Abstractions and sensor design in partial-information, reactive controller synthesis

Jie Fu, Rayna Dimitrova and Ufuk Topcu

Abstract— **Automated synthesis of reactive control protocols from temporal logic specifications has recently attracted considerable attention in various applications in, for example, robotic motion planning, network management, and hardware design. An implicit and often unrealistic assumption in this past work is the availability of complete and precise sensing information during the execution of the controllers. In this paper, we use an abstraction procedure for systems with partial observation and propose a formalism to investigate effects of limitations in sensing. The abstraction procedure enables the existing synthesis methods with partial observation to be applicable and efficient for systems with infinite (or finite but large number of) states. This formalism enables us to systematically discover sensing modalities necessary in order to render the underlying synthesis problems feasible. We use counterexamples, which witness unrealizability potentially due to the limitations in sensing and the coarseness in the abstract system, and interpolation-based techniques to refine the model and the sensing modalities, i.e., to identify new sensors to be included, in such synthesis problems. We demonstrate the method on examples from robotic motion planning.**

I. Introduction

Automatically synthesizing reactive controllers with proofs of correctness for given temporal logic specifications has emerged as a methodology complementing post-design verification efforts in building assurance in system operation. Its recent applications include autonomous robots [1], [2], hardware design [3], and vehicle management systems [4]. This increasing interest is partly due to both theoretical advances [5], [6] and software toolset developments [7]–[9].

An implicit and often unrealistic assumption in the past work on reactive synthesis is the availability of complete and precise information during the execution of controllers. For example, while navigating through a workspace, a robot rarely (if ever) has global awareness about its surrounding dynamic environment and its sensing of even its own configuration is imprecise. This paper takes an initial step toward explicitly accounting for the effects of such incompleteness and imperfectness in sensing (and other means through which information is revealed to the controller at runtime).

More specifically, we use an abstraction procedure for games with partial observation [10] and propose a formalism to investigate the effects of limitations in sensing. The abstraction reduces the size of the control synthesis problem with sensing limitations by focusing on relevant properties of the control objective and enables automatic synthesis for systems with potentially large state spaces using the solutions for partial-information, turn-based, temporal-logic games [11], [12]. Given unrealizable specifications, where a potential cause for unrealizability is the lack of runtime information, a simple question we investigate is what new sensing modalities and with what precision shall be included in order to render the underlying synthesis problem feasible. We focus on particular safety type temporal logic specifications for which counterexamples witness the unrealizability. Using such counterexamples and interpolation-based techniques [13], the method searches for predicates to be included in the abstraction. We interpret addition of such newly discovered predicates as abstraction refinements as well as adding new sensing modalities or increasing the precision of the existing sensors. Besides the partial-information, turn-based games (see [14], [15] in addition to the earlier references mentioned) the problem we study in this paper has similarities with the partially observable Markov decision processes [16]–[18]. The main deviation in the formalism we employ is the inclusion of a second player which represents a dynamic, possibly adversarial environment, particularly well suited for reactive synthesis in a number of applications, for example, autonomous navigation.

The rest of the paper is organized as follows. We begin with an overview of the setup, problem, and solution approach. In section III, we discuss some preliminaries as they build toward a formal statement of the problem. The solution approach is detailed in the following two sections in which first an abstraction procedure and then refinements in abstractions based on counterexamples are presented. This presentation partly follows the development in [10]. Section VI gives an interpretation of the results in the reconfiguration of sensing modalities and section VII is on a case study. Throughout the paper, we consider motivating and running examples loosely from the context of autonomous robotic motion planning subject to temporal logic specifications.

II. Overview

We begin with a running example and an overview of the problem and our solution approach.

Example 1: Consider a robot in the environment as shown in Fig. 1 with two other dynamic obstacles. The position of this robot is represented by variables x and y in the coordinate system and the initial position is at $x_0 = 4$ and $y_0 = 3$. At each time instance, it can apply the control input u to change its position. The domain of u is $Dom(u) = \Sigma = \{\sigma_1 = (2,0)^T, \sigma_2 = (-2,0)^T, \sigma_3 = (0,1)^T, \sigma_4 = (0,-1)^T\}$. At each time, with input σ_1 (resp. σ_2) the robot

This work was supported in part by the AFOSR (FA9550-12-1-0302) and ONR (N00014-13-1-0778).

The authors are with the University of Delaware (USA), Max Planck Institute for Software Systems (Germany), and the University of Pennsylvania (USA), respectively.

can move in the x-direction precisely with 2 (resp. -2) units, however, in the y-direction there is uncertainty: by σ_3 (resp. σ_4), the robot proceeds some distance ranging from 1 to 1.5 (resp. from -1.5 to -1) unit. There are two uncontrollable moving obstacles, obj1 and obj2, whose behaviors are not known a priori but are known to satisfy certain temporal logic formulas. Suppose as an example design question that the available sensor for y has slow sampling rate, for example, the value of y cannot be observed at every time instance. Can it eventually reach and stay in R_2 while avoiding all the obstacles and not hitting the walls?

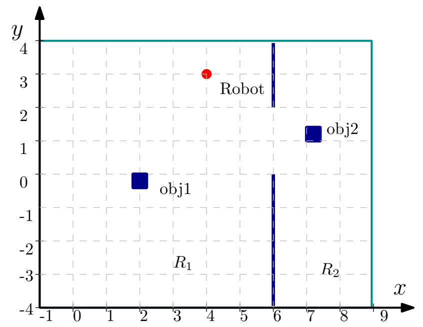


Fig. 1: An environment including a robot (represented by the red dot) and two dynamic obstacles, obj1, obj2. Regions R_1 and R_2 are connected by a door.

A reactive controller senses the environment and decides an action in response based on that sensor reading (or a finite history of sensor readings). For control synthesis in reactive systems with partial observation, two problems are critical. One is a synthesis problem: given the *current* sensor design, is there a controller that realizes the specification? Another is a design problem: given an unrealizable specification, would it be possible to find a controller by introducing new sensing modalities? If so, what are the necessary modalities to add?

To answer these questions, we consider the counterexample guided abstraction refinement procedure for two-player games with partial observation in [19] . First, we formalize the interaction between a system and its environment as a *(concrete) game*. A safety specification determines the winning conditions for both players. Then, an initial set of predicates is selected to construct an *abstract game* with finite state space. The abstraction is *sound* in the sense that if the specification is realizable with the system's partial observation in the abstract game, then it is so in the concrete game. However, if there does not exist such a controller, a *counterexample* that exhibits a violation of the specification can be found. The procedure checks whether this counterexample exists in the concrete game. If it does not, i.e., it is spurious, then the abstract game is refined until a controller is obtained, or a genuine counterexample is found.

In the latter case, the task is not realizable by the system with its current sensor design. Then, we check whether it is realizable under the assumption of complete information, using the same abstraction refinement procedure. If the answer is yes, then the set of predicates obtained in the abstraction refinement indicates the sensing modalities that are sufficient, with respect to the given specification.

