$ with $\sigma_i = ((x_{s,i}, x_{ns,i}), y_i) \in X \times Y$.

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 one evaluation $x_{ns,i} \in X_{ns}$ 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 e_i 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* $x_{ns,i}^{'}$. *Since* $\tilde{\varphi}_p^s = \varphi_s^s$ *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 problem (4) implies realizability of the short horizon problem (3). Moreover, a strategy for the reduced short horizon problem can be used as a strategy for the short horizon problem.

Proof: If the reduced short horizon problem (4) is *realizable* then there exists a strategy $g : X_s^2 \times Y_{\mathcal{R}(\mathcal{W}_j)} \rightarrow Y_{\mathcal{R}(\mathcal{W}_j)}$ (or $g_i : X_s \times Y_{\mathcal{R}(\mathcal{W}_j)} \rightarrow Y_{\mathcal{R}(\mathcal{W}_j)}$ for the initial state) that given inputs x will generate an infinite sequence of states $\sigma_s = ((x_0, y_0), (x_1, y_1), ...$ 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 used to satisfy 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 $h : X^2 \times Y \rightarrow Y_{\mathcal{R}(\mathcal{W}_j)}$ 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 $y_0 \in \mathcal{Y}_{\mathcal{R}(\mathcal{W}_j)}$ 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 $\sigma \vDash \varphi^e \rightarrow \varphi^s$. This means that h is a strategy for the *short horizon problem* (3) and because this strategy exists we conclude that the problem is *realizable*. ∎

V. Application to Problem Classification

It is reasonable to believe that several short horizon problems might be of very similar structure. However, before any reduction of the short horizon problems are done, this is very hard to check and to take advantage of. Using the method proposed in this paper, each of the problems ϕ_j^i, previously defined on $X \times Y$ will now be defined on the smaller subset $X_s \times Y_{\mathcal{R}(\mathcal{W}_j)}$ and thus with fewer variables $\mathcal{S} = \mathcal{X}_s \cup \mathcal{Y}$. Given two such problems $\hat{\phi}_1, \hat{\phi}_2$ defined on $\mathcal{S}_1, \mathcal{S}_2$, if a mapping $\mathcal{M} : \mathcal{S}_1 \rightarrow \mathcal{S}_2$ exists such that $f_1 \circ \mathcal{M} = f_2$ for all $f \in \left\{(\hat{\varphi}_{init} \wedge \hat{\Phi}), \hat{\varphi}_s^e, \hat{\varphi}_l^e, (\hat{\Phi} \wedge \hat{\varphi}_s^s), \hat{\varphi}_p^s\right\}$, then a controller g_1 for $\hat{\phi}_1$ can be used as a controller $g_2 = g_1 \circ \mathcal{M}$ for $\hat{\phi}_2$. Finding such a mapping $\mathcal{M}$ is in general hard, but we have implemented a method in the special case where a bijection $\mathcal{M}$ exists. The method relies on testing all bijections $\mathcal{M} : \mathcal{S}_1 \rightarrow \mathcal{S}_2$, and although this is in general is a very demanding method, it proves useful in practice by reducing the number of bijections that actually has to be tested.

1) Algorithm: We want to find the set E of categories of short horizon problems such that for all $\hat{\phi}_1, \hat{\phi}_j \in eq_i \in E$ a mapping exists such that $\hat{\phi}_j = \hat{\phi}_1 \circ \mathcal{M}$. Let F be the set of functions that characterizes each of the short horizon problems $F = \left\{(\hat{\varphi}_{init} \wedge \hat{\Phi}), \hat{\varphi}_s^e, \hat{\varphi}_l^e, (\hat{\Phi} \wedge \hat{\varphi}_s^s), \hat{\varphi}_p^s\right\}$. We can then find the equivalence classes eq_i through the

