Lazy Synthesis of Symbolic Output-Feedback Controllers for State-Based Safety Specifications

Mehrdad Zareian
KIT, Germany
mehrdad.zareian@kit.edu

Anne-Kathrin Schmuck
MPI-SWS, Germany
akschmuck@mpi-sws.org

ABSTRACT

This short paper presents a lazy symbolic *output-feedback* controller synthesis algorithm for *state-based* safety specifications over large transition systems. The novel idea of our approach is to integrate an *iterative* algorithm for observer design with an *online adaptable* safety controller synthesis algorithm. This allows us to iteratively update the safety controller to observer refinements and to guide these refinements by the existing controller. This results in efficient lazy synthesis of a safety controller whose domain increases with the time spent in synthesis. We present simulation results for a synthetic robot motion planning example showing the benefits of our algorithm compared to the standard approach.

ACM Reference Format:
Mehrdad Zareian and Anne-Kathrin Schmuck. 2023. Lazy Synthesis of Symbolic Output-Feedback Controllers for State-Based Safety Specifications. In *Proceedings of the 26th ACM International Conference on Hybrid Systems: Computation and Control (HSCC '23), May 09–12, 2023, San Antonio, TX, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3575870.3587111

1 INTRODUCTION

The problem of symbolic output-feedback control arises if a dynamical system is controlled through a digital interface which limits the sensing capabilities of the controller to a finite set of boolean output variables [3, 6, 7]. Such restrictions arise for example due to the deployment of sensors which only record threshold crossings of continuous variables. In contrast to classical output-feedback control, where the output is a continuous signal with a functional relationship to the continuous state variables, the *symbolic* output-feedback control problem considers an output that switches between a finite number of boolean outputs in a discontinuous manner. Due to the lack of constant feedback, the construction of an observer for controller synthesis is substantially different from classical Kalman or Luenberger observer design.

The typical approach to solve *symbolic* output-feedback control problems for dynamical systems is to first abstract the underlying continuous system both in time and in space, to obtain a (typically large but finite-state) discrete transition system. If the specification is given over the available boolean *output* variables, the controller synthesis problem reduces to solving a two-player game *under imperfect information* over a finite graph induced by the abstract transition system [11, Sec.3]. In order to solve such games, one has to first construct another transition system, called *observer*, which incurs an exponential blow-up in the state space. Only after the observer is constructed, a normal synthesis game can be solved to obtain the controller. In the context of symbolic control for continuous systems, this approach is not applicable, as the obtained abstraction typically has a very large state space that prohibits the efficient construction of an observer - let alone the subsequent solution of the synthesis game.

To overcome this challenge, we propose a new *lazy* observer design approach which interleaves observer construction and safety controller synthesis. Our technique overapproximates the true observer by a deterministic finite transition system and uses this overapproximation for controller synthesis. Thereafter, we refine our approximate observer by adding more distinguished states and transitions. By using an online safety controller synthesis algorithm, we can update the controller for the refined transition system without a full recomputation. In addition, we can utilize the information from a previous synthesis attempt to *lazily* refine the observer. This way, we yield an iterative synthesis technique that only computes tight observations for parts of the underlying system relevant for the posted synthesis question. In addition, the quality of the controller, in terms of its domain, increases with the number of refinements, and hence, with the resources spent in synthesis. We show that for a robot motion planning example with symbolic observations our technique outperforms a standard approach.

Related work. The construction of symbolic controllers was addressed in [1, 10] for omega-regular properties under *classical* output-feedback, i.e., where the output is a continuous function of the state variable. This setting allows to incorporate classical (non-)linear observer design methods *before* the abstraction step. In the distinguished setting of *symbolic* output-feedback control addressed in this paper, the problem of constructing observers for infinite state systems [7] and the classical two-step approach of observer and controller synthesis for output-based specifications was investigated [8, 9]. Iterative approaches appeared in the context of supervisory control [13], and, recently, for omega-regular specifications in the context of symbolic output-feedback control [4]. While [4] addresses the same synthesis problem as this paper, their approach is orthogonal and their tool BOCoSy does not yet scale to the size of the transition systems considered in our experiments.

2 PRELIMINARIES

Notation. We use $f : A \rightrightarrows B$ and $f : A \to B$ to denote a set-valued and ordinary map, respectively. The map f is called strict if $f(a) \neq \emptyset$ for all $a \in A$. We define the inverse map $f^{-1}(b) = \{a \in A | b \in f(a)\}$

Both authors were partially supported by DFG projects 389792660 TRR 248–CPEC and SCHM 3541/1-1. This research was conducted while the first author was at MPI-SWS.

HSCC '23, May 09–12, 2023, San Antonio, TX, USA

ACM ISBN 979-8-4007-0033-0/23/05.
https://doi.org/10.1145/3575870.3587111

and lift maps to subsets of their domain in the usual way, i.e., for $f : A \rightrightarrows B$ and $\alpha \subseteq A$ we have $f(\alpha) = \{b | \exists a \in \alpha . b \in f(a)\}$.

Systems. We consider systems $S = (X, X_0, U, F, Y, H)$ which consists of a state space X, set of initial states $X_0 \subseteq X$, a input space U, a *strict* transition function $F : X \times U \rightrightarrows X$, an output space Y , and a *strict* output function $H : X \to Y$. By some abuse of notation we associate the functions F and H sometimes with their corresponding relations $F \subseteq X \times U \times X$ and $H \subseteq X \times Y$, respectively. The system S is called finite if X, U and Y are finite sets.

Trace semantics. The set of all finite and infinite state-input traces (trace for short) of system S are denoted by $\mathrm{Tr}^*(S)$ and $\mathrm{Tr}^\omega(S)$ and consist of all traces $\pi = x_0 u_0 x_1 u_1 \ldots$ s.t. $x_0 \in X_0$ and $x_{i+1} \in F(x_i, u_i)$ for all $i \in dom(\pi)$ where $dom(\pi) = [0, k)$ for finite traces $\pi = x_0 u_0 x_1 u_1 \ldots u_{k-1} x_k$ and $dom(\pi) = \mathbb{N}$ for infinite traces. We define the map $\Omega_S : \pi \mapsto y_0 u_0 y_1 \ldots$ such that $\forall i \in dom(\pi) : y_i = H(x_i)$ to map a state-input trace to its corresponding input-output trace. We collect all finite and infinite input-output traces in the sets $\mathrm{ETr}^*(S)$ and $\mathrm{ETr}^\omega(S)$, respectively. The last state of a finite trace is $\mathrm{Last}(\pi)$.

