On Abstraction-Based Controller Design With Output Feedback

Rupak Majumdar
MPI-SWS, Germany

Necmiye Ozay
Univ. of Michigan, Ann Arbor, USA

Anne-Kathrin Schmuck
MPI-SWS, Germany

ABSTRACT

We consider abstraction-based design of *output-feedback* controllers for dynamical systems with a finite set of inputs and outputs against specifications in linear-time temporal logic. The usual procedure for abstraction-based controller design (ABCD) first constructs a *finite-state* abstraction of the underlying dynamical system, and second, uses reactive synthesis techniques to compute an abstract *state-feedback* controller on the abstraction. In this context, our contribution is two-fold: (I) we define a suitable relation between the original system and its abstraction which characterizes the soundness and completeness conditions for an abstract *state-feedback* controller to be refined to a concrete *output-feedback* controller for the original system, and (II) we provide an algorithm to compute a *sound finite-state abstraction* fulfilling this relation.

Our relation generalizes feedback-refinement relations from ABCD with state-feedback. Our algorithm for constructing sound finite-state abstractions is inspired by the simultaneous reachability and bisimulation minimization algorithm of Lee and Yannakakis. We lift their idea to the computation of an observation-equivalent system and show how *sound abstractions* can be obtained by stopping this algorithm at any point. Additionally, our new algorithm produces a realization of the topological closure of the input/output behavior of the original system if it is finite-state realizable.

1 INTRODUCTION

Controller synthesis for dynamical systems against specifications in linear temporal logic is a core problem in correct-by-construction design of cyber-physical systems. One way to solve this problem relies on abstracting the state space to a finite-state system, followed by algorithmic techniques from reactive synthesis to compute an abstract controller which is then refined to a concrete one for the original system [1, 7, 21, 25]. Most algorithms, and certainly most state-of-the-art synthesis tools such as SCOTS [22], pFaces [11], or Mascot [10], implement this abstraction-based control design (ABCD) workflow while assuming the entire state of the underlying system to be observable. In this paper, we relax the condition of full state observation. We consider ABCD when the system has a finite number of observable outputs and a controller must decide its input choice (from a finite set) based solely on the history of applied inputs and observed outputs. Such *output-feedback control* is common in control design, as the observation of the state is usually limited by the availability and precision of the sensors.

As an example, consider the tank reactor shown in Fig. 1. It has a finite number of water level sensors ($l_0, \dots, l_5$) which indicate whether the current water level touches the sensor or not by returning true or false. Further, it can be observed (but not controlled) whether the outlet valve is open (o = `true`) or closed (o = `false`). The controller can set the inlet valve open (by applying $u = +$) or closed (by applying $u = 0$). The actual state of the system, i.e., the precise value of the water level, is not observable. In this example,

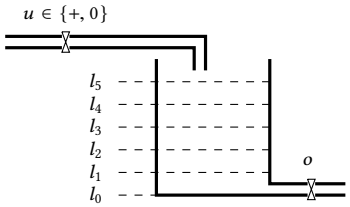

Figure 1: Tank reactor modeled as a dynamical system S over an infinite bounded state space $X \subset \mathbb{R}^3$ with finite input space $U = \{+, 0\}$ and finite output space $Y \subseteq 2^\sigma$ denoting the set of sensors $\sigma = \{l_0, \dots, l_5, o\}$ which are currently 'true.'

a given input/output sequence of observed true sensor values and applied inputs (e.g., $\nu = \{l_0\}\{+\}\{l_0\}\{+\}\{l_0, l_1, o\}\{0\}\{l_0, o\}\{+\}\dots$) provides a certain knowledge about the current true state (i.e., real water level value) of the tank system, which might be sufficient to implement a controller ensuring the satisfaction of a specification over the observables. For example, one might want to ensure that the tank never overflows (i.e., l_5 never becomes true) while still containing a limited amount of water (i.e., l_1 is always true). We show how finite-state abstractions of the input/output behavior of such an infinite state dynamical system can be constructed for the purpose of ABCD with output-feedback.

There is a rich history of output-feedback control design for continuous dynamical systems w.r.t. classical control objectives (such as stability or tracking) based on observer design [13, 24], with recent extensions to systems with finite external alphabets [6] and estimator-based abstractions for control with partial-information [5, 8, 15]. In the context of temporal-logic control of *finite-state* systems, output-feedback control gives rise to games of incomplete information [3, 5, 19]. The construction of finite-state abstractions of input/output traces for the purpose of output-feedback control is further enabled by so called l-complete abstractions [17, 20, 23, 29]. Here, the underlying state dynamics of the original system are typically not assumed to be known, which is in contrast to the situation commonly handled in ABCD for dynamical systems.

In this paper we connect the above listed lines of work by building a sound ABCD framework for synthesizing output-feedback controllers for infinite-state dynamical systems with finite input and output sets. In this context, our contribution is two-fold.

(I) We define *sound abstractions* for ABCD under output feedback by relating *states* of the abstract system to the *external input/output traces* of the original system which directly allows to refine an abstract *state-feedback* controller to an *output-feedback* controller on the original system. Our relation generalizes feedback-refinement relations (FRR) [21] to systems with inputs and outputs and is inspired by the framework of abstract interpretation [4], which formalizes the interpretation of a given abstraction function over different system semantics.

(II) We provide an algorithm to compute a *sound finite-state abstraction* of the original infinite-state system, which we call KAM,

the Knowledge-based Abstraction with Minimization algorithm. It combines two distinct ideas. First, it utilizes the *forward* computation of a *Knowledge-based Abstraction* (KA) typically used to solve partial observation games over *finite-state* systems [3, 19]. Second, it deploys a *backward* partition refinement algorithm for bisimulation-equivalence [9, 18] to construct the language equivalence quotient of a given system. Neither algorithm is guaranteed to terminate for infinite-state systems, even if there exists an exact finite-state realization of the input/output behavior of the original system. The KAM algorithm *simultaneously* executes the KA algorithm *forward*, and the *M*inimization of sets through refinement of partitions *backward* and computes a finite-state realization of the topological closure of the input/output behavior of the original system if it exists. Further, stopping KAM after any finite number of iterations returns a *sound finite-state abstraction*, even if no finite-state realization exists.

The minimization part of KAM is inspired by the simultaneous reachability and bisimulation minimization algorithm of Lee and Yannakakis [12]. However, as we are aiming at constructing an observation- (not bisimulation-) equivalent system, our algorithm only applies predecessor operations and intersection with outputs, but does not take set differences. This is, indeed, in contrast to any algorithm that constructs bisimulation relations, and is crucial in implementations. For example, one can implement KAM for linear dynamical systems by only manipulating convex polyhedra, as convexity is maintained by both predecessor operations and intersections, but not by set difference.