III. Problem formulation

In this section we provide necessary background for presenting the results in this paper. For a variable x we denote with $Dom(x)$ its domain. Given a set of variables X, a *state* v is a function $v : X \to \bigcup_{x\in X} Dom(x)$ that maps each variable x to a value in $Dom(x)$. For $Y \subseteq X$, we write $v(Y)$ for the projection of v on Y. Let the set of states over X be V. A *predicate* (atomic formula) p is a statement over a set of variables X. For a given state v, p has a unique value —true (1) or false (0). We write $p(v) = 1$ if p is evaluated to true by the state v. Otherwise, we write $p(v) = 0$. Given a state $v \in V$, we write $v \models \varphi$, if the valuation of φ at v is true. Otherwise, we write $v \not\models \varphi$. Given a formula φ over a set of predicates $\mathcal{P}$, let $\mathsf{Preds}(\varphi) \subseteq \mathcal{P}$ be the set of predicates that occur in φ. A *substitution* of all variables X in φ with the set of new variables X' is denoted $\varphi(X')$.

A. The model

A *(first-order) transition system* symbolically represents an infinite-state transition system [13].

Definition 1: A transition system (TS) $\mathbb{C}$ is a tuple $\langle X, \mathcal{T}, \varphi_{init}\rangle$ with components as follows.

- X is a finite set of variables.
- $\mathcal{T}(X, X')$ is a (quantifier-free) first-order logic formula describing the transition relation. $\mathcal{T}$ relates the variables X which represent the *current* state, with the variables X' which represent the state after this transition.
- φ_{init} is a (quantifier-free) first-order formula over X which denotes the set of initial states of $\mathbb{C}$.

The interaction between system and its environment is captured by a reactive system formalized as a TS.

Example 2: We consider a modified version of Example 1 in which the environment does not contain any obstacle or internal walls. The set of variables is $X = \{x, y, u, t\}$ where t is a Boolean variable. When $t = 0$, the values of variables x, y, u are updated. Formally, the transition relation is

$$\begin{aligned}\mathcal{T} := &\big(t \wedge t' = \neg t \wedge x' = x \wedge y' = y \wedge (\vee_{\sigma_i\in\Sigma} u' = \sigma_i)\big)\\ \vee\ &\big(\neg t \wedge t' = \neg t \wedge \big((u = \sigma_1 \wedge x' = x + 2 \wedge y' = y)\\ &\vee(u = \sigma_2 \wedge x' = x - 2 \wedge y' = y)\vee\\ &(u = \sigma_3 \wedge x' = x \wedge y' \geq y + 1 \wedge y' \leq y + 1.5)\vee\\ &(u = \sigma_4 \wedge x' = x \wedge y' \leq y - 1 \wedge y' \geq y - 1.5)\big)\big).\end{aligned}$$

Initially, $\varphi_{init} := x = 4 \wedge y = 3$ holds.

A TS can be considered in a game formulation in which the system is player 1 and the environment is player 2. For this purpose, the set of variables X is partitioned into $X_I \cup X_O \cup \{t\}$, where X_I is the set of *input variables*, controlled by the environment, and X_O is the set of *output variables*, controlled by the system, and t is a Boolean *turn variable* indicating whose turn it is to make a transition: 1 for the system and 0 for the environment. In Example 2, the set of input variables is $X_I = \{x, y\}$, the set of output variables is $X_O = \{u\}$, and the turn variable is t. We assume the domain of each output variable is finite. Without loss of generality [1],

[1]For a set of output variables, each of which has a finite domain, one can always construct a single new output variable to replace the set, and the domain of this new variable is the Cartesian product of the domains of these output variables.

let X_O be a *singleton* $X_O = \{u\}$ and $Dom(u) = \Sigma$, which is a finite alphabet.

A TS $\mathbb{C}$ defines a game structure. In this paper, we assume that the system and its environment do not perform concurrent actions, and thus the game structure is turn-based.

Definition 2: A *game structure* capturing the interactions of a system (player 1) and its environment (player 2) in a TS $\mathbb{C} = \langle X, \mathcal{T}, \varphi_{init} \rangle$ is a tuple $G = \langle V, T, I \rangle$

- $V = V_1 \cup V_2$ is the set of states over X. $V_1 = \{v \in V \mid v(t) = 1\}$ is the set of states at which player 1 makes a move ($t = 1$). $V_2 = V \setminus V_1$ consists of the states at which player 2 makes a move.
- $T = T_1 \cup T_2$ is the transition relation:
  - $((x_I, x_O, 1), (x'_I, x'_O, 0)) \in T_1$ if and only if $x_I = x'_I$ and $\mathcal{T}((x_I, x_O, 1), (x'_I, x'_O, 0))$ evaluates to true.
  - $((x_I, x_O, 0), (x'_I, x'_O, 1)) \in T_2$ if and only if $x_O = x'_O$ and $\mathcal{T}((x_I, x_O, 0), (x'_I, x'_O, 1))$ evaluates to true.
- $I = \{v \in V \mid v \models \varphi_{init}\}$ is the set of initial states.

A *run* is a finite (or infinite) sequence of states $\rho = v_0 v_1 v_2 \ldots \in V^*$ (or $\rho \in V^\omega$) such that $(v_i, v_{i+1}) \in T$, for each $0 \le i < |\rho|$ where $|\rho|$ is the length of ρ. We assume the game is nonblocking, that is, for all $v \in V$, there exists $v' \in V$ such that $(v, v') \in T$. This can be achieved by including "idle" action in the domain of the output variable.

Definition 3 (Sensor model): Assuming the output variable u and the Boolean variable t are globally observable, the sensor model is given as a set of formulas $\{\mathcal{O}_x \mid x \in X_I\}$, where for each input $x \in X_I$, $\mathcal{O}_x$ is a formula over the set of input variables X_I such that the value of the input variable x is observable at state v if and only if the formula $\mathcal{O}_x$ evaluates to true at the state v.

For a state $v \in V$, the set of *observable variables* at v is $\mathsf{Obs}_X(v) = \{x \in X_I \mid v \models \mathcal{O}_x\} \cup \{t, u\}$. The *observation* of v is $\mathsf{Obs}(v) = v(\mathsf{Obs}_X(v))$, which is the projection of v onto the set of variables observable at v. Two states v, v' are *observation-equivalent*, denoted $v \equiv v'$ if and only if $\mathsf{Obs}(v) = \mathsf{Obs}(v')$. The observation-equivalence can be extended to sequences of states: let $\mathsf{Obs}(\epsilon) = \epsilon$ and $\mathsf{Obs}(v\rho) = \mathsf{Obs}(v)\mathsf{Obs}(\rho)$, for $v \in V$ and $\rho \in V^*$(or V^ω). Two runs $\rho, \rho' \in V^*(V^\omega)$ are observation equivalent, denoted $\rho \equiv \rho'$, if and only if $\mathsf{Obs}(\rho) = \mathsf{Obs}(\rho')$.

This sensor model is able to capture both global and local sensing modalities: if a variable x is globally observable (globally unobservable), $\mathcal{O}_x = \top$ (resp. $\mathcal{O}_x = \bot$). Here $\top$ and $\bot$ are symbols for unconditional true and false, respectively. As an example of a local sensing modality, consider a sensor model in which an obstacle at (px, py) is observable if it is in close proximity of the robot at (x, y), can be described as $\mathcal{O}_{px} = (-2 \le px - x \le 2) \wedge (-2 \le py - y \le 2) \wedge \mathcal{O}_x \wedge \mathcal{O}_y$.

B. Specification language

We use Linear temporal logic (LTL) formulas [20] to specify a set of desired system properties such as safety, liveness, persistence and stability.

In this paper, we consider safety objectives: the given specification is in the form $\Box\neg\varphi_{err}$, where $\Box$ is the LTL operator for "always" and φ_{err} is a formula specifying a set of unsafe states $E = \{v \in V \mid v \models \varphi_{err}\}$. The objective of the system is to always avoid the states in E and the goal of the environment is to drive the game into a state in E.