Safety specifications. We consider safety specifications defined by a proper subset $B \subset X$ of bad states. We collect all infinite state/input and input/output traces respecting this specification in the sets $\mathrm{Tr}^\omega_{\neg B}(S) := \{\pi \in \mathrm{Tr}^\omega(S) \mid \forall i \in \mathbb{N} : \mathrm{Last}(\pi_{[0:i]}) \notin B\}$ and $\mathrm{ETr}^\omega_{\neg B}(S) := \{\pi \in \mathrm{ETr}^\omega(S) \mid \forall i \in \mathbb{N} : \mathrm{Last}(\Omega_S^{-1}(\pi_{[0:i]})) \cap B \neq \emptyset\}$, respectively. We see that, by definition, $\nu \in \mathrm{ETr}^\omega_{\neg B}(S)$ iff for all $\pi \in \Omega_S^{-1}(\nu)$ holds that $\pi \in \mathrm{Tr}^\omega_{\neg B}(S)$.

Controller. Given a system $S = (X, X_0, U, F, Y, H)$, we define a *state-feedback* and an *output-feedback* controller for S as a function $C : X \rightrightarrows U$ and $C : Y(UY)^* \rightrightarrows U$. We collect all closed loop traces of system S controlled by C in the set $\mathrm{Tr}^\omega(S, C) = \{\pi \in \mathrm{Tr}^\omega(S) \mid \forall i \in \mathbb{N} : u_{i+1} \in C(\mathrm{Last}(\pi_{[0:i]}))\}$ if C is a state-feedback controller and in the set $\mathrm{Tr}^\omega(S, C) = \{\pi \in \mathrm{Tr}^\omega(S) \mid \forall i \in \mathbb{N} : u_i \in C(\Omega_S(\pi_{[0:i-1]}))\}$ if C is an output-feedback controller. We define the *initial domain* of C by $\mathrm{dom}_0(C) := \mathrm{dom}(C) \cap X_0$ if C is a state-feedback controller, and by $\mathrm{dom}_0(C) := \Omega_S^{-1}(\mathrm{dom}(C) \cap Y) \cap X_0$ if C is an output-feedback controller. A controller C is called a *safety controller* for S w.r.t. B if $\emptyset \neq \mathrm{Tr}^\omega(S, C) \subseteq \mathrm{Tr}^\omega_{\neg B}(S)$. In addition, a safety controller C is said to have *maximal* domain if there does not exist another safety controller C' (of the same type, i.e., either state-feedback or output-feedback) s.t. $\mathrm{dom}_0(C) \subsetneq \mathrm{dom}_0(C') \subseteq X_0$.

3 PROBLEM STATEMENT

Starting with a dynamical system governed by non-linear differentail equations, we assume that this system has already been abstracted into a (large but) finite-state transition system S using standard techniques, e.g. SCOTS [12]. For this system S we consider the following synthesis problem.

Problem 3.1. *Given a* finite *system S with state space X and a safety specification $B \subseteq X$, construct an output-feedback safety controller C for S w.r.t. B.*

Problem 3.1 is a synthesis problem under partial observation - the specification (i.e., its satisfaction) is not directly observable by the output-feedback controller. The standard procedure to solve this problem consists of three steps: (i) construction of an observer, (ii) synthesis of a state-feedback controller on the observer, and (iii) refinement of the abstract state-feedback controller into an output-feedback controller for the original system. Section 4 outlines this rather standard sequential synthesis approach. Thereafter, Sec. 5 presents the main contribution of this paper, which is the integration of all three steps into a dynamic algorithm that lazily interleaves observer and controller design.

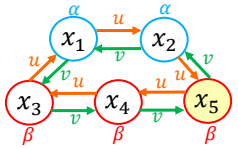

Figure 1: System S for Ex.3.1 with states $X = \{x_1, \ldots, x_5\}$, inputs $U = \{u, v\}$, outputs $Y = \{\alpha, \beta\}$ and specification $B = \{x_5\}$.

Example 3.1. We will use a running example to illustrate the constructions given in the following sections. For this, we consider the transition system illustrated in Fig. 1 with two symbolic outputs α and β, control inputs u and v and the set of bad states $B = \{x_5\}$.

4 SEQUENTIAL SYNTHESIS

This section recalls necessary concepts used in the standard solution of Problem 3.1, which executes the full observer construction (Sec. 4.1) before the synthesis step (Sec. 4.2) and then combines both (Sec. 4.3) to derive an output feedback controller.

4.1 Knowledge-based Abstraction (KA)

This section presents a particular knowledge-based abstraction (KA) algorithm, given in Alg. 1, which will allow us to integrate the observer construction with the safety controller synthesis in Sec. 5. As usual, this KA algorithm takes as input a system S with *non-deterministic* input/output behaviour and builds a *deterministic* observer $\widehat{S}$ which has the same input-output behavior as S. In particular, the states of the observer correspond to the state-subsets of S which are reachable by the same input/output trace and are therefore indistinguishable by the output-feedback controller.

Algorithm 1 Knowledge-based Abstraction (KA)

Require: $S = (X, X_0, U, F, Y, H)$;
1: $\widehat{X}_0 \leftarrow \{X_0 \cap H^{-1}(y) | y \in Y\}$; $L \leftarrow \widehat{X}_0$;
2: **while** $L \neq \emptyset$ **do**
3: $\quad L_{new} \leftarrow \emptyset$
4: $\quad$ **for** $\widehat{x} \in L, u \in U, y \in Y$ **do**
5: $\quad\quad \widehat{x}' \leftarrow F(\widehat{x}, u) \cap H^{-1}(y) \neq \emptyset$;
6: $\quad\quad L_{new} \leftarrow L_{new} \cup \{\widehat{x}'\} \setminus \widehat{X}$;
7: $\quad\quad \widehat{X} \leftarrow \widehat{X} \cup \{\widehat{x}'\}$; $\widehat{F} \leftarrow \widehat{F} \cup \{(\widehat{x}, u, \widehat{x}')\}$;
8: $\quad$ **end for**
9: $\quad L \leftarrow L_{new}$;
10: **end while**
11: $\widehat{H}(\widehat{x}) = y$ iff $y \in H(\widehat{x})$;
12: **return** $\widehat{S} = (\widehat{X}, \widehat{X_0}, U, Y, \widehat{F}, \widehat{H})$;