To decide when KAM should terminate it must recognize when the current abstraction captures the reachable portion of the language equivalence quotient, which is undecidable in general. Thus, for infinite-state systems, KAM might not realize when it should terminate, even though it may have constructed the language equivalence quotient. This is also the case for the Lee-Yannakakis algorithm and the construction of l-complete abstractions.

We tackle the termination problem similar to the l-complete abstraction framework [17]. Since KAM always constructs sound abstractions of the original system, we can run a synthesis procedure at any point to see if an abstract controller ensuring the specification exists. If a controller can be found, the abstraction construction can stop. If not, the construction continues until we try again after a future iteration. This iterative ABCD procedure is sound and relatively complete—if a topologically closed finite-state abstraction that allows to construct an abstract controller for the given specification exists, our procedure will eventually find it.

2 PRELIMINARIES

Notation. We use the symbols $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$, and $\mathbb{R}_{>0}$ to denote the sets of natural numbers, integers, reals, and positive reals, respectively. Given $a, b \in \mathbb{R}$ s.t. $a \leq b$, we denote by $[a, b]$ a closed interval and define $[a; b] = [a, b] \cap \mathbb{Z}$ as its integer counterpart. For a set W, we write W^* and W^ω for the sets of finite and infinite sequences over W, respectively. For $w \in W^*$, we write $|w|$ for the length of w and ε for the empty string with $|\varepsilon| = 0$; the length of $w \in W^\omega$ is ∞. We define $\mathrm{dom}(w) = \{0, \ldots, |w| - 1\}$ if $w \in W^*$, and $\mathrm{dom}(w) = \mathbb{N}$ if $w \in W^\omega$. For $k \in \mathrm{dom}(w)$ we write $w(k)$ for the k-th symbol of w and $w|_{[0;k]}$ for the restriction of w to the domain $[0; k]$. Given two sets A and B, $f : A \rightrightarrows B$ and $f : A \to B$ denote a set-valued and ordinary map, respectively. f is called *strict* if $f(a) \neq \emptyset$ for all $a \in A$. The inverse mapping $f^{-1} : B \rightrightarrows A$ is defined via its respective binary relation: $f^{-1}(b) = \{a \in A \mid b \in f(a)\}$. By slightly abusing notation, we lift maps to subsets of their domain in the usual way, i.e., for a set-valued map $f : A \rightrightarrows B$ and $\alpha \subseteq A$ we have $f(\alpha) = \{b \mid \exists a \in \alpha \,.\, b \in f(a)\}$, and similarly for ordinary maps.

Systems. A *system* $S = (X, X_0, U, F, Y, H)$ consists of a state space X, a set of initial states $X_0 \subseteq X$, a *finite* input space U, a strict set-valued transition function $F : X \times U \rightrightarrows X$, a *finite* output space Y, and an output function $H : X \to Y$. To simplify notation, we assume that H respects X_0, that is, if $H^{-1}(y) \cap X_0 \neq \emptyset$ we have $H^{-1}(y) \subseteq X_0$. The system S is called *finite state* if X is finite.

Trace Semantics. A *path* of S is an infinite sequence $\pi = x_0 u_0 x_1 u_1 \ldots$ such that $x_0 \in X_0$ and for all $k \in \mathbb{N}$ we have $x_{k+1} \in F(x_k, u_k)$. The set of all paths over S is denoted by $\mathrm{Paths}(S)$. The *prefix up to* x_n of a path π over S is denoted by $\pi_{[0;n]}$ with length $|\pi_{[0;n]}| = n + 1$ and last element $\mathrm{Last}(\pi_{[0;n]}) = x_n$. The set of all such prefixes is denoted by $\mathrm{Prefs}(S)$.

The unique *external* sequence of a path π of S is defined as $\mathrm{Ext}(\pi) = y_0 u_0 y_1 u_1 \ldots$, where $y_k = H(x_k)$ for all $k \in \mathbb{N}$. The sets of all external sequences over S are denoted by $\mathrm{Ext}(S)$ and we define $\mathrm{EPrefs}(S) := \mathrm{Ext}(\mathrm{Prefs}(S))$. The set $\mathrm{Ext}(S)$ is called *topologically closed* (or *closed* for short) if for any infinite sequence $\nu = y_0 u_0 y_1 u_1 \ldots \in Y(UY)^\omega$, whenever $\nu_{[0;k]} \in \mathrm{EPrefs}(S)$ for all $k \in \mathbb{N}$ it holds that $\nu \in \mathrm{Ext}(S)$. We say that S has *closed external behavior* if $\mathrm{Ext}(S)$ is closed (see, e.g., [28] for details).

We lift the map Last to external sequences and write $x \in \mathrm{LastX}_S(\rho)$ if there exists $\pi \in \mathrm{Prefs}(S)$ s.t. $\rho = \mathrm{Ext}(\pi)$ and $x = \mathrm{Last}(\pi)$. For a state $x \in X$ we define all prefixes of S that reach x as $\mathrm{Hist}_S(x) = \{\pi \in \mathrm{Prefs}(S) \mid \mathrm{Last}(\pi) = x\}$ and all external sequences generated by such prefixes as $\mathrm{EHist}_S(x) = \{\rho \in \mathrm{EPrefs}(S) \mid x \in \mathrm{LastX}_S(\rho)\}$. If the system S we are referring to is clear from the context we omit the subscript S from the maps LastX and EHist.

Control Strategies. We define *state-feedback* and *output-feedback control strategies* as functions $C^\dagger : \mathrm{Prefs}(S) \to U$ and $C : \mathrm{EPrefs}(S) \to U$, respectively. We say that a path π of S is *compliant* with C (resp. $C^\dagger$) if for all $k \in \mathbb{N}$, we have $u(k) = C(\mathrm{Ext}(\pi_{[0;k-1]}))$ (resp. $u(k) = C^\dagger(\pi_{[0;k-1]})$). We denote the set of all paths and prefixes of S compliant with C by $\mathrm{CPaths}(S, C)$ and $\mathrm{CPrefs}(S, C)$, respectively. We further use $\mathrm{Ext}(S, C)$ and $\mathrm{EPrefs}(S, C)$ to denote the sets $\mathrm{Ext}(\mathrm{CPaths}(S, C))$ and $\mathrm{Ext}(\mathrm{CPrefs}(S, C))$, respectively. For a state-feedback controller $C^\dagger$ all sets are defined analogously. It should be noted that by defining *compliance* of a controller C with a system S over the set of path prefixes, the set $\mathrm{Ext}(S, C)$ is topologically closed if $\mathrm{Ext}(S)$ is.