Let $v_0 \in I$ be the designated initial state of the system. We obtain the game $\mathcal{G}^c = \langle V, v_0, T, E \rangle$, corresponding to the reactive system $\mathbb{C}$ with the initial state v_0. From now on, $\mathcal{G}^c$ and $\mathbb{C}$ are referred to as the *concrete* game and *concrete* reactive system, respectively. The state set V is the set of *concrete* states. A run $\rho \in V^\omega$ is winning for player 1 if it does not contain any state in the set of unsafe states E.

A *strategy* for player i is a function $f_i : V^* V_i \to V_j$ which maps a finite run ρ into a state $f_i(\rho) \in V_j$, to be reached, such that $(v, v') \in T$, where v is the last state in ρ, $v' = f_i(\rho)$ and $(i, j) \in \{(1, 2), (2, 1)\}$. The set of runs in $\mathcal{G}$ with the initial state $v_0 \in I$ induced by a pair of strategies (f_1, f_2) is denoted by $Out_{v_0}(f_1, f_2)$. Given the initial state v_o, a strategy f_1 is winning for player 1, if and only if for any strategy f_2 of player 2, any run in $Out_{v_0}(f_1, f_2)$ is winning for player 1. A winning strategy for player 2 is defined dually.

Since the system (player 1) has partial observability, the strategies it can use are limited to the following class.

Definition 4: An *observation-based* strategy for player 1 is a function $f_1 : V^* V_1 \to V_2$ that satisfies: (1) f_1 is a strategy of player 1; and (2) for all ρ_1, ρ_2, if $\rho_1 \equiv \rho_2$, then given $v = f_1(\rho_1), v' = f_1(\rho_2)$, it holds that for the output variable u, $v(u) = v'(u)$, and $v(t) = v'(t)$.

For a game with partial observation, one can use knowledge-based subset construction to obtain a game with complete observation. The winning strategy for player 1 in the latter is an observation-based winning strategy for player 1 in the former. The reader is referred to [21] for the solution of games with partial observation.

C. Problem statement

We now formally state the problem investigated in this paper.

Problem 1: Given a transition system $\mathbb{C}$ with the initial state $v_0 \in I$, with a sensor model $\{\mathcal{O}_x \mid x \in X_I\}$ and a safety specification $\Box\neg\varphi_{err}$, determine whether there exists an observation-based strategy (i.e. controller) f_1 such that for any strategy of the environment f_2 and for any $\rho \in Out_{v_0}(f_1, f_2)$, $\rho \models \Box\neg\varphi_{err}$. If no such controller exists, then determine a new sensor model for which one can find such a controller, if there exists one.

IV. Predicate abstraction

Since the game $\mathcal{G}^c$ may have a large number of states, the synthesis methods for finite-state games cannot be directly applied or are not efficient. To remedy this problem, we apply an abstraction procedure which combines predicate abstraction and knowledge-based subset construction and yields an *abstract finite-state* game with *complete* information $\mathcal{G}^a$ from the (symbolically represented) concrete game $\mathcal{G}^c$.

A. An abstract game

Given a *finite* set of predicates, the abstraction procedure constructs a finite-state reactive system (game structure). Let

$\mathcal{P} = \{p_1, p_2, \dots, p_N\}$ be an indexed set of predicates over variables X. The *abstraction function* $\alpha_{\mathcal{P}} : V \to \{0,1\}^{|\mathcal{P}|}$ maps a concrete state into a binary vector as follows.

$$\alpha_{\mathcal{P}}(v) = s \in \{0,1\}^{|\mathcal{P}|} \text{ iff } \ s(i) = p_i(v), \text{for all } p_i \in \mathcal{P},$$

where $s(i)$ is the ith entry of binary vector s. The *concretization function* $\gamma_{\mathcal{P}} : \{0,1\}^{|\mathcal{P}|} \to 2^V$ does the reverse:

$$\gamma_{\mathcal{P}}(s) = \{v \mid \forall\ p_i \in \mathcal{P}.\ p_i(v) = s(i)\}.$$

In the following, we omit the subscript $\mathcal{P}$ in the notation for the abstraction and concretization functions wherever they are clear from the context. The following lemma shows that with a proper choice of predicates, we can ensure that a set of concrete states grouped by the abstraction function shares the same set of observable and unobservable variables.

Lemma 1: Let $\bigcup_{x \in X_I} \mathsf{Preds}(\mathcal{O}_x) \subseteq \mathcal{P}$. Then for any binary vector $s \in \{0,1\}^{|\mathcal{P}|}$ and any two states $v, v' \in \gamma(s) \neq \emptyset$, it holds that $\mathsf{Obs}_X(v) = \mathsf{Obs}_X(v')$.

Proof: Since for any $v, v' \in \gamma(s)$, $\alpha(v) = \alpha(v') = s$, for any $p \in \mathcal{P}$, p has the same truth value at states v and v'. Thus, for any $x \in X_I$, the formula $\mathcal{O}_x$, for which $\mathsf{Preds}(\mathcal{O}_x) \subseteq \mathcal{P}$, has the same value at v and v'. Hence, if x is observable (or unobservable) at v, then it must be observable (or unobservable) at v' and vice versa. ■

Intuitively, by including the predicates in the formulas defining the sensor model, for each $s \in \{0,1\}^{|\mathcal{P}|}$, the set of concrete states $\gamma(s)$ share the same sets of observable and unobservable variables. Hence, we use $X_v(s)$ to denote set of observable/visible input variables in s and $X_h(s) = X_I \backslash X_v(s)$ for the set of unobservable/hidden input variables.

A predicate p is observable at a state v if and only if the variables in p are observable at v. According to Lemma 1, if there exists $v \in \gamma(s)$ such that p is observable at v, then p is observable for all $v \in \gamma(s)$ and we say that p is observable at s. Slightly abusing the notation $\mathsf{Obs}(\cdot)$, the *observation of a binary vector* s is $\mathsf{Obs}(s) = \{(p_i, s(i)) \mid p_i \text{ is observable at } s\}$, which is a set of assignments for observable predicates. Two binary vectors s, s' are observation-equivalent, denoted $s \equiv s'$, if and only if $\mathsf{Obs}(s) = \mathsf{Obs}(s')$.

The abstraction of the concrete game $\mathcal{G}^c = \langle V, v_0, T, E\rangle$ with respect to a finite set of predicates $\mathcal{P}$ is a game with *complete information* $\alpha(\mathcal{G}^c, \mathcal{P}) = \mathcal{G}^a = \langle S^a, s_0^a, T^a, E^a\rangle$:

- $S^a = S_1^a \cup S_2^a$ is the set of abstract states with sets of player 1's and player 2's abstract states respectively, $S_1^a = \{s^a \mid \exists v \in V_1.\ s^a \subseteq \{s \mid s \equiv \alpha(v)\}, s^a \neq \emptyset\}$ and $S_2^a = \{s^a \mid \exists v \in V_2.\ s^a \subseteq \{s \mid s \equiv \alpha(v)\}, s^a \neq \emptyset\}$.
- $s_0^a = \{s \in \{0,1\}^{|\mathcal{P}|} \mid s \equiv \alpha(v_0)\}$ is the initial state.
- $T^a = T_1^a \cup T_2^a$ where
  - $(s_1^a, s_2^a) \in T_1^a$ if and only if the following conditions (1), (3) and (4) are satisfied.
  - $(s_1^a, s_2^a) \in T_2^a$ if and only if the following conditions (2), (3) and (4) are satisfied.