Alg. 1 operates on subsets of the state space. For avoiding ambiguity, we call these subsets *covers*. The algorithm initially starts with a distinct *root* vertex and successively generates a tree whose vertices are labeled by covers. The first level of the tree consists of nodes with covers which correspond to the subsets of initial states generating the same output (line 1). Afterwards, we grow the tree by exploration (line 5). Given a leaf node with cover $\widehat{x} \subseteq X$, we

compute for every input $u \in U$ a new child with cover $\widehat{x}' \subseteq X$ s.t. $\widehat{x}'$ is the subset of states which are reachable by u from $\widehat{x}$, i.e., $\widehat{x}' \subseteq F(\widehat{x}, u)$, and have the same output, i.e., there exists a $y \in Y$ s.t. $\widehat{x}' = H^{-1}(y)$. We only further explore a leaf node if there is not already another node with the same cover in the tree (line 6 & 9). The resulting KA-tree for Ex. 3.1 is depicted in Fig.2.

In order to extract the observer $\widehat{S} = (\widehat{X}, \widehat{X_0}, U, Y, \widehat{F}, \widehat{H})$ from the KA-tree, we collect all distinct covers in the abstract state set $\widehat{X}$ (line 7). Further, a transition is contained in $\widehat{F}$ if and only if there exist a transition in the tree between corresponding cover vertices. Intuitively, this maps all nodes in the KA-tree with the same cover on top of each other and removes the root-node of the tree. By construction, every state x in a cover $\widehat{x} \subseteq X$ has the same output, hence H immediately defines the abstract output function $\widehat{H}$ (line 11). The observer $\widehat{S}$ for Ex. 3.1 is shown in Fig.3.

It follows directly from the exploration-step of Alg. 1 that for any finite input/output sequence $\pi \in (UY)^*$ the set of states reached in S by any state-input trajectory $\pi' \in \Omega_S^{-1}(\pi)$ coincide with the cover of the unique state reached in $\widehat{S}$ under this input, i.e.,

$$\forall \pi \in (UY)^* \,.\, \text{Last}(\Omega_S^{-1}(\pi)) = \text{Last}(\Omega_{\widehat{S}}^{-1}(\pi)). \tag{1}$$

With this it immediately follows from the construction of $\widehat{F}$ and $\widehat{H}$ that $\Omega_S(\text{Tr}^\omega(S)) = \Omega_{\widehat{S}}(\text{Tr}^\omega(\widehat{S}))$, i.e., the input/output behaviors of S and $\widehat{S}$ coincide. We can therefore interpret $\widehat{S}$ as an observer for S and specification B by defining

$$\widehat{B} := \{\widehat{x} \in \widehat{X} | \widehat{x} \cap B \neq \emptyset\} \tag{2}$$

and observing that this implies $\Omega_{\widehat{S}}\left(\text{Tr}^\omega_{\neg\widehat{B}}(\widehat{S})\right) = \text{ETr}^\omega_{\neg\widehat{B}}(\widehat{S}) = \text{ETr}^\omega_{\neg B}(S)$ as $\widehat{H}$ is deterministic. I.e., every input/output trace ν of S for which all corresponding input/state traces $\pi \in \Omega_S^{-1}(\nu)$ are safe, corresponds to a safe input/state trace $\widehat{\pi}$ of $\widehat{S}$ w.r.t. the safety specification $\widehat{B}$ over $\widehat{S}$. In order to construct an *output-feedback* controller for S w.r.t. B it therefore suffices to construct a *state-feedback* controller for $\widehat{S}$ w.r.t. $\widehat{B}$. This is done next.

4.2 Online Safety Controller Synthesis

This section introduces a synthesis algorithm for a *state-feedback* safety controller, which allows to update the controller *online*, when states or transitions are added to the system. This algorithm is a simplified version of the online strategy synthesis algorithm for Büchi games [2], which is based on *progress measures* [5].

A *progress measure* ρ for system S is a map $\rho : X \to \{0, 1, \ldots, |X|\} \cup \{\top\}$ which assigns every state $x \in X$ of S either an integer from the set $\{0, 1, \ldots, |X|\}$ or the top element $\top$. The ordering over progress measures is the natural order over $\mathbb{N}$ and $n < \top$ as well as $\top \leq \top$, for all $n \in \{0, 1, \ldots, |X|\}$. We define

$$\rho(x) + 1 := \begin{cases} n+1 & \rho(x) = n < |X| \\ \top & otherwise \end{cases}. \tag{3}$$

Given $B \subset X$, a progress measure is called *valid* for B over S if

$$\rho(x) = \begin{cases} \max\limits_{u \in U} \left(\min\limits_{x' \in F(x,u)} (\rho(x')) \right) + 1 & x \in X \setminus B \\ 0 & x \in B \end{cases} \tag{4}$$

for all $x \in X$. In order to compute a valid progress measure over a system S w.r.t. safety specification $B \subset X$ one computes the least fixed-point over an update function which iteratively updates progress measures of single states, based on the current progress measure of their neighbours by evaluating (4). This iterative computation is initialized with the progress measure $\rho^0(x) = 0$ for all $x \in X$ and formalized in Alg. 2. It is known that the progress measure $\rho^* = \text{PMUpdate}(X, F, B, X \setminus B, \rho^0)$ computed by Alg. 2 for a system S w.r.t. B is indeed valid [2].