Control Problem. We consider ω-regular specifications over a finite set of atomic input and output propositions AP_I and AP_O. We omit the standard definitions of ω-regular languages (see, e.g., [26, 27]). To simplify notation, we assume that $U = 2^{\mathsf{AP}_I}$ and $Y = 2^{\mathsf{AP}_O}$. In this setting, an ω-regular specification ψ can be written as a language $\langle\!\langle \psi \rangle\!\rangle \subseteq Y(UY)^\omega$ of desired external sequences. Given a system S and a specification ψ, the *output-feedback control problem*, written $\langle S, \psi \rangle$, asks to find an output-feedback control strategy C such that $\mathrm{Ext}(S, C) \subseteq \langle\!\langle \psi \rangle\!\rangle$. We define $\mathcal{W}(S, \psi) =$

$\{C \mid \mathrm{Ext}(S, C) \subseteq [\![\psi]\!]\}$ as the set of all such output-feedback control strategies. For a state-feedback controller $C^\dagger$, we define analogously the set $\mathcal{W}^\dagger(S, \psi)$.

3 ABSTRACTION-BASED CONTROLLER DESIGN WITH OUTPUT-FEEDBACK

Abstraction-Based Controller Design (ABCD) is a well-known approach to solving a controller synthesis problem for a dynamical system S against specifications defined by a language $[\![\psi]\!]$. Here, the dynamical system S is first abstracted to a finite-state system $\widehat{S}$ and then techniques from reactive synthesis (e.g., [14, 27]) are used to design an abstract controller for $\widehat{S}$ ensuring ψ.

In this section, we will formalize the required relation between S and $\widehat{S}$ to refine an abstract *state-feedback* controller $\widehat{C}^\dagger$ on $\widehat{S}$ to an *output-feedback* controller C on S. We start our formalization by providing a general definition of *sound abstractions* in Sec. 3.1 which adapts feedback refinement relations [21] to systems with finite input and output sets. We show that for this definition the usual refinement of an abstract state-feedback controller to a concrete *state-feedback controller* carries over from [21]. As the main contribution of this section, we then show in Sec. 3.2 that the definition of sound abstraction needs to be applied to the external trace semantics of S rather than to its state transitions to allow for ABCD with *output feedback control*.

3.1 Sound Abstractions

Given two systems we define a *sound abstraction* as follows.

Definition 3.1. Let $S = (X, X_0, U, F, Y, H)$ and $\widehat{S} = (\widehat{X}, \widehat{X}_0, U, \widehat{F}, \widehat{Y}, \widehat{H})$ be systems. Further, let $\alpha : X \rightrightarrows \widehat{X}$ and $\gamma : \widehat{X} \rightrightarrows X$ be two set valued functions s.t. $x \in \gamma(\widehat{x})$ iff $\widehat{x} \in \alpha(x)$. Then we call $\widehat{S}$ a *sound abstraction* of S, written $S \preccurlyeq^{\gamma}_{\alpha} \widehat{S}$, if

(A1) $\alpha(X_0) \subseteq \widehat{X}_0$,
(A2) $\forall x \in X, u \in U \,.\, \alpha(F(x,u)) \subseteq \widehat{F}(\alpha(x), u)$, and
(A3) $\forall \widehat{x} \in \widehat{X} \,.\, H(\gamma(\widehat{x})) \subseteq \{\widehat{H}(\widehat{x})\}$.

$\widehat{S}$ is a *sound realization* of S, written $S \cong^{\gamma}_{\alpha} \widehat{S}$, if $S \preccurlyeq^{\gamma}_{\alpha} \widehat{S}$ and $\widehat{S} \preccurlyeq^{\alpha}_{\gamma} S$.

As common in abstract interpretation [4], we make γ explicit in Def. 3.1 to emphasize that $\{\widehat{x}\} \subseteq \alpha(\gamma(\widehat{x}))$, where equality may not hold. However, to simplify notation, we often omit γ and write $\preccurlyeq_\alpha$ and $\cong_\alpha$, as γ is fully determined by knowing α. Further, we write $\preccurlyeq$ to indicate that there exists α s.t. $\preccurlyeq_\alpha$ holds.

Remark 1. *Sound abstractions are an adaptation of feedback refinement relations (FRR) [21, Def. V.2] to systems with finite input and output sets in the following sense.*

(A1): *An FRR is defined for fully initialized systems (i.e., $X_0 = X$), where (A1) follows from the fact that an FRR must be a* strict *relation.*

(A2): *To simplify notation, we assume that F is a strict function*[1]*. This implies that all inputs are enabled in every state, i.e.,* $\mathrm{Enab}_S(x) = \{u \in U \mid F(x,u) \neq \emptyset\} = U$ *for all $x \in X$. The definition of FRR makes* $\mathrm{Enab}(x)$ *explicit by replacing (A2) with the two conditions*
(A2. 1) $\forall x \in X \,.\, \mathrm{Enab}_{\widehat{S}}(\alpha(x)) \subseteq \mathrm{Enab}_S(x)$, *and*
(A2. 2) $\forall x \in X, u \in \mathrm{Enab}_{\widehat{S}}(\alpha(x)) \,.\, \alpha(F(x,u)) \subseteq \widehat{F}(\alpha(x), u)$
which coincide with (A2) if $\mathrm{Enab}(x) = U$.

(A3): *An FRR is defined for systems with full state observation, i.e., $Y = X$, $\widehat{Y} = \widehat{X}$ and $\widehat{H} = H = \mathrm{id}$ with $\mathrm{id}(x) = x$ for all $x \in X$. This renders Y infinite if X is infinite and does not allow the direct interpretation of an ω-regular specification over U and Y. While our condition (A3) enables the use of a common specification for both S and $\widehat{S}$ (due to their equivalent finite input/output spaces), this is not possible in [21], due to Y being infinite and $Y = X \neq \widehat{X} = \widehat{Y}$. [21, Def.VI.2] handles this by defining a different* abstract specification *from the defined FRR and the specification over the original system S.*