  (1) for every $s \in s_1^a$ and every $v \in \gamma(s)$, there exist $s' \in s_2^a$ and $v' \in \gamma(s')$ such that $(v, v') \in T_1$;
  (2) there exists $s \in s_1^a$, $v \in \gamma(s)$, $s' \in s_2^a$ and $v' \in \gamma(s')$ such that $(v, v') \in T_2$;
  (3) for every $s' \in s_2^a$, there exist $s \in s_1^a$, $v \in \gamma(s)$ and $v' \in \gamma(s')$ such that $(v, v') \in T$;
  (4) for every $s_1', s_2' \in \alpha(V)$, if $s_1' \in s_2^a$, $s_1' \equiv s_2'$ and there exist $s \in s_1^a$, $v \in \gamma(s)$ and $v' \in \gamma(s_2')$ with $(v, v') \in T$, then also $s_2' \in s_2^a$.
- $E^a = \{s^a \mid \exists s \in s^a.\ \exists v \in \gamma(s).\ v \in E\}$ is the set of unsafe states.

In what follows, we refer to a state $s^a \in S^a$ as an *abstract state*. By definition, each s^a in $\mathcal{G}^a$ is a set of observation-equivalent binary vectors in $\alpha(V)$.

We relate a binary vector $s \in \{0,1\}^{|\mathcal{P}|}$ with a formula $[s]$ that is a conjunction such that $[s] = \wedge_{0 \leq i \leq |\mathcal{P}|} h_i$ where if $s(i) = 1$, then $h_i = p_i$, otherwise $h_i = \neg p_i$. Further, for any $s^a \in S^a$, we define the following formula in disjunctive normal form $[s^a] = \vee_{s \in s^a}[s]$.

Example 2 (cont.): We assume x is globally observable and y is globally unobservable and require that the robot shall never hit the boundary, that is, $\Box\neg\varphi_{err}$ where $\varphi_{err} = \big(t = 0 \wedge (x \geq 9 \vee y \geq 4 \vee x \leq -1 \vee y \leq -4)\big)$. Let $\varphi_{init} := (x = 4 \wedge y = 3 \wedge u = \sigma_1 \wedge t = 1)$. Let $\mathcal{P} = \{x \geq 9, y \geq 4, x \leq -1, y \leq -4, u = \sigma_1, u = \sigma_2, u = \sigma_3, u = \sigma_4, t = 1\}$. The initial state of $\mathbb{C}$ is $v_0 = (4, 3, \sigma_1, 1)$, and the corresponding initial state in $\mathcal{G}^a$ is $s_0^a = \{(000010001)\}$ where the values for the predicates in s_0^a are given in the same order in which they are listed in $\mathcal{P}$. Given $v' = (4, 3, \sigma_2, 0)$, since $(v_0, v') \in T$, we determine $(s_0^a, s_1^a) \in T^a$ where $s_1^a = \{(000001000)\}$ indicating $u = \sigma_2$ and $t = 0$.

We show that by a choice of predicates, it is ensured that for any $s^a \in S^a$, all concrete states in the set $\{v \mid \exists s \in s^a.\ v \in \gamma(s)\}$ share the same observable and unobservable variables.

Lemma 2: If $\bigcup_{x \in X_I} \mathsf{Preds}(\mathcal{O}_x) \subseteq \mathcal{P}$, then for any $s^a \in S^a$ and $v, v' \in \{v \mid \exists s \in s^a.\ v \in \gamma(s)\}$, it holds that $\mathsf{Obs}_X(v) = \mathsf{Obs}_X(v')$.

Proof: By Lemma 1, since for any $s \in \{0,1\}^{|\mathcal{P}|}$, for any $v, v' \in \gamma(s) \neq \emptyset$, $\mathsf{Obs}_X(v) = \mathsf{Obs}_X(v')$, then it suffices to prove that for any $s, s' \in s^a$, $X_v(s) = X_v(s')$ and $X_h(s) = X_h(s')$. By definition, $s \equiv s'$ implies that the set of observable (unobservable) predicates is the same in both s and s'. Thus, the set of observable (unobservable) variables that determines the observability of predicates has to be the same in both s and s'. That is, $X_v(s) = X_v(s')$ and $X_h(s) = X_h(s')$. ■

Let $X_v(s^a)$ (resp. $X_h(s^a)$) be the observable (resp. unobservable) input variables in the abstract state s^a. That is, $X_i(s^a) = X_i(s)$ for any $s \in s^a$, for $i \in \{v, h\}$.

B. Concretization of strategies

In the abstract game $\mathcal{G}^a$, there exists a winning strategy for one of the players. We show that a winning strategy for the system in $\mathcal{G}^a$ can be concretized into a set of observation-based winning strategies for the system in $\mathcal{G}^c$.

For $(i, j) \in \{(1,2), (2,1)\}$, the *concretization of a strategy* $f_i : (S^a)^* S_i^a \to S_j^a$ in $\mathcal{G}^a$ is a *set* of strategies in $\mathcal{G}^c$, denoted $\gamma(f_i)$ and can be obtained as follows. Consider $\rho^c \in V^*$, $\rho \in S^*$, $\rho^a \in (S^a)^*$ in the following, where

$$\begin{aligned}\rho^c &= v_0\ v_1\ v_2\ \ldots\ v_n\ ,\\ \rho &= s_0\ s_1\ s_2\ \ldots\ s_n\ ,\\ \rho^a &= s_0^a\ s_1^a\ s_2^a\ \ldots\ s_n^a\ .\end{aligned}$$

and $v_i \in \gamma(s_i)$, $s_i \in s_i^a$ for each $i : 0 \leq i \leq n$. Given $f_i(\rho^a) = s_{n+1}^a$, the output $f_i^c(\rho^c) = v_{n+1}$ such that there exist $s \in s_{n+1}^a$ and $v_{n+1} \in \gamma(s)$ such that $(v_n, v_{n+1}) \in T$. In other words, v_{n+1} is a concrete state reachable from the current state v_n and can be abstracted into a binary vector s in the abstract state s_{n+1}^a. Intuitively, given the run ρ^c, one can find a run in the abstract system ρ^a, and uses the output of f_i on ρ^a to generate an abstract state. Then f_i^c picks a reachable concrete state, which can also be abstracted into a binary vector contained this abstract state. A strategy f is *concretizable* if $\gamma(f) \neq \emptyset$. Otherwise it is *spurious*.

Theorem 1: The concretization $\gamma(f_1)$ of a player 1's winning strategy $f_1 : (S^a)^* S_1^a \to S_2^a$ in $\mathcal{G}^a$ is a non-empty set that consists of observation-based winning strategies for player 1 in the concrete game $\mathcal{G}^c$.

Proof: Follows from the proof in [10]. ■

In case there is no winning strategy for player 1 in $\mathcal{G}^a$, the synthesis algorithm gives us a winning strategy for player 2 in $\mathcal{G}^a$, which we refer to as *counterexample*. Then we need to check if it is spurious, as explained in the next section.

V. Abstraction refinement

We consider an initial set of predicates $\mathcal{P}$ which consists of the predicates occurring in φ_{err}, the predicates describing the output u of the system, and those occurring in the sensor model. With this initial choice of predicates, if player 1 wins the game $\mathcal{G}^a = \alpha(\mathcal{G}^c, \mathcal{P})$, then the abstraction does not need to be further refined, according to Theorem 1, the winning strategy of player 1 is concretizable in the concrete game. However, if player 2 wins, there exists a deterministic winning strategy $f_2 : (S^a)^* S_2^a \to S_1^a$ in the game $\mathcal{G}^a$. The next step is to check if f_2 is spurious. If it is, then the abstract model is too coarse and needs to be further refined.