Algorithm 2 PMUpdate

Require: $S_d = (X, U, F)$, B, Q, ρ ;
1: **while** Q is not empty **do**
2: $x \leftarrow$ Q.dequeue();
3: **if** $x \notin B$ **then** $\rho_{new} = \max\limits_{u \in U}(\min\limits_{x' \in F(x,u)} \rho(x')) + 1$;
4: **end if**
5: **if** $\rho_{new} \neq \rho(x)$ **then**
6: $\rho(x) \leftarrow \rho_{new}$; Q.queue $(\{x_{pre} | (x_{pre}, \cdot, x) \in F\})$;
7: **end if**
8: **end while**
9: **return** ρ;

Intuitively, the valid progress measure ρ^* represents the minimal controllable distance of a state from any bad state. I.e., if $\rho^*(x) = n \in \{0, 1, \ldots, |X|\}$ the controller can ensure that the system does not visit a bad state for at least n steps, independent on how the system resolves the non-determinism in F. Therefore, only states x with valid progress measure $\rho^*(x) = \top$ allow to remain safe infinitely under control. By defining the set $X_\top := \{x \in X \mid \rho^*(x) = \top\}$ we have that

$$(\exists u \in U \,.\, F(x, u) \subseteq X_\top) \;\Leftrightarrow\; x \in X_\top \tag{5}$$

which is an immediate consequence of ρ^* being a fixed-point of the update equation in line 3 of Alg. 2. With this, we see that every controller that enables every input u with the property in (5) for every state $x \in X_\top$, defined by

$$C(x) := \begin{cases} \{u \in U \mid F(x, u) \subseteq X_\top\} & x \in X_\top \\ \text{undefined} & \text{otherwise} \end{cases}, \tag{6}$$

is indeed a maximal state-feedback safety controller for S w.r.t. B if $X_0 \cap X_\top \neq \emptyset$. That is $\emptyset \neq \text{Tr}^\omega(S, C) \subseteq \text{Tr}^\omega_{\neg B}(S)$.

Applying Alg. 2 and (6) to the observer $\widehat{S}$ w.r.t. the safety specification $\widehat{B}$ in (2), we therefore obtain a maximal state-feedback safety controller $\widehat{C}$ for $\widehat{S}$ w.r.t. $\widehat{B}$. For the system S in Fig. 1 and its observer $\widehat{S}$ in Fig. 3, the resulting valid progress measure is indicated by purple numbers on the states of $\widehat{S}$ in Fig. 3 (left) and the resulting state-feedback controller is depicted in Fig. 3 (right).

4.3 Controller Refinement

Given the observer construction and the abstract state-feedback safety controller synthesis algorithms discussed in Sec. 4.1 and Sec. 4.2, it remains to refine $\widehat{C}$ into a output-feedback safety controller C for S w.r.t. B which has maximal domain. This can be obtained by defining

$$C(\pi) := \widehat{C}(\text{Last}(\Omega_{\widehat{S}}^{-1}(\pi))). \tag{7}$$

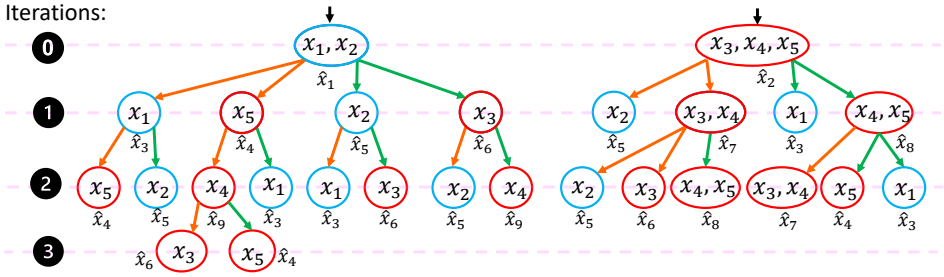

Figure 2: KA tree for Ex.3.1. All iterations are displayed. Inputs and outputs are color-coded and states are labled by their cover.

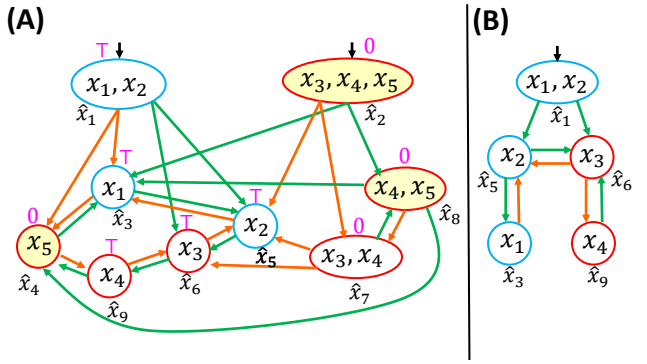

Figure 3: (A) Observer $\widehat{S}$ extracted from the KA tree in Fig. 2. Inputs and outputs are color-coded, states are labled with their cover, states in $\widehat{B}$ are yellow filled. The progress measure ρ ($\top$ or 0) is displayed in in pink. (B) Abstract controller $\widehat{C}$ extracted from $\widehat{S}$ and ρ via (6).

In order to see that C is indeed a maximal output-feedback safety controller for S w.r.t. B, recall that (1) holds. This implies from the construction of C and the fact that $\widehat{H}$ is deterministic that

$$\Omega_S(\mathrm{Tr}^\omega(S,C)) = \Omega_{\widehat{S}}(\mathrm{Tr}^\omega(\widehat{S},\widehat{C})). \tag{8}$$

As C is a safety controller with maximal domain, this immediately implies that $\widehat{C}$ is a safety controller with maximal domain as well.

5 LAZY INTEGRATED SYNTHESIS

This section presents the main contribution of this paper, which is an algorithm, called KAty, which combines observer construction and abstract safety controller synthesis in a lazy iterative manner.

Overview. KAty starts by exploring the KA tree for a fixed small number of steps. Given the knowledge on the transition structure of $\widehat{S}$ obtained during those steps, KAty constructs an abstraction $\widetilde{S}$, which allows to synthesize an abstract safety controller $\widetilde{C}$ via the progress measure algorithm in Alg. 2 along with (6). The progress measure ρ computed over $\widetilde{S}$ to extract $\widetilde{C}$ is then used to guide the further refinement of the KA tree, by only exploring "promising" branches for a fixed small number of steps. If this exploration is completed, a new, refined abstraction $\widetilde{S}'$ is extracted. The main insight that KAty exploits is that the computation of the progress measure ρ' over $\widetilde{S}'$ can be warm-started with the progress measure ρ from the previous abstraction $\widetilde{S}$. This insight makes our approach computationally efficient.