Observe that for a system S and its sound abstraction $\widehat{S}$, corresponding states in two runs $x_0 u_0 x_1 \ldots$ and $\hat{x}_0 u_0 \hat{x}_1 \ldots$ stay related by α during arbitrarily but finite executions, if they start at related initial states $\hat{x}_0 \in \alpha(x_0)$ (A1) and the same input sequence is applied (A2). In this case (A3) ensures that S always produces a subset of the outputs generated by $\widehat{S}$ in every instance of the trace. This implies that any arbitrarily but *finite* external sequence v generated by ξ is contained in $\mathrm{EPrefs}(\widehat{S})$. Therefore, any abstract controller solving a given control problem over $\widehat{S}$ can be guaranteed to be refinable to a sound controller for S, if $\widehat{S}$ has *closed external behavior*. If this is not the case, spurious infinite external traces generated by this controller on S which are not contained in $\mathrm{Ext}(\widehat{S})$ might violate the specification. Requiring $\widehat{S}$ to have closed external behavior is not with loss of much generality in ABCD: any finite-state system (of the form considered in this paper) has closed external behavior, and we require $\widehat{S}$ to be finite-state in order to apply reactive synthesis techniques for abstract controller design anyways. The next theorem formalizes the above discussion for ABCD with state feedback. The proof uses the same insights as the proof of [21, Thm.VI.3] and is therefore only provided in the appendix.

Theorem 3.2. *Let S and $\widehat{S}$ be systems s.t. $\widehat{S}$ has closed external behavior. If $S \preccurlyeq_\alpha \widehat{S}$ and $\widehat{C}^\dagger \in \mathcal{W}^\dagger(\widehat{S}, \psi)$ then $C^\dagger = \widehat{C}^\dagger \circ \alpha \in \mathcal{W}^\dagger(S, \psi)$. Further, if S has closed external behavior and $S \cong_\alpha \widehat{S}$ then $\mathcal{W}^\dagger(S, \psi) = \emptyset$ iff $\mathcal{W}^\dagger(\widehat{S}, \psi) = \emptyset$.*

3.2 Sound Abstractions for Output Feedback

Now we consider the case of output feedback. Here, the only available information about the system S that we can utilize for control are external prefixes $v \in \mathrm{EPrefs}(S)$. With this, however, we usually cannot uniquely determine the current state of the system, i.e., $\mathrm{LastX}(v)$ is usually a set of states and not a singleton. Further, it is well known that any state of a system S possesses the Markovian property, that is, knowing the current state of the system is enough to uniquely determine all its future behaviors, which is utilized in (A2) of Def. 3.1. This is, however, not true for the output space Y. In general, one needs to look at the entire history seen so far, i.e., at the generated string $v \in \mathrm{EPrefs}(S)$, to uniquely determine all future observable behaviors of this system. This intuition is captured by the so called *external trace system* $S^\star$ of S in which a state represents a finite external history of S, and the transitions extend the external history by one step.

Definition 3.3. Given a system $S = (X, X_0, U, F, Y, H)$, its induced *external trace system* is the system $S^\star =$

[1] See Rem. 2 in Sec. 4.1 for a discussion of this choice.

$(X^\star, X_0^\star, U, F^\star, Y, H^\star)$, where $X^\star := \text{EPrefs}(S)$, $X_0^\star := H(X_0)$, $F^\star(\rho, u) := \{\rho u y \mid F(\text{LastX}(\rho), u) \cap H^{-1}(y) \neq \emptyset\}$ and $H^\star(\rho) := \text{Last}(\rho)$.

It should be noted that, by definition, $S^\star$ has closed external behavior. We further have $\text{EPrefs}(S) = \text{EPrefs}(S^\star)$, $\text{Ext}(S) \subseteq \text{Ext}(S^\star)$, and $\text{Ext}(S) = \text{Ext}(S^\star)$ iff S has closed external behavior. That is, $\text{Ext}(S^\star)$ is the *behavioral closure* of $\text{Ext}(S)$ [28].

To refine an abstract state-feedback controller to an output-feedback controller for the original system, one needs to relate abstract states to external prefixes of S. As the latter form the state space of $S^\star$, such a refinement is possible if $\widehat{S}$ is a sound abstraction of $S^\star$. More precisely, it follows from Thm. 3.2 that $S^\star \preccurlyeq \widehat{S}$ implies that a state-feedback control strategy $\widehat{C}^\dagger : \text{Prefs}(\widehat{S}) \to U$ for $\widehat{S}$ can be refined into a *state-feedback* control strategy $C^{\star\dagger} : \text{Prefs}(S^\star) \to U$ for the external trace system $S^\star$ of S. Now recalling the definition of $S^\star$'s state space $X^\star := \text{EPrefs}(S)$, we see that for a string $\xi_0 u_0 \xi_1 u_1 \ldots \xi_k \in \text{Prefs}(S^\star)$ we have $\xi_i = \xi_k|_{[0;i]}$ for all $i \in [0;k]$. Therefore, ξ_k carries all information needed for $C^{\star\dagger}$'s control choice. $C^{\star\dagger}$ can therefore be redefined into a memoryless strategy $C^\star : X^\star \to U$, which, by definition, is an output-feedback control strategy for the original system S (as $X^\star := \text{EPrefs}(S)$). The following corollary of Thm. 3.2 summarizes this observation.

Corollary 3.4. *Let S be a system, $S^\star$ its external trace system and $\widehat{S}$ a system with closed external behavior. If $S^\star \preccurlyeq_\alpha \widehat{S}$ and $\widehat{C}^\dagger \in \mathcal{W}^\dagger(\widehat{S}, \psi)$ then $C = \widehat{C}^\dagger \circ \alpha \in \mathcal{W}(S, \psi)$. Further, if S has closed external behavior and $S^\star \cong_\alpha \widehat{S}$ then $\mathcal{W}(S, \psi) = \emptyset$ iff $\mathcal{W}^\dagger(\widehat{S}, \psi) = \emptyset$.*

It should be noted that $S^\star$ is infinite state even when the system S is finite state. This should not worry us too much as S is typically also infinite state and we cannot efficiently check Def. 3.1 over S either. The contribution of Cor. 3.4 is therefore conceptual. It shows that the same notion of sound abstractions developed for ABCD with state-feedback control can be utilized for output-feedback when applied to the external trace semantics of S captured by $S^\star$. In addition, the next section shows a construction of a finite-state (and therefore closed) abstraction $\widehat{S}$ directly from S which can be proven to be a sound abstraction of $S^\star$ and thereby allows to apply Cor. 3.4 to obtain a sound ABCD framework for output-feedback control without explicitly computing $S^\star$.

4 COMPUTING ABSTRACTIONS

We now turn to the algorithmic problem of *computing* system abstractions such that designing a state-feedback controller on the abstraction allows us, through Cor. 3.4, to construct a corresponding output-feedback controller for the original system. For this we assume that the original system has an infinite state space—e.g., defined by a continuous-state dynamical system—and our goal is to compute a *finite-state abstraction* on which algorithmic techniques for state-based controller synthesis (e.g., [14, 27]) can be applied.