1: **procedure** PROBLEM CLASSIFICATION($\{\phi_j^i\}$)
2: $E \leftarrow \{\}$ ▷ Found equivalence classes
3: **for** $\phi_{this} \in \{\phi_j^i\}$ **do** ▷ All short horizon problems
4: $c_1, ..., c_n \leftarrow$ `VariableClassification`(ϕ_{this})
5: **for** $eq \in E$ **do**
6: $\phi_{other} \leftarrow$ `First`(eq)
7: **if** $|c_i(\phi_{this})| \neq |c_i(\phi_{other})|$ for any i **then**
8: **Next** eq at line 5
9: **end if**
10: **for** all orderings $\mathcal{M} = \mathcal{M}_1\mathcal{M}_2...\mathcal{M}_n$, where $\mathcal{M}_i$ is a permutation of the variables in c_i **do**
11: **for** $f \in F$ **do**
12: **if** $f_{\phi_{this}} \neq f_{\phi_{other}} \circ \mathcal{M}$ **then**
13: **Next** $\mathcal{M}$ at line 10
14: **end if**
15: **end for**
16: Add ϕ_{this} to eq ▷ Mapping $\mathcal{M}$ found
17: **Next** ϕ_{this} at line 3
18: **end for**
19: **Next** eq at 5▷ No mapping exists for this eq
20: **end for**
21: ▷ ϕ does not belong to any class eq
22: Add new $eq = \{\phi_{this}\}$ to E
23: **end for**
24: **end procedure**
25: **procedure** VARIABLE CLASSIFICATION($\{\phi\}$)
26: $c_0, ..., c_k \leftarrow \{\}, \{\}, ..., \{\}$
27: **for** variable $v \in S(\phi)$ **do**
28: $val \leftarrow 0$
29: **for** set $s_j \in \{\mathcal{X}_s, S(f_1), S(f_2), ..., S(f_n)\}$ **do**
30: **if** $v \in s_j$ **then**
31: $val \leftarrow val + 2^j$
32: **end if**
33: **end for**
34: Add v to class c_{val}
35: **end for**
36: **return** $c_0, c_1..., c_k$
37: **end procedure**

Problem Classification procedure above. There are several reasons why this method is effective:

- We compute the supporting sets only once for each short horizon problem and function $f \in F$, which categorizes each variable into one of $2^{2\cdot|F|} = 2^{10} = 1024$ groups. We then only consider mappings if the specifications have compatible types of variables, that is equal number of variables in each class c_i.
- Instead of considering all $|\mathcal{X}_s|! \cdot |\mathcal{Y}|!$ permutations of all variables, we consider only permutations of variables within each class ($\Pi_i(|c_i|!)$).
- We compare two functions by computing and comparing their Binary Decision Diagrams (BDDs) since the BDD of a function is unique for a fix variable ordering. When computing $f_1 \circ \mathcal{M}$, the BDD (or a hash of the BDD) can be saved for each permutation $\mathcal{M}$ and each $f \in F$ for the first function $\hat{\phi}_1$ of every equivalence class in E. It is then sufficient to compare the BDD of f_2 to all the BDDs of $f_1 \circ \mathcal{M}$, which are saved, when looking for a mapping. This greatly reduces the complexity, since comparing two BDDs can be done in linear time once the BDDs are computed!
- Specifications are usually built using some algorithm. 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 mappings that will be tried.
- This method could be extended to categorize variables based on on other characteristics too. Such as if a variable is *essential* for a function (e_i *essential* if $f \Leftrightarrow f \wedge e_i$).

VI. RESULTS

A. Example

The proposed methods were implemented and tested in the Temporal Logic Planning Toolbox (TuLiP) [12]. We chose a simple problem inspired by the road example previously used for the receding horizon framework [11]. We consider a discrete road of width 2 with a bend. We let the length before and after the bend be n and m. At each location of the road there might be an obstacle O. We denote an obstacle at distance i from the start as $O_{i,l}, O_{i,r}$ at the left and right location respectively. We denote the position of the car with $X_{i,l}$ and $X_{i,r}$ analogously, and let the set of distances from the start be I. 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:

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

and they cannot block the turn $\neg(O_{n-1,r} \wedge O_{n-1,r})$. The goal for the car is to get to $\psi_g = X_{n+m,l} \vee X_{n+m,r}$ by starting in $\psi_{init} = X_{0,l} \vee X_{0,r}$. It may move to adjacent squares but not diagonally (implicitly assumed for the rest of the paper), and it may never be in the same square as an obstacle:

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

This system is thus defined over the variables $\mathcal{X} = \cup_{i\in I}(O_{i,l} \cup O_{i,r})$ and $\mathcal{Y} = \cup_{i\in I}(X_{i,l} \cup X_{i,r})$ with the implicit mutual exclusion over the system variables. For the receding horizon framework we choose the partitioning $\mathcal{W}_i$ as the sets $2^{\mathcal{X}} \times 2^{\{X_{i,l},X_{i,r}\}}$, the mapping $\mathcal{F}(\mathcal{W}_i) = \mathcal{W}_{\min(i+h,max(I))}$ where h is the horizon. Lastly, we choose $\mathcal{R}(\mathcal{W}_i) = \{\mathcal{W}_i, \mathcal{W}_{i+1}, ..., \mathcal{F}(\mathcal{W}_i)\}$. An illustration of the problem is seen in Figure 1.

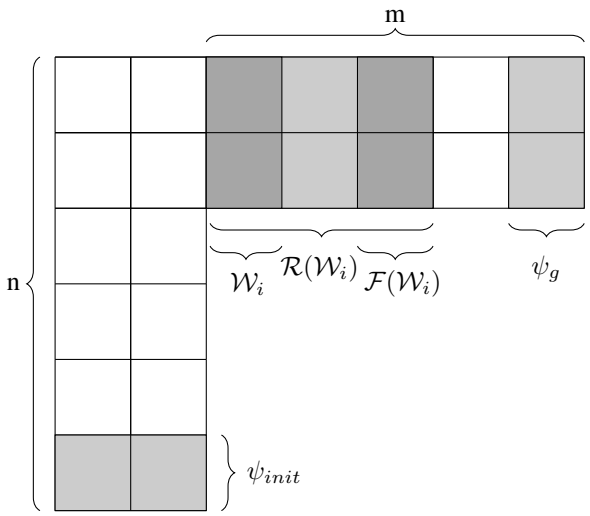

Fig. 1. Example illustrating the road example of size n=6, m=7. ψ_{init} and ψ_g represents the initial condition and goal of the full specification. The figure also illustrates $\mathcal{W}_i, \mathcal{F}(\mathcal{W}_i), \mathcal{R}(\mathcal{W}_i)$ for a specific short horizon problem with horizon 2.

TABLE I

RESULTS FOR THE DOUBLE ROAD WITHOUT RECEDING HORIZON FRAMEWORK

length	states	table size
3	163	<50000
4	698	>50000
5	2827	>400000
6	?	>3200000

B. Regular road, no receding horizon

For comparison, we ran the solver on a similar problem without a turn 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 how many states the respective controllers have and the approximate table sizes needed when extracting them for different road lengths. 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 for synthesizing all controllers, which seems to increase linearly with the length. t_{reduce} is the total time spent on reducing the short horizon problems. $state_{max}$, $state_{med}$ and $state_{tot}$ are the maximum, median and sum of 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

We included a few examples with horizon $h = 2$ too even though $h = 1$ is sufficient. The number of states quickly increases with the horizon as expected. However, the complexity of the reduction still grows much slower than the complexity did without reduction, we are thus able to solve much larger problems. The results are seen in table III.

TABLE II

RESULTS FOR THE DOUBLE ROAD WITH HORIZON 1

n	m	t_{synth}	t_{reduce}	$state_{max}$	$state_{med}$	$state_{tot}$
3	3	11	22	230	45	410
4	4	13	47	237	45	505
5	5	14	84	197	45	552
2	9	15	109	197	45	597
2	10	15	109	197	45	642
10	10	22	568	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	43	89	303	303	1827
2	9	51	112	303	303	2179
10	10	105	571	303	303	4440