Algorithm 3 Knowledge-based Abstraction & Safety (KAty)

Require: $S = (X, X_0, U, F, Y, H)$ and $B \subseteq X$;
1: $\widehat{X}_0 \leftarrow \{X_0 \cap H^{-1}(y) | y \in Y\}$;
2: $\widehat{X} \leftarrow \{root\} \cup \widehat{X}_0$; $\widehat{F} \leftarrow \{(root, \varepsilon, \widehat{x}) | \widehat{x} \in \widehat{X}_0\}$;
3: $\rho \leftarrow \{(\widehat{x}, 0) | \widehat{x} \in \widehat{X}_0\}$;
4: $\widehat{B} = \{\widehat{x} \in \widehat{X}_0 \mid \widehat{x} \cap B \neq \emptyset\}$; $L \leftarrow \widehat{X}_0 \setminus \widehat{B}$;
5: **while** TCond == `false` **do**
6: $\widetilde{X} \leftarrow \widehat{X}$; $\widetilde{F} \leftarrow \widehat{F}$; $\widetilde{B} \leftarrow \widehat{B}$;
7: **for** $\widehat{x} \in L, u \in U, y \in Y$ **do**
8: $\widehat{x}' \leftarrow F(\widehat{x}, u) \cap H^{-1}(y) \neq \emptyset$
9: $\widehat{X} \leftarrow \widehat{X} \cup \widehat{x}'$; $\widehat{F} \leftarrow \widehat{F} \cup \{(\widehat{x}, u, \widehat{x}')\}$;
10: $\widehat{B} \leftarrow \widehat{B} \cup \{\widehat{x}' \mid \widehat{x}' \cap B \neq \emptyset\}$;
11: **if** $\exists$ a smallest $\widetilde{x} \in \widetilde{X}$ s.t. $(\widehat{x}' \subseteq \widetilde{x})$ **then**
12: $\widetilde{F} \leftarrow \widetilde{F} \cup (\widehat{x}, u, \widetilde{x})$;
13: **else**
14: $\widetilde{F} \leftarrow \widetilde{F} \cup (\widehat{x}, u, \widehat{x}')$; $\widetilde{B} \leftarrow \widetilde{B} \cup \{\widehat{x}'\}$; $\rho(\widehat{x}') \leftarrow 0$;
15: **end if**
16: **end for**
17: $\rho \leftarrow$UpdatePM$((\widetilde{X}, U, \widetilde{F}), \widetilde{B}, L, \rho)$;
18: $L \leftarrow \emptyset$;
19: **for** $\widehat{x}' \in \widehat{X} \setminus \widetilde{X}$ **do**
20: **if** ECond$(\widehat{x}')$ == `true` **then**
21: $L \leftarrow L \cup \{\widehat{x}'\}$; $\rho(\widehat{x}') \leftarrow 0$;
22: **else**
23: $\widehat{F} \leftarrow \left(\widehat{F} \setminus \{(\cdot, \cdot, \widehat{x}')\}\right) \cup \left(\widehat{F} \cap \{(\cdot, \cdot, \mathrm{f}(\widehat{x}'))\}\right)$;
24: **end if**
25: **end for**
26: **end while**
27: $\widehat{H}(\widehat{x}) = y$ iff $y \in H(\widehat{x})$;
28: **return** $\widetilde{S} = (\widehat{X}, \widehat{X}_0, U, \widehat{F}, Y, \widehat{H}), \rho$

The pseudo-code of KAty is given in Alg. 3. It starts with an initialization phase (line 1-4) which is almost identical to the initialization step in the KA algorithm (Alg. 1). At the heart of KAty is an iterative loop with three parts: (i) exploration (line 8-10), (ii) folding (line 11-15) and (iii) progress measure update (line 17). Afterwards an *exploration condition* ECond is checked, that determines which leafs are explored in the next iteration (line 20-25). This overall iteration loop of KAty terminates if a particular *termination condition* TCond is fulfilled.

Folding. The folding step of KAty is inspired by the KAM algorithm from [7]. Intuitively, folding takes leaf nodes with cover $\widehat{x}'$ of the KA tree and maps them onto nodes in the upper part of the tree with the tightest overapproximating cover $\widehat{x}''$ s.t. $\widehat{x}' \subseteq \widehat{x}''$ (line 11-15 of Alg. 3, illustrated in Fig. 4 (A) dashed purple), if one exists. This folding is sound as the overapproximation of covers ensures that

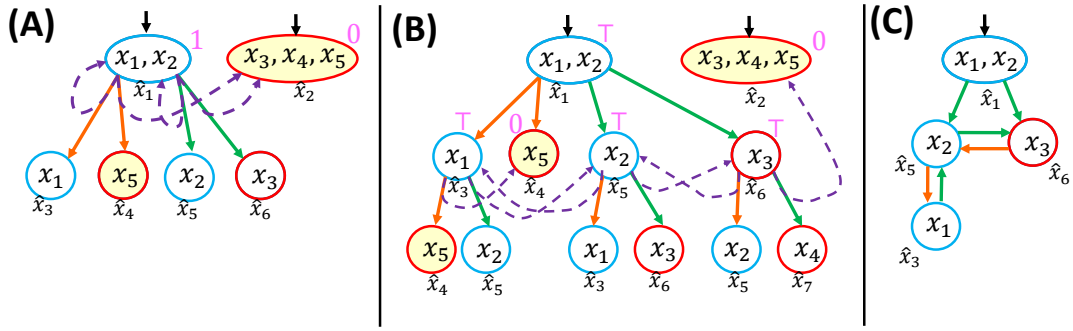

Figure 4: Illustation of the KAty tree for Ex. 3.1 after the first itation (A) and the second iteration (B) using the same visuations as in Fig. 2-3. (C) shows the controller $\widetilde{C}$ extracted from $\widetilde{S}$ after the second iteration of KAty.

every future behavior of $\widehat{S}$ starting in node $\widehat{x}'$ is overapproximated by all behaviors starting in $\widehat{x}''$. If no behavior overapproximating the future of a leaf is already explored, i.e., no upper-tree node with overapproximating cover exits, we mark the leaf node bad, i.e., add it to $\widetilde{B}$ (line 13-14). This ensures that the safety controller computed over $\widetilde{S}$ w.r.t. $\widetilde{B}$ surely underapproximates the safety controller for the original system. This is formalized in the following proposition.