We first recall two well-known approaches to compute such *finite-state abstractions* which were developed for the setting where the original system has a *finite* state space, and show that they may not terminate for *infinite-state systems*, even if a finite-state realization of the topological closure of its external behavior exists. Based on this insight, we provide (Sec. 4.4) an algorithm for abstracting *infinite-state systems* which overcomes this problem.

Algorithm 1 KA: Knowledge-Based Abstraction

Require: $S = (X, X_0, U, F, Y, H)$
1: $\widehat{X}_0 \leftarrow \{X_0 \cap H^{-1}(y) \in 2^X \setminus \{\emptyset\} \mid y \in Y\}$
2: $\widehat{X}_{old} \leftarrow \emptyset$ and $\widehat{X} \leftarrow \widehat{X}_0$
3: **while** $\widehat{X}_{old} \neq \widehat{X}$ **do**
4: $\quad \widehat{X}_{old} \leftarrow \widehat{X}$
5: $\quad$ **for** $\widehat{x} \in \widehat{X}_{old}, u \in U, y \in Y$ **do**
6: $\quad\quad \widehat{x}' \leftarrow F(\widehat{x}, u) \cap H^{-1}(y)$
7: $\quad\quad \widehat{X} \leftarrow \widehat{X} \cup \{\widehat{x}'\}$ if $\widehat{x}' \neq \emptyset$
8: $\quad$ **end for**
9: **end while**
10: Define $\widehat{x}' \in \widehat{F}(\widehat{x}, u)$ iff there exist y s.t. $\widehat{x}' = F(\widehat{x}, u) \cap H^{-1}(y)$
11: Define $\widehat{H}(\widehat{x}) = y$ iff $y \in H(\widehat{x})$
12: **return** $\widehat{S}^{\mathsf{K}} = (\widehat{X}, \widehat{X}_0, U, Y, \widehat{F}, \widehat{H})$

4.1 Knowledge-Based Abstraction

A standard way to solve control-strategy synthesis problems over finite-state systems with partial observation [3, 19, 30] is to use a *knowledge-based* subset construction. Starting from the subsets of initial states generating the same output, the knowledge-based subset construction algorithm, given in Alg. 1, explores all inputs to the system and successively generates subsets of states that are indistinguishable given the full history of applied inputs and observed outputs. Such subsets $\widehat{x}$ of states of the original system S become the states of the knowledge-based abstraction $\widehat{S}^{\mathsf{K}} := \text{KA}(S)$. Note that every reachable state $\widehat{x}$ of $\widehat{S}^{\mathsf{K}}$ computed via Alg. 1 has the property that all $x \in \widehat{x}$ have the same output; thus, we can define $\widehat{H}(\widehat{x})$ as the (unique) output $H(x)$ of some $x \in \widehat{x}$.

Remark 2. *We restrict our attention to systems with strict transition function in this paper to simplify the discussion of the KA algorithm in Alg. 1 and KAM in Alg. 2. If not all inputs are enabled in every state, KA would need to distinguish state sets further based on the set of available inputs. This would require the controller to "observe" the status of currently enabled inputs. The not fully input-enabled case can be implicitly handled by introducing an* observable *"dummy" state and redirecting all transitions with disabled inputs to the dummy state. This indirectly observes the status of enabled inputs and provides a system with strict transition function. Then one can conjoin the specification with the constraint that the dummy state is never visited to obtain the original control problem. We postpone a more in-depth treatment of this implicit observation of enabled inputs to future work.*

The next proposition formalizes the intuition that $\widehat{S}^{\mathsf{K}}$ is a useful abstraction for a given output-feedback control problem over S. With Prop. 4.1 in place, it immediately follows from Cor. 3.4 that one can compute an output feedback controller $C := \widehat{C}^\dagger \circ \text{LastX}_{\widehat{S}^{\mathsf{K}}} \in \mathcal{W}(S, \psi)$ from an abstract state-feedback controller $\widehat{C}^\dagger \in \mathcal{W}^\dagger(\widehat{S}^{\mathsf{K}}, \psi)$, if it exists.

Proposition 4.1. *Let S be a system, $S^\star$ its external trace system, and $\widehat{S}^{\mathsf{K}} = \text{KA}(S)$. Then, $S^\star \cong_\alpha \widehat{S}^{\mathsf{K}}$ with $\alpha = \text{LastX}_{\widehat{S}^{\mathsf{K}}}$.*

Proof. To simplify notation we define $\widehat{S} := \widehat{S}^{\mathsf{K}}$.
▶ We first prove that $\text{LastX}_{\widehat{S}}(\text{EHist}_{\widehat{S}}(\widehat{x})) = \{\widehat{x}\}$ for all $\widehat{x} \in \widehat{X}$

by picking $\widehat{\pi} = \widehat{x}_0 u_0 \widehat{x}_1 u_1 \dots \widehat{x}_n$ and $\widehat{\pi}' = \widehat{x}'_0 u_0 \widehat{x}'_1 u_1 \dots \widehat{x}'_n$ s.t. $\widehat{H}(\widehat{x}_k) = \widehat{H}(\widehat{x}'_k)$ for all $k \in [0;n]$ and showing $\widehat{x}_n = \widehat{x}'_n$ by induction. ▷ For $k = 0$ we have $\widehat{x}_0, \widehat{x}'_0 \in \widehat{X}_0$. As $\widehat{H}(\widehat{x}_0) = \widehat{H}(\widehat{x}'_0)$, we have $\widehat{x}_0 = \widehat{x}'_0$. ▷ Now let $k \in [1;n]$ and assume $\widehat{x}_{k-1} = \widehat{x}'_{k-1}$. Then it follows that there exists y, y' s.t. $\widehat{x}_k = F(\widehat{x}_{k-1}, u_{k-1}) \cap H^{-1}(y)$ and $\widehat{x}'_k = F(\widehat{x}_{k-1}, u_{k-1}) \cap H^{-1}(y')$. Again, $\widehat{H}(\widehat{x}_k) = \widehat{H}(\widehat{x}'_k)$ implies $y = y'$. Then it is easy to see that $\widehat{x}_k = \widehat{x}'_k$.