E. Problem classification

The time to synthesize controllers can be greatly reduced by using the problem classification proposed above. By using our algorithm we were able to classify most of the short horizon problems as equivalent and can therefore check realizability or extract 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, and two for the partitions in the turn. All the other problems were identified as the same class. An illustration of the classification is shown in Figure 2. The resulting reduced short horizon specification for the majority of the specifications can be described by the following functions

$$\begin{aligned}
\hat{\varphi}_{init} \wedge \hat{\Phi} &= (X_{k,l} \vee X_{k,r}) \wedge \neg C_k \\
\hat{\varphi}_s^e &= \bigwedge_{i \in \{k,k+1\}} (O_{k,l} \leftrightarrow \bigcirc O_{k,l}) \wedge (O_{k,r} \leftrightarrow \bigcirc O_{k,r}) \\
\hat{\varphi}_l^e &= True \\
\hat{\varphi}_s^s \wedge \hat{\Phi} &= \neg C_k \wedge \neg C_{k+1} \\
\hat{\varphi}_l^s &= X_{k+1,l} \vee X_{k+1,r},
\end{aligned}$$

where k is the starting partition and $C_i = (X_{i,l} \wedge O_{i,l}) \vee (X_{i,r} \wedge O_{i,r})$ 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 $O_{i,r}, X_{i,l}, X_{i,r}$.

VII. CONCLUSION

We have developed a method that effectively reduces the complexity of the short horizon problems. This makes the receding horizon framework usable in practice without compromising the correctness of controllers. We have also shown how it is possible to identify and reuse controllers for several short horizon problems.

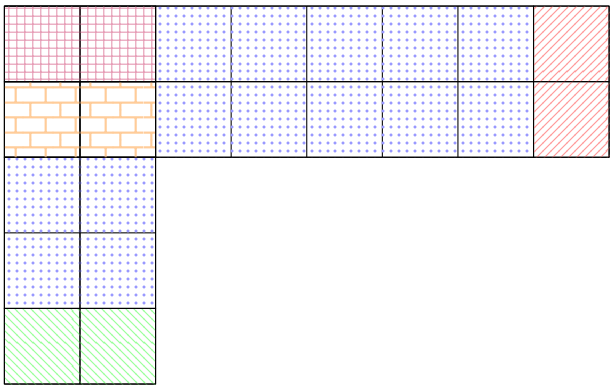

Fig. 2. Illustration of the five different categories of problems identified. Each horizon is identified as equivalent to the others except for the horizons in the start, turn and end.

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] A. Cimatti, M. Roveri, V. Schuppan, and A. Tchaltsev. Diagnostic information for realizability. In *Proceedings of the 9th International Conference on Verification, Model Checking, and Abstract Interpretation*, VMCAI'08, pages 52–67, Berlin, Heidelberg, 2008. Springer-Verlag.

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

[4] 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.

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

[6] 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.

[7] Jun Liu and Necmiye Ozay. Abstraction, discretization, and robustness in temporal logic control of dynamical systems. In *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*, HSCC '14, pages 293–302, New York, NY, USA, 2014. ACM.

[8] P. Nuzzo, H. Xu, N. Ozay, J.B. Finn, A.L. Sangiovanni-Vincentelli, R.M. Murray, A. Donze, and S.A. Seshia. A contract-based methodology for aircraft electric power system design. *Access, IEEE*, PP(99):1–1, 2013.

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

[10] T. Wongpiromsarn, U. Topcu, and R.M. Murray. Receding horizon temporal logic planning. *Automatic Control, IEEE Transactions on*, 57(11):2817–2830, Nov 2012.

[11] T. Wongpiromsarn, Ufuk Topcu, and R.M. Murray. Receding horizon temporal logic planning for dynamical systems. In *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, pages 5997–6004, Dec 2009.

[12] Tichakorn Wongpiromsarn, Ufuk Topcu, Necmiye Ozay, Huan Xu, and Richard M. Murray. Tulip: A software toolbox for receding horizon temporal logic planning. In *Proceedings of the 14th International Conference on Hybrid Systems: Computation and Control*, HSCC '11, pages 313–314, New York, NY, USA, 2011. ACM.