Proposition 5.1. *Let S be a system with specification B, $\widehat{S} = KA(S)$ the observer of S computed via Alg. 1 with approximated specification $\widehat{B}$ as in (2) and $\widetilde{S} = (\widetilde{X}, \widehat{X}_0, U, \widetilde{F}, Y, \widetilde{H})$ a system, where $\widetilde{X}$ and $\widetilde{F}$ are computed in some iteration of KAty(S, B). Then*

$$\mathrm{ETr}^{\omega}_{\neg\widetilde{B}}(\widetilde{S}) \subseteq \mathrm{ETr}^{\omega}_{\neg\widehat{B}}(\widehat{S}). \tag{9}$$

Proof. Pick $\nu = y_0u_0y_1u_1 \ldots \in \mathrm{ETr}^{\omega}_{\neg\widetilde{B}}(\widetilde{S})$ and recall that $\widetilde{S}$ and $\widehat{S}$ are deterministic. This implies the existence of unique runs $\widetilde{\pi} = \widetilde{x}_0u_0\widetilde{x}_1u_1 \ldots \in \Omega^{-1}_{\widetilde{S}}(\nu)$ and $\widehat{\pi} = \widehat{x}_0u_0\widehat{x}_1u_1 \ldots \in \Omega^{-1}_{\widehat{S}}(\nu)$. Further, as $\nu \in \mathrm{ETr}^{\omega}_{\neg\widetilde{B}}(\widetilde{S})$ we have $\widetilde{x}_i \notin \widetilde{B}$ for all $i \in \mathbb{N}$. Hence, it follows from the construction of $\widetilde{F}$ and $\widehat{F}$ that $\widehat{x}_i \subseteq \widetilde{x}_i$ and therefore from the construction of $\widehat{B}$ and $\widetilde{B}$ that $\widehat{x}_i \notin \widehat{B}$ for all $i \in \mathbb{N}$. □

Progress Measure Updates. With this insight in place, it remains to show that we can indeed warm-start the computation of the progress measure values from the previous iteration. As the valid progress measure returned by UpdatePM is a smallest fixed-point of the update equation in line 3 of Alg. 2, the online application of this function after the transition system has changed (i.e., new leaf nodes have been added and folded), is only valid if every node's current progress measure is smaller than the valid one it would get, if we would re-run UpdatePM on the new transition system with progress measure values initialized by 0. This is shown to indeed be true for the systems $\widetilde{S}$ iteratively computed by KAty in the next proposition. Intuitively, this feature results from the fact that the overapproximation of the observer $\widetilde{S}$ and the corresponding abstract bad state set $\widetilde{B}$ is iteratively refined and therefore, it can only become easier for the controller to win in every new iteration, resulting in a higher progress measure of every node.

Proposition 5.2. *Given the premisses of Prop. 5.1 let $\tilde{\rho}_{\widetilde{S}^j}$ be the progress measure returned by UpdatePM in line 17 in the jth iteration of Alg. 3. Further, let $\rho^*_{\widehat{S}}$ be the progress measure computed by Alg. 2 on $\widehat{S} = KA(S)$ when initialized with 0. Then $\tilde{\rho}_{\widetilde{S}^{k-1}}(\widehat{x}) \leq \tilde{\rho}_{\widetilde{S}^k}(\widehat{x}) \leq \rho^*_{\widehat{S}}(\widehat{x})$ for all $\widehat{x} \in \widetilde{X}$.*

Proof. To simplify notation we denote $\tilde{\rho}_{\widetilde{S}^j}$ by $\tilde{\rho}_j$. We prove the claim by induction. For the base case, observe that all progress measures are initialized to zero and are therefore equivalent in all cases. Now assume that $k > 0$ and $\tilde{\rho}_{k-1}(\widehat{x}) \leq \tilde{\rho}_k(\widehat{x}) \leq \widehat{\rho}(\widehat{x})$. We show that $\tilde{\rho}_k(\widehat{x}) \leq \tilde{\rho}_{k+1}(\widehat{x}) \leq \widehat{\rho}(\widehat{x})$. Now let L_j be the set computed in line 21 of iteration j of Alg. 3. Then we have three cases:

▶ $\widehat{x} \notin L_k$: Then $\widetilde{F}(\widehat{x}, u) = \widehat{F}(\widehat{x}, u) = F(\widehat{x}, u)$ for all $u \in U$ by construction. As $\tilde{\rho}_k(\widehat{x}) \leq \widehat{\rho}(\widehat{x})$, the claim follows from the monotonicity of the update equation in line ... of Alg. 2 and the fact that $\widehat{\rho}$ is the smallest fixed-point of that equation.

▶ $\widehat{x} \in L_k$ and line 13 is true for *all* $u \in U$. Then $\rho(\widetilde{F}(\widehat{x}, u)) = 0$ by construction, and therefore $\tilde{\rho}_{k+1}(\widehat{x}) = 1$ if $\widehat{x} \notin \widehat{B}$ and $\tilde{\rho}_{k+1}(\widehat{x}) = 0$ otherwise. In both cases this is the minimal element of the progress measure and it follows again from monotonicity and the induction assumption that $\tilde{\rho}_{k+1}(\widehat{x}) \leq \widehat{\rho}(\widehat{x})$. In addition, $\widehat{x} \in L_k$ implies $\tilde{\rho}_k(\widehat{x}) = 0$, which proves the claim.

▶ $\widehat{x} \in L_k$ and line 11 is true for *some* $u \in U$. Then it follows from the above discussion that the progress measure for $\widehat{x}$ is determined by those inputs. Further, it follows from $\widehat{x}' \subseteq \widetilde{x}$ that $\widehat{x}' := \widehat{F}(\widehat{x}, u) \subseteq \widetilde{F}(\widehat{x}, u) =: \widetilde{x}'$. It remains to show that $\rho^k(\widehat{x}') \leq \widehat{\rho}(\widehat{x}')$. Let $\widehat{\rho}(\widehat{x}') = \gamma$. Then there exists an infinite sequence $\nu := u_0y_0u_1y_1 \ldots$ s.t. the (unique) input/state trace $\widehat{\pi} = \widehat{x}'u_0\widehat{x}_1u_1 \ldots$ compliant with ν in $\widehat{S}$ and starting in $\widehat{x}'$ visits $\widehat{B}$ after n steps if $\gamma = n$ (i.e., $\widehat{x}_n \cap B \neq \emptyset$), or never, if $\gamma = \top$ (i.e., $\widehat{x}_i \cap B = \emptyset$ for all $i \in \mathbb{N}$). Now consider the (unique) input/state trace $\widetilde{\pi} = \widetilde{x}'u_0\widetilde{x}_1u_1 \ldots$ compliant with ν in $\widetilde{S}$ and observe that, by construction, $\widehat{x}_i \subseteq \widetilde{x}_i$. With this $\widehat{x}_n \cap B \neq \emptyset$ implies $\widetilde{x}_n \cap B \neq \emptyset$, and hence $\widetilde{x}_n \in \widetilde{B}$, giving $\rho^k(\widetilde{x}') = \gamma$. Of cause, as $\widehat{x}_i \subseteq \widetilde{x}_i$, there can exists an $m < n$ s.t. $\widetilde{x}_m \in \widetilde{B}$, giving $\rho^k(\widetilde{x}') = m < \gamma$, but not vice versa. This proves the claim. □