▶ We now show that equality holds for (A1)-(A3) from Def. 3.1:
▷ (A1): By definition, $X_0^\star = H(X_0)$; and by line 1 in Alg. 1, we have $\mathrm{LastX}_{\widehat{S}}(H(X_0)) = \widehat{X}_0$. ▷ (A2): Let $\widehat{x} = \mathrm{LastX}_{\widehat{S}}(v)$ and $u \in U$. Further, let $\widehat{x}'_y = F(\widehat{x}, u) \cap H^{-1}(y)$ and define $Y' = \{y \in Y \mid \widehat{x}'_y \neq \emptyset\}$. Now recall that $F^\star(v,u) = \{vuy \mid F(\mathrm{LastX}_{\widehat{S}}(v), u) \cap H^{-1}(y) \neq \emptyset\}$. This implies $\widehat{x}'_y \in \mathrm{LastX}_{\widehat{S}}(F^\star(v,u))$ if $y \in Y'$. Further, as $\mathrm{LastX}_{\widehat{S}}(\mathrm{EHist}_{\widehat{S}}(\widehat{x})) = \{\widehat{x}\}$ we have $\mathrm{LastX}_{\widehat{S}}(F^\star(v,u)) = \bigcup_{y \in Y'} \{\widehat{x}'_y\}$. From the definition of $\widehat{F}$, it further follows that $\widehat{x}'_y \in \widehat{F}(\widehat{x}, u)$ if $y \in Y'$ and in particular $\widehat{F}(\widehat{x}, u) = \bigcup_{y \in Y'} \{\widehat{x}'_y\}$. Recalling that $\widehat{x} = \mathrm{LastX}_{\widehat{S}}(v)$ this shows that $\mathrm{LastX}_{\widehat{S}}(F^\star(v,u)) = \widehat{F}(\mathrm{LastX}_{\widehat{S}}(v), u)$. ▷ (A3): Observe that $\gamma = \mathrm{EHist}_{\widehat{S}}$ for $\alpha = \mathrm{LastX}_{\widehat{S}}$. Then $H(\gamma(\widehat{x})) = H(\mathrm{EHist}_{\widehat{S}}(\widehat{x})) = H(\{\widehat{x}\})$, hence $H(\{\widehat{x}\}) = \{\widehat{H}(\widehat{x})\}$. □

Alg. 1 incrementally constructs $\widehat{S}^{\mathsf{K}}$ from S by *forward exploration* from the initial states. As the abstract state space $\widehat{X} \subseteq 2^X$ contains subsets of X it terminates if X is finite. This case is the one most prominently discussed in existing literature, e.g., in [3, 30]. However, Alg. 1 might also terminate if $\widehat{X}$ is infinite (see, e.g., the example in Sec. 4.3), given that the necessary operations (in particular "Post" and "Intersect") can be implemented if state subsets are infinite. If X is infinite, Alg. 1 might however also not terminate even if there exists a finite-state realization of S. This is shown in Ex. 4.2. It is interesting to note that this might still be the case even if $X = X_0$. This can be verified by checking that Alg. 1 does also not terminate if all states in the system S depicted in Fig. 2 are initial.

Example 4.2. Consider the infinite state system S in Fig. 2, with $U = \{u\}, Y = \{A, B\}$. By omitting the trivial input, the external language $\mathrm{Ext}(S)$ of this system is $A(B)^+(A)^\omega \mid A(B)^\omega$, for which one can construct a finite trace equivalent system, for instance, using one of the methods discussed in the following sections. Yet, Alg. 1 will separate every state labeled with B, leading to an infinite chain of states with observation B, and will therefore not terminate.

4.2 Bisimulation Minimization

The knowledge-based abstraction algorithm KA computes reachable subsets going forward, but it may fail to terminate by trying to distinguish states that are language equivalent to already computed ones, that is, states that generate the same future sequence of outputs under the same input sequence. Thus, one could first compute a *bisimulation quotient* [2, 9, 16] of the system S and only then compute the knowledge-based abstraction. It is possible that an infinite-state system has a finite bisimulation quotient; in that case, constructing the quotient first will allow the knowledge-based abstraction to terminate (see Fig. 2 (bottom) for an example).

For a system $S = (X, X_0, U, F, Y, H)$, a partition of the set X is a set of non-empty sets of X, called blocks, that are pairwise disjoint

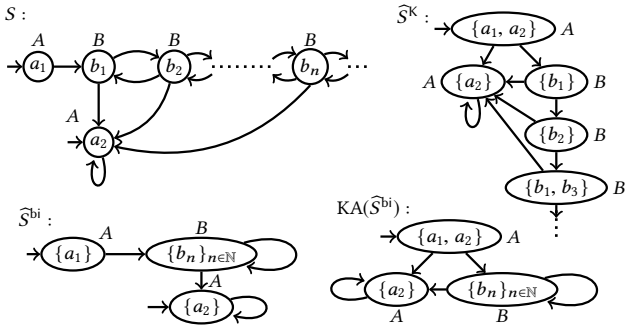

Figure 2: The system S (top left) has an infinite-state knowledge abstraction $\widehat{S}^{\mathsf{K}}$ (top right) while an exact finite-state representation of $\mathrm{Ext}(S)$ **exists, which is correctly computed by first computing the bisimilarity abstraction $\widehat{S}^{\mathsf{bi}}$ (bottom left, see Sec. 4.2) and then applying Alg. 1 (bottom right).**

and whose union is X. A partition is *stable* if the following properties hold. First, for each block $\widehat{x}$ of the partition, every state in the block has the same output: for all $x, x' \in \widehat{x}$, we have $H(x) = H(x')$. Second, for each pair of blocks $\widehat{x}, \widehat{x}'$ with $y' = H(x)$ for all $x \in \widehat{x}'$ and for each input $u \in U$ we have either $F(\widehat{x}, u) \cap H^{-1}(y') \subseteq \widehat{x}'$ or $F(\widehat{x}, u) \cap \widehat{x}' = \emptyset$. Using the notion of a stable partition of X we can define the *bisimulation abstraction* $\widehat{S}^{\mathsf{bi}} = (\widehat{X}, \widehat{X}_0, U, \widehat{F}, Y, \widehat{H})$ of S as follows. The set of abstract states $\widehat{X}$ is the minimal stable partition of X. The initial abstract states $\widehat{X}_0$ are those blocks that contain some initial states from X_0. The abstract transition function is defined as $\widehat{F}(\widehat{x}, u) = \{\widehat{x}' \in \widehat{X} \mid \exists x \in \widehat{x}. F(x, u) \subseteq \widehat{x}'\}$. Moreover, since every state in each block of the partition has the same output, we can uniquely define $\widehat{H}(\widehat{x})$ to be the output of some state in $\widehat{x}$.