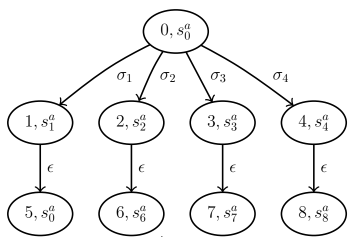
A. Constructing abstract counterexample tree

We construct a formula from the strategy tree generated from this counterexample that characterizes the concretizability of this counterexample in the concrete system $\mathbb{C}$, and then we construct a formula from the tree. If the formula is satisfiable, then the counterexample is genuine.

Given the initial state s_0^a, the abstract counterexample tree (ACT) for f_2 is $\mathbb{T}(f_2, s_0^a) = (\mathcal{N}, \mathcal{E})$ where $\mathcal{N}$ are nodes and $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$ are edges. Each node n in $\mathcal{N}$ is labeled by a state $s^a \in S^a$ and we denote the labeling $n : s^a$. A node $n : s^a$ belongs to player i if $s^a \in S_i^a$, for $i = 1, 2$.

In the case of a safety specification, $\mathbb{T}(f_2, s_0^a)$ is a finite tree in which the following conditions hold. 1) The root 0 is labeled by s_0^a, that is, $0 : s_0^a$. 2) If $n : s^a$ is a player 1's node and n is not a leaf, then for each t^a such that $(s^a, t^a) \in T^a$, add a new child m of n and label m with t^a. Let $n \xrightarrow{\sigma} m$ for which $[t^a] \implies u = \sigma$. 3) If $n : s^a$ is a player 2's node and n is not a leaf, then add one child m of n, labeled with $t^a = f_2(\rho)$, where ρ is the sequence of nodes' labels (states) on the path from the root to the node n. Let $n \xrightarrow{\epsilon} m$ where ϵ is the empty string. 4) For a node $n : s^a$, n is a leaf if either $s^a \in E^a$ or there is no outgoing transition from s^a. 5) Each node has at most one parent.

We illustrate the ACT construction on the small example. Fig. 2 shows a fragment of ACT for Example 2. First we define the root 0, labeled with the abstract state s_0^a. At s_0^a, player 1 can select any output in Σ. Therefore, the children of 0 are $1, 2, 3, 4$, one for each input in Σ. For instance, the output σ_2 labels the edge from 0 to 2 and we have $2 : s_2^a$. The only child of 2 is 6, labeled with $s_6^a = \{(001001001)\}$. Clearly, the actual value of x after executing σ_2 is 2. Yet the reached state s_6^a in which the predicate $x \leq -1$ is true is because there exists some $x \in (-1, 1]$ at state s_2^a, and will make $x \leq -1$ satisfied after action σ_2. This is caused by the coarseness of the abstraction. If player 1 takes action σ_3, then it will have no information about the value of the predicate $(y \geq 4)$, as this predicate is not observable. In Fig. 2, each state $s_i^a, 0 \leq i \leq 7$ is related with a formula $[s_i^a]$ (shown below the figure).



$[s_0^a]$	$u = \sigma_1 \wedge t = 1 \wedge \phi_0$	$[s_1^a]$	$u = \sigma_1 \wedge t = 0 \wedge \phi_0$
$[s_2^a]$	$u = \sigma_2 \wedge t = 0 \wedge \phi_0$	$[s_3^a]$	$u = \sigma_3 \wedge t = 0 \wedge \phi_0$
$[s_4^a]$	$u = \sigma_4 \wedge t = 0 \wedge \phi_0$	$[s_6^a]$	$u = \sigma_1 \wedge t = 1 \wedge \phi_6$
$[s_7^a]$	$u = \sigma_2 \wedge t = 1 \wedge \phi_7$	$[s_8^a]$	$u = \sigma_4 \wedge t = 1 \wedge \phi_8$

Fig. 2: A fragment of ACT for Example 2. Note that nodes 5 and 0 are labeled with the same state s_0^a. In the formulas $[s_i^a]$,

$$\begin{aligned}\phi_0 &= \neg(x \geq 9) \wedge \neg(x \leq -1) \wedge \neg(y \geq 4) \wedge \neg(y \leq -4),\\ \phi_6 &= \neg(x \geq 9) \wedge (x \leq -1) \wedge \neg(y \geq 4) \wedge \neg(y \leq -4),\\ \phi_7 &= \neg(x \geq 9) \wedge \neg(x \leq -1) \wedge \neg(y \leq -4),\\ \phi_8 &= \neg(x \geq 9) \wedge \neg(x \leq -1) \wedge \neg(y \geq 4).\end{aligned}$$

For a node $n \in \mathcal{N}$, $C(n)$ is the set of children of n and $\mathsf{Paths}(n) \subseteq \mathcal{N}^*$ is the set of paths from the node n to a leaf. For a path $\rho \in \mathcal{N}^*$, the *trace* of ρ, denoted $\mathsf{Trace}(\rho) \in \Sigma^*$, is the sequence of labels on the edges in the path. A node $n \in \mathcal{N}$ is related with a set of traces $\mathsf{Traces}(n) = \{\mathsf{Trace}(\rho) \mid \rho \in \mathsf{Paths}(n)\}$. For a leaf, $\mathsf{Traces}(n) = \{\epsilon\}$ by default. For example, $\mathsf{Trace}(0 \xrightarrow{\sigma_1} 1 \xrightarrow{\epsilon} 5) = \sigma_1\epsilon = \sigma_1$.

Note that in the tree structure defined here, for each node $n \in \mathcal{N}$, there exists exactly one path from the root to n, and hence there is one trace $w \in \Sigma^*$ that labels that path.

We annotate each node $n : s^a$ with a set of variables $\mathcal{X}^n$ as $n : s^a : \mathcal{X}^n$ where $\mathcal{X}^n = \{X^{n,w} \mid w \in \mathsf{Traces}(n)\}$ and $X^{n,w} = (X_v^n, X_h^{n,w}, u^n, t^n)$ where $X_v^n \cup \{t^n, u^n\}$ are observable variables in s^a and $X_h^{n,w}$ are hidden variables in s^a when the trace from n is w. For example, we annotate 0 with $\mathcal{X}^0 = \{X^{0,w} = (x^0, y^{0,w}, u^0, t^0) \mid w \in \mathsf{Traces}(0) = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}\}$ as y is not observable. With this annotation, the unobservable variables X_h at node n can be assigned with different values for different traces from n. It corresponds to the fact that the concrete states, grouped into an abstract state, share the same values for observable variables but may have different values for unobservable ones.

In what follows, we relate a trace with a *trace formula*. By checking the satisfiability of a *tree formula*, built from trace formulas with a Satisfiability Modulo Theories (SMT) solver, we can determine whether the counterexample is spurious.

B. Analyzing the counterexample

Given a trace $w \in \mathsf{Traces}(n)$, the trace formula $F(n,w)$ is constructed recursively as follows.