Exploration and Termination Conditions. Based on the progress measure computed over $\widetilde{S}$ KAty decides which leaf nodes to further explore (line 20-25). That is ECond$(\widehat{x}')$ == `true` iff

(1) the cover of $\widehat{x}'$ has not yet been explored, i.e., $\widehat{x}' \notin \widetilde{X}$,
(2) $\widehat{x}'$ is not bad, i.e., $\widehat{x}' \notin \widehat{B}$,
(3) $\widehat{x}'$ cannot already be controlled to stay safe, i.e., a smallest $\widehat{x}$ existed in line 12 and has progress measure $\rho(\widehat{x}) \neq \top$, and
(4) $\widehat{x}'$ is not the ancestor of a root $\widehat{x}_0 \in \widehat{X}_0$ s.t. $\rho(\widehat{x}_0) = \top$.

If a leaf is further explored, it is added to L and its progress measure is initialized to zero. Else, the folding of this leaf is retained in $\widetilde{F}$, by keeping the folded edge $\mathsf{f}(\widehat{x}')$ (line 23).

On the other hand, the termination condition determines when KAty should terminate. We allow to either terminate if $L = \emptyset$, or if a pre-defined number of iterations is exhausted.

Controller Extraction. Whenever KAty returns the abstraction $\widetilde{S}$ along with the progress measure ρ over $\widetilde{S}$, we can then use (5) - (6) with $\widetilde{F}$ and $\widetilde{X}$ to compute the abstract safety state-feedback controller $\widetilde{C}$ for $\widetilde{S}$ w.r.t. $\widetilde{B}$. We can then refine $\widetilde{C}$ to an output-feedback controller C for S by using (7) over $\widetilde{S}$ and $\widetilde{C}$ instead of $\widehat{S}$ and $\widehat{C}$. The resulting controller is indeed a safety controller for S w.r.t. B. In particular, if KAty terminates with L being empty, we show that the domain of C is indeed maximal. This is formalized in the following theorem, which includes the main result of this paper.

Theorem 5.3. *In the context of Problem 3.1 let* $(\widetilde{S}, \rho) = KA\text{ty}(S, B)$ *s.t.* $\widetilde{X}_{0,\top} \neq \emptyset$ *and* C *extracted via* (5)-(7) *as discussed before. Then* C *is an output-feedback safety controller for* S *w.r.t.* B*. Further, if* $L = \emptyset$ *upon termination,* C *has maximal initial domain.*

Proof. We first prove that C is a safety controller. Let $\widehat{S} = \mathrm{KA}(S)$. Then it follows from Alg. 3 that $\widetilde{X} \subseteq \widehat{X}$. It further follows from the proof of Prop. 5.2 that $\widehat{F}(\widetilde{x}, u) \subseteq \widetilde{F}(\widetilde{x}, u)$ and $\widetilde{X}_\top \subseteq \widehat{X}_\top$. With this, it follows from the construction of $\widetilde{C}$ and the fact that $\widetilde{H}$ is still deterministic, that

$$\Omega_S(\mathrm{Tr}^\omega(S, C)) = \Omega_{\widehat{S}}(\mathrm{Tr}^\omega(\widehat{S}, \widehat{C})) \subseteq \Omega_{\widetilde{S}}(\mathrm{Tr}^\omega(\widetilde{S}, \widetilde{C})). \tag{10}$$

It further follows from Prop. 5.1, the construction of C and the fact that $\widetilde{C}$ is a state-feedback controller for $\widetilde{S}$ that

$$\Omega_S(\mathrm{Tr}^\omega(S, C)) \subseteq \Omega_{\widetilde{S}}(\mathrm{Tr}^\omega_{\neg\widetilde{B}}(\widetilde{S})) = \mathrm{ETr}^\omega_{\neg\widetilde{B}}(\widetilde{S}) \subseteq \mathrm{ETr}^\omega_{\neg B}(S) \tag{11}$$

which implies that C is a safety controller for S.

Now assume $L = \emptyset$ upon termination. We need to show that for all $\widehat{x}_0 \in \widehat{X}_0$ s.t. $\rho(\widehat{x}_0) < \top$ holds that the progress measure $\check{\rho}$ of the same node returned from first applying KA and then PMUpdate is also not top, i.e., $\rho(\check{\widehat{x}}_0) < \top$. Due to the similarities of Alg. 1 and Alg. 3 this implies that there exists a ancestor $\widehat{x} \in \widehat{X}$ of $\widehat{x}_0$ with $\rho(\widehat{x}) < \top$ and $\check{\rho}(\widehat{x}) = \top$. As $L = \emptyset$ we know that for any such $\widehat{x}$ one of the four exploration conditions is violated. If (ii) is violated $\rho(\widehat{x}) = \check{\rho}(\widehat{x}) = 0$ in all cases, if (iii) is violated $\rho(\widehat{x}) = \check{\rho}(\widehat{x}) = \top$ and if (iv) is violated $\rho(\widehat{x}_0) = \top$. Hence, the only interesting case is whenever (i) is violated. However, in this case the folding does not introduce any over-approximation as the future of $\widehat{x}$ and the already explored node $\widetilde{x}'$ it is folded to, have the same future. This implies again that $\rho(\widehat{x}) = \check{\rho}(\widehat{x})$, which concludes the proof. □

Example. Fig. 4 depicts two iteration of KAty on Ex. 3.1. In the first iteration (A), we start with two initial states $\widehat{x}_1 = \{x_1, x_2\}$ and $\widehat{x}_2 = \{x_3, x_4, x_5\}$ where $\widehat{x}_2 \in \widehat{B}$ and hence $L = \{\widehat{x}_1\}$. I.e., we only explore $\widehat{x}_1$, and observe that all resulting leaves can be folded (dashed purple edges). Running the progress measure algorithm on $\widehat{S}$ (which only consists of $\widehat{x}_1$ and $\widehat{x}_2$ connected by the folding edges in this case) results in the two indicated progress measures. As $\widetilde{X}_\top = \emptyset$, no safety controller can be extracted. We further observe that for all nodes except for node $\widehat{x}_4 = \{x_5\} \in \widehat{B}$ the exploration condition is true and they are explored in the second iteration (B).