A partition refinement algorithm [9, 18] can be used to compute $\widehat{S}^{\mathsf{bi}}$ from S. Unlike Alg. 1, this algorithm proceeds *backwards* by splitting blocks based on their predecessors, starting with the partition defined by the outputs, i.e., $\{q \in 2^X \setminus \{\emptyset\} \mid \exists y \in Y \,.\, q = H^{-1}(y)\}$. This algorithm may terminate if X is infinite and the necessary operations are implementable over infinite state subsets. Going back to the system described in Ex. 4.2 we see that the bisimulation quotient $\widehat{S}^{\mathsf{bi}}$ (depicted in Fig. 2 (bottom left)) is finite, while the original system S (depicted in Fig. 2 (top left)) and its knowledge-based abstraction $\widehat{S}^{\mathsf{K}}$ (depicted in Fig. 2 (top right)), are infinite. Applying the KA algorithm on $\widehat{S}^{\mathsf{bi}}$ returns the desired finite state abstraction (depicted in Fig. 2 (bottom right)) which allows for output feedback control. However, if S is infinite-state, the partition refinement algorithm is not guaranteed to terminate even if the knowledge-based abstraction of the original system is finite. This is further illustrated by the example discussed in the next section, which shows that knowledge-based abstraction and bisimulation minimization are incomparable and the suggested procedure to compute $\widehat{S}^{\mathsf{bi}}$ first, before utilizing *KA*, may not terminate.

4.3 Illustrative Example

Before explaining KAM, we introduce an illustrative example. Consider the infinite state system S depicted in Fig. 3 (top left) with $U = \{u\}$ and $Y = \{A, B, C, D, E, F\}$. It consists of one initial state a_1 which outputs A, an infinite chain of states $b_i, i \in \mathbb{N}$, all of which output B, and four different modules Λ_D^I (light blue, dashed), Λ_D^{II}

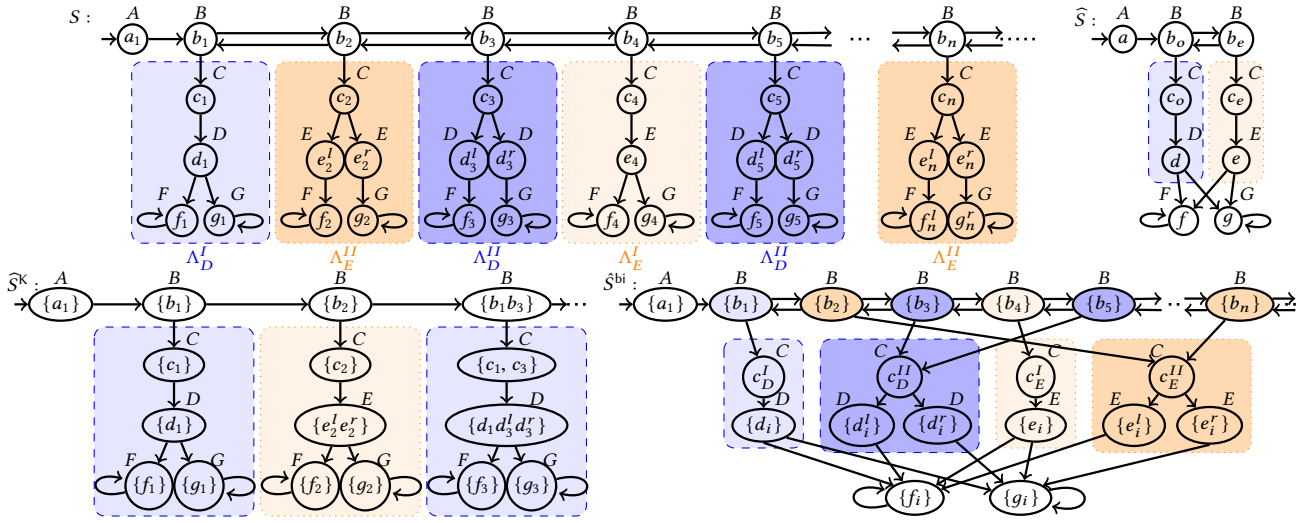

Figure 3: Infinite-state system S (top left) discussed in Sec. 4.3, its sound finite-state abstraction $\widehat{S}$ (top right), part of its infinite-state knowledge abstraction $\widehat{S}^{\mathsf{K}}$ (bottom left) and its infinite bisimulation quotient $\widehat{S}^{\mathsf{bi}}$ (bottom right). The single input $U = \{u\}$ is omitted and outputs $Y = \{A, \ldots, F\}$ are indicated next to the respective state. A state subset $\{\alpha_i\}$ denotes the set $\{\alpha_i\}_{i\in\mathbb{N}}$.

(dark blue, dashed), Λ^I_E (light orange, dotted) and Λ^{II}_E (dark orange, dotted), attached to one b-state each. System S is constructed s.t. modules of type D (resp. of type E) are reachable after output B has occurred an *odd* (resp. *even*) number of times, i.e., from all states $X^{odd}_B := \{b_{2i+1}\}_{i\in\mathbb{N}}$ (resp. from all states $X^{even}_B := \{b_{2i}\}_{i\in\mathbb{N}}$). However, the sequence of class I and II modules of the same type $i \in \{E, D\}$ is irregular, i.e., there is no ω-regular expression to describe how Λ^I_i and Λ^{II}_i modules repeat.

By closely investigating the modules of the same i-type it can be observed that modules Λ^I_i and Λ^{II}_i for the same $i \in \{D, E\}$ are external language equivalent. Therefore, the regularity of alternating between type D and type E modules is enough to obtain a sound finite-state realization $\widehat{S}$ of S depicted in Fig. 3 (top right).

KA-algorithm (Sec. 4.1). The KA algorithm computes the abstract state space by combining all states with the same observable *past* while going *forward*. For the system S in Fig. 3 (top left) it constructs state subsets as depicted in Fig. 3 (bottom left). We see that the KA algorithm discovers that class I modules are a sound realization of class II modules, i.e., $\widehat{S}^{\mathsf{K}}$ only consists of class I modules s.t. type D and type E modules are reachable from states in X^{odd}_B and X^{even}_B respectively. However, the KA algorithm still does not terminate on this example as it explores language equivalent states unnecessarily. I.e., by computing state subsets only going forward, it computes a new, not yet explored subset of b-states in every iteration. The KA-algorithm is not able to *generalize* and thereby merge all states corresponding to X^{odd}_B or X^{even}_B due to their unique future.