- If $n : s^a : \mathcal{X}^n$ is a leaf, then $\mathcal{X}^n = \{X^{n,\epsilon}\}$ is a singleton. Let $F(n,w) = [s^a](X^{n,\epsilon})$, which is satisfiable if there exists a concrete state v for $X^{n,\epsilon}$ such that $[s^a](v) = \mathsf{true}$.
- If $n : s^a : \mathcal{X}^n$ is a player 1's node and not a leaf, then for each $w = \sigma w' \in \mathsf{Traces}(n)$, for each child $m : t^a : \mathcal{X}^m$ such that $n \xrightarrow{\sigma} m$, let
$$F(n,m,w) = F(m,w') \wedge [s^a](X^{n,w}) \wedge [t^a](X^{m,w'}) \wedge u^m = \sigma \wedge \mathcal{T}(X^{n,w}, X^{m,w'})$$
where $F(m,w')$ is false if $w' \notin \mathsf{Traces}(m)$. Then let $F(n,w) = \vee_{m \in C(n), u^m=\sigma} F(n,m,w)$. Intuitively, $F(n,m,w)$ can be satisfied if there exist a state v for $X^{n,w}$ and v' for $X^{m,w'}$ such that $[s^a]$ and $[t^a]$ evaluate to true at v and v', respectively; action σ enables the transition from v to v'; and $F(m,w')$ is satisfied. The disjunction is needed because for a node n, there can be more than one σ-successors.
- If $n : s^a : \mathcal{X}^n$ is a player 2's node and not a leaf, there exists exactly one child of n, say, $m : t^a : \mathcal{X}^m$, then for each $w \in \mathsf{Traces}(n)$, let $F(n,w) = F(m,w) \wedge [s^a](X^{n,w}) \wedge [t^a](X^{m,w}) \wedge \mathcal{T}(X^{n,w}, X^{m,w})$.

The *tree formula* is
$$F(0) = \wedge_{w \in \mathsf{Traces}(0)} (F(0,w) \wedge \varphi_{init}(X^{0,w})).$$

Theorem 2: Let f_2 be a winning strategy for the environment in the game $\mathcal{G}^a$, the strategy f_2 is genuine, i.e., $\gamma(f_2) \neq \emptyset$, if and only if the tree formula $F(0)$ is satisfiable.

Proof: The reader is referred to [19]. ■

Example 2 (cont.): Consider, for instance, the trace $\sigma_1 w' \in \mathsf{Traces}(0)$ corresponds to a labeled path $0 \xrightarrow{\sigma_1} 1 \xrightarrow{\epsilon} 5$ and $w' \in \mathsf{Traces}(5)$. Since $0 \xrightarrow{\sigma_1} 1$, we have $F(0,\sigma_1 w') = F(1,w') \wedge [s_0^a](X^{0,\sigma_1 w'}) \wedge [s_1^a](X^{1,w'}) \wedge u^1 = \sigma_1 \wedge \mathcal{T}(X^{0,\sigma_1 w'}, X^{1,w'})$ where $X^{1,w'} = (x^1, y^{1,w'}, u^1, t^1)$. Then given $1 \xrightarrow{\epsilon} 5$, $F(1,w') = F(5,w') \wedge [s_1^a](X^{1,w'}) \wedge [s_5^a](X^{5,w'}) \wedge \mathcal{T}(X^{1,w'}, X^{5,w'})$, where $X^{5,w'} = (x^5, y^{5,w'}, u^5, t^5)$. In above equations, for instance $[s_0^a](X^{0,\sigma w'}) = \neg(x^0 \geq 9 \vee x^0 \leq -1 \vee y^{0,\sigma w'} \leq -4 \vee y^{0,\sigma w'} \geq 4) \wedge u^0 = \sigma_1 \wedge t^0 = 1$.

C. Refining the abstract transition relations

Given a node n and a trace $w \in \mathsf{Traces}(n)$, if $F(n,w)$ is unsatisfiable, then the occurrence of the spurious counterexample is due to the approximation made in abstracting the transition relation. To rule out this counterexample, we need to refine the abstract transition relation. For this purpose, we define a *node formula* $\tilde{F}(n,w)$ as described below.

First, we define the *pre-condition of a formula*: for a formula φ and $\sigma \in \Sigma$, the pre-condition of φ with respect to σ, $\textsc{Pre}_1(\sigma,\varphi)$ is a formula such that $v \models \textsc{Pre}_1(\sigma,\varphi)$ if and only if there exists $v' \in V$ such that $v' \models \varphi$, $v'(u) = \sigma$ and $(v,v') \in T_1$. Intuitively, at any state v that satisfies this formula $\textsc{Pre}_1(\sigma,\varphi)$, the system, after initiating the output σ, can reach a state v' at which φ is satisfied. Let $\textsc{Pre}_1(\varphi) = \vee_{\sigma \in \Sigma} \textsc{Pre}_1(\sigma,\varphi)$. Correspondingly, $\textsc{Pre}_2(\varphi)$ is a formula such that $v \models \textsc{Pre}_2(\varphi)$ if and only if there exists $v' \in V$, $v' \models \varphi$ and $(v,v') \in T_2$.

Now, we define the *node formula* $\tilde{F}(n,w)$ as follows.

- If $n : s^a$ is a leaf node, then $w = \epsilon$ and $\tilde{F}(n,\epsilon) = \vee_{s \in s^a, [s] \implies \varphi_{err}} [s]$.
- If $n : s^a$ belongs to player 1 and is not a leaf, and $w = \sigma w'$, then
$$\tilde{F}(n,w) = [s^a] \wedge \textsc{Pre}_1(\sigma, \vee_{\ell \in C(n), u^\ell = \sigma} \tilde{F}(\ell, w')),$$
where $\tilde{F}(\ell,w')$ is false if $w' \notin \mathsf{Traces}(\ell)$. Here, the set $\{\ell \in C(n) \mid u^\ell = \sigma\}$ is a set of σ-successors of n.
- If $n : s^a$ belongs to player 2's and is not a leaf, then
$$\tilde{F}(n,w) = [s^a] \wedge \textsc{Pre}_2\left(\tilde{F}(m,w)\right)$$
where $m \in C(n)$ is the unique child of node n.

We augment the current set $\mathcal{P}$ with all predicates that occur in the formula $\tilde{F}(n,w)$, i.e., $\mathcal{P}' := \mathcal{P} \cup \mathsf{Preds}(\tilde{F}(n,w))$. For each node n and each $w \in \mathsf{Traces}(n)$ such that $F(n,w)$ is unsatisfiable, the procedure generates a set of predicates $\mathsf{Preds}(\tilde{F}(n,w))$, which are then combined with the current predicate set to generate a new abstract game. We repeat this procedure iteratively until a set of predicates is found such that for any n and any $w \in \mathsf{Traces}(n)$, $F(n,w)$ is satisfiable.

D. Refining the abstract observation equivalence

If each trace formula for the considered counterexample tree is satisfiable, but the tree formula is not, then we need to check whether the existence of a counterexample is because of the coarseness in the abstraction observation-equivalence.

We are in the case when for all $w \in \mathsf{Traces}(0)$, $F(0,w) \wedge \varphi_{init}(X^{0,w})$ is satisfiable. Let $\Phi = \{F(0,w) \wedge \varphi_{init}(X^{0,w}) \mid w \in \mathsf{Traces}(0)\}$. Since $F(0) = \wedge_{\phi \in \Phi} \phi$ is unsatisfiable, there exists a subset Ψ of Φ such that $\psi = \wedge_{\phi \in \Psi} \phi$ is satisfiable and a formula $\varphi \in \Phi \setminus \Psi$ such that $\varphi \wedge \psi$ is unsatisfiable. Let the sets of free variables in ψ and φ be Y and Z respectively. Since only observable variables are shared between different traces, $Y \cap Z$ only consists of *observable* variables.