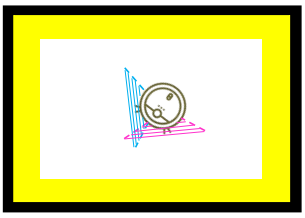

Figure 5: Example robot equipped with 2 sensors.

Again all new leaves can be folded and the progress measure can be updated on the new state set $\widehat{X}$. Now we can extract a controller $\widehat{C}$ via (6) which is depicted in Fig. 4 (C). We see that without condition (iv) in place, KAty would run for one more iteration to only explore node $\{x_4\}$. The resulting controller would then coincide with the one depicted in Fig. 3 (B). However, in order to get a controller with a maximal initial domain, i.e., a controller that can keep the system safe from all initial states from which such a controller exists, two iterations suffice.

6 EXPERIMENTAL EVALUATION

We implemented KAty in an OCaml-based tool with a BDD backend and applied it to a robot motion planning problem under partial observation, depicted in Fig. 5. We run the experiment on a cluster with Intel Xeon E5-2667 v2, with 256 GB of RAM.

We consider a robot with continuous dynamics described by the following differental equation

$$\dot{x}_1 = u_1 cos(x_3) \qquad \dot{x}_2 = u_1 sin(x_3) \qquad \dot{x}_3 = u_2$$

where x_1 and x_2 represent the position of the robot and x_3 represents its heading. We used SCOTS [12] to construct a finite-state abstraction of this system with time and space discretization parameters $\tau = 0.3$ and $\eta = [0.2, 0.2, 0.25\pi]$ over the restricted state space $X = [0, 5] \times [0, 5] \times [-1.25\pi, 1.25\pi]$ and input space $U = \{(1, 0), (1, \pi), (1, -\pi), (1, 2\pi), (1, -2\pi), (1, 3\pi), (1, -3\pi)\}$. This returned a transition system with 7436 states and 646875 transitions.

We have added symbolic outputs to the problem modelling the fact that the robot has only two radar sensors that tell it whether it is close to a wall or not. This situation is depicted in Fig. 5. Here, the blue and the violet sensor trigger if the robot moves to the yellow region, giving in total 4 distinct logical outputs (blue on/off and/or pink on/off), depending on the orientation and location of the robot. The safety objective of the robot is to not collide with the (black) wall, which happens if he senses the yellow region but keeps moving towards the direction of the wall. The initial condition of the robot is the entire state space.

We applied our tool on this example. We find an output-feedback safety controller with maximal initial domain in 12694 seconds. The constructed KA tree has 2842 states, however, the extracted output-feedback controller has only 66 states. Our BDD-based re-implementation of the standard KA algorithm from Alg. 1 times out after 24h without returning an observer, with which we could have initialized controller synthesis. Reducing the precision of the abstraction does not help in this case, as then no feasible control strategy under the given sensing constrains exists. Intuitively, the discretization needs to be fine enough to allow the robot to react to a wall detection in a timely and precise manner.

REFERENCES

[1] WA Apaza-Perez, Antoine Girard, Christophe Combastel, and Ali Zolghadri. 2020. Symbolic observer-based controller for uncertain nonlinear systems. *IEEE Control Systems Letters* 5, 4 (2020), 1297–1302.

[2] Krishnendu Chatterjee and Monika Henzinger. 2014. Efficient and dynamic algorithms for alternating Büchi games and maximal end-component decomposition. *Journal of the ACM (JACM)* 61, 3 (2014), 1–40.

[3] Donglei Fan and Danielle C. Tarraf. 2018. Output Observability of Systems Over Finite Alphabets With Linear Internal Dynamics. *IEEE Trans. Automat. Control* 63, 10 (2018), 3404–3417. https://doi.org/10.1109/TAC.2018.2793463

[4] Bernd Finkbeiner, Kaushik Mallik, Noemi Passing, Malte Schledjewski, and Anne-Kathrin Schmuck. 2022. BOCoSy: Small but Powerful Symbolic Output-Feedback Control. In *25th ACM International Conference on Hybrid Systems: Computation and Control* (Milan, Italy) *(HSCC '22)*. Association for Computing Machinery, New York, NY, USA, Article 24, 11 pages.

[5] Marcin Jurdziński. 2000. Small Progress Measures for Solving Parity Games. In *STACS 2000*, Horst Reichel and Sophie Tison (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 290–301.

[6] Steven M LaValle et al. 2012. Sensing and filtering: A fresh perspective based on preimages and information spaces. *Foundations and Trends® in Robotics* 1, 4 (2012), 253–372.

[7] Rupak Majumdar, Necmiye Ozay, and Anne-Kathrin Schmuck. 2020. On abstraction-based controller design with output feedback. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*. 1–11.

[8] Masashi Mizoguchi and Toshimitsu Ushio. 2018. Deadlock-free output feedback controller design based on approximately abstracted observers. *Nonlinear Analysis: Hybrid Systems* 30 (2018), 58–71.

[9] Thomas Moor, Jörg Raisch, and Siu O'young. 2002. Discrete supervisory control of hybrid systems based on l-complete approximations. *Discrete Event Dynamic Systems* 12, 1 (2002), 83–107.

[10] Giordano Pola, Maria Domenica Di Benedetto, and Alessandro Borri. 2019. Symbolic control design of nonlinear systems with outputs. *Automatica* 109 (2019), 108511.

[11] Jean-François Raskin, Thomas A Henzinger, Laurent Doyen, and Krishnendu Chatterjee. 2007. Algorithms for omega-regular games with imperfect information. *Logical Methods in Computer Science* 3 (2007).

[12] Matthias Rungger and Majid Zamani. 2016. SCOTS: A tool for the synthesis of symbolic controllers. In *Proceedings of the 19th international conference on hybrid systems: Computation and control*. 99–104.

[13] Jung-Min Yang, Thomas Moor, and Jörg Raisch. 2020. Refinements of behavioural abstractions for the supervisory control of hybrid systems. *Discrete Event Dynamic Systems* 30, 3 (2020), 533–560.