A *Craig interpolant* [22] for the pair $(\psi(Y), \varphi(Z))$ is a formula $\theta(Y \cap Z)$ such that 1) $\psi(Y)$ implies $\theta(Y \cap Z)$, 2) $\varphi(Z) \wedge \theta(Y \cap Z)$ is unsatisfiable. To illustrate, consider the following example. Let $\varphi_1 = (y^5 = y^0 + 1) \wedge (y^5 \geq 4)$ and $\varphi_2 = (y^0 \leq 1)$. Clearly, $\varphi_1 \wedge \varphi_2 \equiv \perp$ because y^0 in φ_1 needs to satisfy $y^0 \geq 3$. Then the formula $\theta = y^0 \geq 3$ is an interpolant for the pair of formulas $(\varphi_1(y^0,y^5), \varphi_2(y^0))$. For a number of logical theories commonly used in verification, including linear real arithmetic, Craig interpolants can be automatically computed [23].

After computing the interpolant θ for (ψ,φ), we update the set of predicates to be $\mathcal{P}' := \mathcal{P} \cup \mathsf{Preds}(\theta)$. In the end, Algorithm 1 describes the refinement procedure.

Algorithm 1: `AbstractAndRefine`

```
Input: the concrete game $\mathcal{G}^c = \langle V, v_0, T, E\rangle$ and the
       sensor model $\{\mathcal{O}_x \mid x \in X_I\}$.
Output: a tuple including the abstract game, the set of
        predicates, the winner, the strategy of the
        winner $(\mathcal{G}^a, \mathcal{P}, winner, f)$.
begin
    $\mathcal{P}$ := $\mathsf{Preds}(\varphi_{err}) \cup (\cup_{x\in X_I}(\mathsf{Preds}(\mathcal{O}_x))$
           $\cup(\cup_{\sigma\in\Sigma}\{u = \sigma\})$;
    $\mathcal{G}^a$ :=Abstract ($\mathcal{G}^c, \mathcal{P}$);
    $(winner, f)$ := Solve ($\mathcal{G}^a$) ;
    while $winner = player\ 2$ do
        $\mathbb{T}(f, s^a_0)$ = ConstructACT ($f, s^a_0$) ;
        if $F(0)$ is satisfiable then
            return ($player\ 2, f$)
                /* Unsatisfiable */
        else
            if $\exists w \in \mathsf{Traces}(0)$, $F(0, w)$ is unsatisfiable
            then
                $\mathcal{R}$ := TraceRefinement ($F(0, w)$);
            else
                $\mathcal{R}$ := TreeRefinement ($F(0)$);
        $\mathcal{P} := \mathcal{P} \cup \mathcal{R}$;
        $\mathcal{G}^a$ := Abstract ($\mathcal{G}^c, \mathcal{P}$);
        $(winner, f)$ := Solve ($\mathcal{G}^a$);
    return $(\mathcal{G}^a, \mathcal{P}, winner, f)$.
```

VI. Sensor reconfiguration

Suppose the task specification is unrealizable given the current sensor model. Then, a prelude to refining the sensor is identifying whether the source of unrealizability is limited sensing. To this end, we first check whether it is realizable under the assumption that the system has perfect observation over its environment. For this purpose, we run the procedure `AbstractAndRefine` with the concrete game $\mathcal{G}^c$ and a sensor model defined as $\{\mathcal{O}_x = \top \mid x \in X_I\}$, which means all the input variables are globally observable. If player 1 wins the abstract game, then we can conclude that the task is not realizable because of the limited sensing capability.

The procedure `SensorReconfigure`, shown as Algorithm 2, computes a set of predicates that we need to observe in order to satisfy a given specification. The algorithm takes the concrete system, its current sensor model and an unrealizable specification as input. Then by making all variables observable, we use the procedure `AbstractAndRefine` to determine if the task is realizable given complete observation. If `AbstractAndRefine` terminates with a positive answer, then, the set of predicates obtained by the refinement suffices for realizing the specification. Further, the predicates involving unobservable variables indicate the set of new sensing modalities to be added, and provide the requirements on the sensors' precision and accuracy for both observable and unobservable variables.

Algorithm 2: `SensorReconfigure`

```
Input: the concrete game $\mathcal{G}^c = \langle V, v_0, T, E\rangle$, and the
       sensor model $\{\mathcal{O}_x \mid x \in X_I\}$. There is no
       realizable winning strategy for the system.
Output: a tuple of the winner, its strategy and the set
        of predicates $(winner, f, \mathcal{P})$.
begin
    $\widetilde{\mathcal{O}} = \{\mathcal{O}_x = \top \mid x \in X_I\}$;
    $(\mathcal{G}^a, \mathcal{P}, winner, f)$ :=
        AbstractAndRefine ($\mathcal{G}^c, \widetilde{\mathcal{O}}$) ;
    if $winner = player\ 2$ then
        return $(winner, f, \mathcal{P})$; /* Unsatisfiable
            even with full observation. */
    else
        return $(winner, f, \mathcal{P})$;
```

VII. Case study

We demonstrate the method by revisiting Example 1. Assuming the dynamics of obstacle obj1 with position (x_p, y_p) is given in form of logical formula $\varphi_p := ((x \leq 6 \wedge x'_p \geq 7) \vee (x > 6 \wedge x'_p \geq 6)) \wedge \neg\varphi_{hit}$ where φ_{hit} is a formula that is satisfied when the obstacle hits the wall or the robot. For obstacle obj2 (x_o, y_o), we have $\varphi_o := ((x \leq 6 \wedge x'_o < 4) \vee (x > 6 \wedge x'_o \leq 7)) \wedge \neg\varphi_{hit}$. Here we have a liveness condition which specifies that the robot has to visit and then stay within region R_2. To enforce such constraint, we introduce a Boolean variable err and set $err = 1$ if $x \geq 6 \wedge x' < 6$, which means if the robot in R_2 returns to R_1, an error occurs and the system reaches an unsafe state.

Case 1: Due to the limited sampling rate in the sensor for variable y, the system receives the exact value of y intermittently (every other step). In this case, we introduce a predicate p_s such that if $p_s = 1$ then the exact value of y is observed, otherwise there is no data sampled. The transition relation $\mathcal{T}(X, X')$ is modified to capture this type of partial observation. For example, given $u = \sigma_3$, the transition is $t \wedge (u = \sigma_3) \wedge y' \geq y + 1 \wedge y' \leq y + 1.5 \wedge x' = x \wedge \varphi_p \wedge ((\neg p_s \wedge y'_s = y + 1 \wedge y'_n = y + 1.5) \vee (p_s \wedge y'_s = y' \wedge y'_n = y')) \wedge p'_s = \neg p_s \wedge t' = \neg t$ where y_s and y_n are auxiliary variables used by the robot to keep track of the upper and lower bounds, respectively, for the value of y. Intuitively, when there is no data received, the robot makes a move such that for every y within the upper and lower bounds, for all possible changes in its obstacles, it will not encounter any unsafe state. Then the sensor data received in the next step resolves the ambiguity it had earlier about y and an action is selected accordingly.

The abstraction refinement procedure starts with an initial set of 11 predicates. After 17 iterations, we obtained an abstract game in which the system has a winning strategy. The abstract game is computed from 45 predicates and has 2516 states. The computation takes 5.8 min in a computer with 4 GB RAM, Intel Xeon processors. The obtained predicates relating to the variable y falls into the following categories: (1) Predicates over the unobservable variable y: $y \leq -4$, $y \geq 4$, $y \leq -2.5$, $y \geq 3.5$, $y \leq -1$, $y \geq 2.5$, $y < 1.5$, $y \leq 1.5$, $y \leq 0$, $y \geq 2$, $y \leq y_o$, $y < y_s$. (2) Predicates over the observable variable y_s: $y_s \leq 1.5$, $y_s \geq 1.5$. And (3) there is no predicate over the upper bound variable y_n. The predicates relating to the obstacles $(x_p, y_p), (x_o, y_o)$ are the following: $x_p \leq x$, $x \leq x_o$, $x_p \leq 2$, $x_p \leq 7$, $x_p \leq 6$, $x_o \leq x$, $x_o < 4$, $x_p \leq 6$, $x_o > 4$, $y \leq y_o$.

With the obtained set $\mathcal{P}$ of predicates, we can decide the requirement on the precision of sensor for this task. For every $p \in \mathcal{P}$, the constants in p has at most one decimal place, for example, $y \geq 2.5$. Thus, a sensor which can reliably measure just one decimal place would suffice. Besides, there is no need to keep track of the upper bound y_n for y and also the value of y_p for obj2.

Case 2: In this case we consider the sensor model with an extra limitation: the robot cannot observe obj2 if it is in R_1, or obj1 when it is in R_2. To capture this local sensing modality, we made x_o, y_o, x_p, y_p unobservable and introduce another four auxiliary observable variables x_p^c, y_p^c and x_o^c, y_o^c. When the robot is in R_1, the values of x_o^c, y_o^c equal that of x_o and y_o. But when it is in R_2, (x_o^c, y_o^c) can be any point in R_1 following the dynamic in robot's assumption of obj1. Similar rules applied to x_p^c and y_p^c. For the same task specification, after 21 iterations, which takes about 30 min, the abstraction refinement outputs an abstract system with 8855 states using 60 predicates and finds the robot a winning strategy.

VIII. Conclusion and Future Work

We took a first step toward explicitly accounting for the effects of sensing limitations in reactive protocol synthesis. The formalism we put forward is based on partial-information, turn-based, temporal-logic games. Using witnesses for unrealizability in such synthesis problems and interpolation methods, we proposed an abstraction refinement procedure. An interpretation of this procedure is systematical identification of new sensing modalities and precision in existing sensors to be included in order to construct feasible control policies in reactive synthesis problems.

A potential bottleneck of the proposed formalism is the rapid increase in the problem size due to, for example knowledge-based subset construction. A pragmatic future direction is to consider so-called lazy abstraction methods [24] for partial observation control synthesis, so that different parts of the concrete game can be abstracted using different sets of predicates. In this manner, the system is abstracted with different degree of precision and thus its sensor model can also be configured "locally" for different parts of the system. Furthermore, besides precision, one would also be interested in refinements in sensing with respect to accuracy; therefore, extensions to partially observable stochastic two-player games are also of interest.

References

[1] H. Kress-Gazit, T. Wongpiromsarn, and U. Topcu, "Correct, reactive robot control from abstraction and temporal logic specifications," *IEEE Robotics and Automation Magazine*, vol. 18, pp. 65–74, 2011.

[2] T. Wongpiromsarn, U. Topcu, and R. Murray, "Receding horizon temporal logic planning," *IEEE Transactions on Automatic Control*, vol. 57, no. 11, pp. 2817–2830, 2012.

[3] R. Bloem, S. Galler, N. Piterman, A. Pnueli, and M. Weiglhofer, "Automatic hardware synthesis from specifications: A case study," in *Proc. Design, Automation and Test in Europe*, 2007.

[4] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Formal synthesis of embedded control software for vehicle management systems," in *Proc. AIAA Infotech Aerospace*, 2011.

[5] A. Pnueli and R. Rosner, "On the synthesis of a reactive module," in *Proc. Symposium on Principles of Programming Languages*, 1989, pp. 179–190.

[6] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive(1) designs," in *Proc. International Conference on Verification, Model Checking, and Abstract Interpretation*, vol. 3855, 2006, pp. 364–380.

[7] A. Pnueli, Y. Saar, and L. D. Zuck, "Jtlv: A framework for developing verification algorithms," in *Computer Aided Verification*. Springer, 2010, pp. 171–174.

[8] R. Bloem, A. Cimatti, K. Greimel, G. Hofferek, R. Könighofer, M. Roveri, V. Schuppan, and R. Seeber, "RATSY–a new requirements analysis tool with synthesis," in *Computer Aided Verification*. Springer, 2010, pp. 425–429.

[9] T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R. 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*. ACM, 2011, pp. 313–314.

[10] B. F. R. Dimitrova and B. Finkbeiner, "Abstraction refinement for games with incomplete information," Reports of SFB/TR 14 AVACS 43, SFB/TR 14 AVACS,, Tech. Rep., October 2008.

[11] K. R. Apt and E. Grädel, *Lectures in game theory for computer scientists*. Cambridge University Press, 2011.

[12] K. Chatterjee, L. Doyen, T. A. Henzinger, and J.-F. Raskin, "Algorithms for omega-regular games with imperfect information," in *Computer Science Logic*. Springer, 2006, pp. 287–302.

[13] A. Cimatti, S. Mover, and S. Tonetta, "SMT-based verification of hybrid systems." in *AAAI*, 2012.

[14] A. Arnold, A. Vincent, and I. Walukiewicz, "Games for synthesis of controllers with partial observation," *Theoretical computer science*, vol. 303, no. 1, pp. 7–34, 2003.

[15] M. Wulf, L. Doyen, and J.-F. Raskin, "A lattice theory for solving games of imperfect information," in *Hybrid Systems: Computation and Control*, ser. Lecture Notes in Computer Science, J. Hespanha and A. Tiwari, Eds. Springer Berlin Heidelberg, 2006, vol. 3927, pp. 153–168.

[16] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1, pp. 99–134, 1998.

[17] J. Pineau and S. Thrun, "High-level robot behavior control using pomdps," in *AAAI-02 Workshop on Cognitive Robotics*, vol. 107, 2002.

[18] H. Kurniawati, D. Hsu, and W. S. Lee, "SARSOP: Efficient point-based pomdp planning by approximating optimally reachable belief spaces." in *Robotics: Science and Systems*, 2008, pp. 65–72.

[19] R. Dimitrova and B. Finkbeiner, "Abstraction refinement for games with incomplete information," in *FSTTCS*, ser. Dagstuhl Seminar Proceedings, R. Hariharan, M. Mukund, and V. Vinay, Eds., vol. 08004. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2008.

[20] E. A. Emerson, "Temporal and modal logic." *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, vol. 995, p. 1072, 1990.

[21] K. Chatterjee, L. Doyen, T. A. Henzinger, and J.-F. Raskin, "Algorithms for omega-regular games with imperfect information," in *Computer Science Logic*, ser. Lecture Notes in Computer Science, Z. Ésik, Ed. Springer, 2006, vol. 4207, pp. 287–302.

[22] R. M. Smullyan, *First-order logic*. Courier Dover Publications, 1995.

[23] K. L. McMillan, "Interpolants from z3 proofs," in *Proceedings of the International Conference on Formal Methods in Computer-Aided Design*, ser. FMCAD '11. Austin, TX: FMCAD Inc, 2011, pp. 19–27.

[24] T. A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre, "Lazy abstraction," in *ACM SIGPLAN Notices*, vol. 37, no. 1. ACM, 2002, pp. 58–70.