Bisimulation-Quotient (Sec. 4.2). A partition refinement algorithm computing the bisimulation quotient of S merges states with the same observable *future* going *backward*. For the system S in Fig. 3 (top left) it immediately discovers that all states in $X_F := \{f_i\}_{i\in\mathbb{N}}$ as well as $X_G := \{g_i\}_{i\in\mathbb{N}}$ have the same observable future (namely F^ω and G^ω, respectively). It further merges all states contained in the same Λ^j_j module into one equivalence class (see Fig. 3 (bottom right) indicated by the four color/line patterns). However, as it proceeds backwards, it does not take into account the reachable portion of all state subsets and thereby considers states within class I and II modules of the same type as different. This differentiates b states depending on the class of modules they are connected to (indicated by the coloring of the b-states in Fig. 3 (bottom right)). As the partition refinement algorithm constructs equivalence classes going backward, it generates a distinct equivalence class for the left and right "color pattern" a b state "sees". As we assume that class I and II modules are irregularly sequenced, there exist infinitely many such equivalence classes and the algorithm therefore never terminates.

Combining both algorithms. For this example, running the KA algorithm first and the partition refinement algorithm second, results in the finite state abstraction $\widehat{S}$ depicted in Fig. 3 (top right). This is, however, not practically implementable, as the KA algorithm never terminates. Further, we have shown that for Ex. 4.2 one needs to execute the partition refinement algorithm first, followed by the KA algorithm. One can therefore construct an example where one reachable part of the state space requires executing the KA algorithm first, while the other part requires the partition refinement algorithm to be executed first. In this case, no order would lead to the desired result.

4.4 Knowledge Abstraction with Minimization

We now present the *Knowledge-based Abstraction algorithm with Minimization* (KAM), given in Alg. 2, which interlaces the forward Knowledge-based Abstraction (KA) with backward refinement-based Minimization (M). We also illustrate the algorithm using the example from Sec. 4.3.

Algorithm Description. KAM generates a rooted, labeled tree and a *cover set* $\mathtt{Cover} \subseteq 2^X$. The nodes of the tree are kept in $\mathtt{EXP}_X$ and the edges in $\mathtt{EXP}_F$. The edges are labeled with inputs from

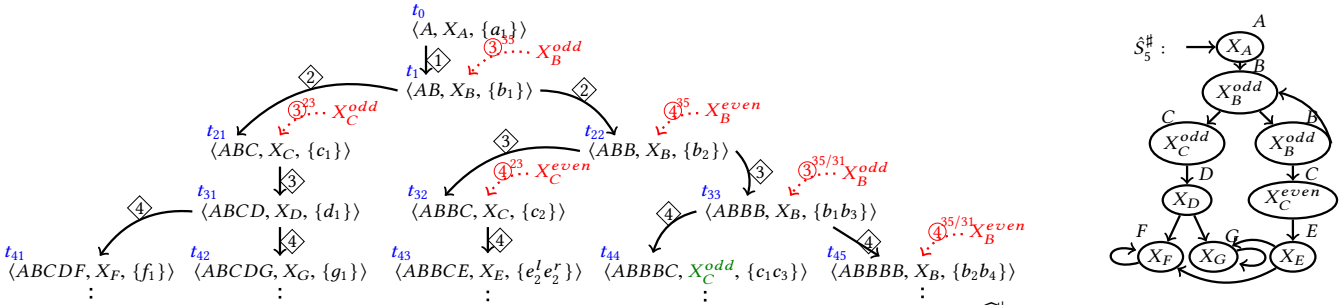

Figure 4: Exploration tree $\mathtt{EXP}_\mathsf{F}$ of S in Fig. 3 computed by Alg. 2 (left) and the abstract system $\widehat{S}^\sharp$ extracted after its 5th iteration (right). Nodes are labeled by t_k (blue) for easier reference and the single input u is omitted to avoid clutter. Diamond-enclosed numbers indicate the iteration in which this transition is explored. Dotted red arcs indicate cover block refinements in the iteration of the main while loop depicted by the red circled number and caused by the line of Refine indicated on its top right. E.g., X_B of t_1 is refined by re-calling Refine in line 35 after X_C of t_{21} was refined in line 23 (as t_1 is a predecessor of t_{21}). The notation 35/31 in t_{45} indicates that its cover block X_B is refined by line 31 after re-calling Refine via line 35 on node t_{22}.

U. The nodes are labeled with a three-tuple $\langle v, q, c\rangle \in \mathtt{EXP}_\mathsf{X}$, consisting of a sequence v of external events seen when reaching the current node from the root of the tree, a *block* $q \subseteq X$ in the current Cover, and a subset of states $c \subseteq X$ (called a *cell*). Intuitively, a tuple $\langle v, q, c\rangle \in \mathtt{EXP}_\mathsf{X}$ remembers the observed input/output sequence from the initial states (in v), the available knowledge about the current state (in c), and the current "guesses" on states which are future observation-equivalent to c (in q). The *cells* c and *blocks* q correspond to the data structures manipulated by the *KA* and the Minimization algorithm, respectively, and are initialized similarly: Cover is initialized with the partition induced by H on X (line 1, see Sec. 4.2), cells are initialized with all initial cover blocks containing an initial state (line 3). Note that the initialization of cells simplifies as we have assumed that H respects the initial state set X_0.

Example 4.3. For the example in Sec. 4.3, we see that the partition induced by H on X results in the initial cover set $\mathtt{Cover} = \{X_y \mid y \in Y\}$ s.t. X_y collects all states of S that generate the output y, e.g., $X_A := \{a_1\}$ and $X_C := \{c_i\}_{i\in\mathbb{N}}$. On the other hand, there is only one initial cell, namely $\{a_1\}$ with $H(\{a_1\}) = A$. This results in the initialization of $\mathtt{EXP}_\mathsf{X}$ with the tuple $\langle A, X_A, \{a_1\}\rangle$ as depicted in Fig. 4 (left).

The main loop of KAM (lines 5–21) grows the tree by iterating between a forward exploration (as in KA) and backward refinement (as in bisimulation). The forward exploration picks the current leaves (v, q, c) of the tree (line 7) and executes one step of KA to generate new *cells* c' for every $u \in U$ and $y \in Y$ (compare Alg. 1, line 6 and Alg. 2, line 10).

For each minimal block q' in the current Cover set that contains c', KAM adds a new node $\langle v', q', c'\rangle$ to the tree (line 12), where v' extends the parents event sequence with the latest input and the last output. The edge from the parent to the new node is labeled with the input and stored in $\mathtt{EXP}_\mathsf{F}$ (line 13).

Example 4.4. The resulting exploration tree for the example in Sec. 4.3 is depicted in Fig. 4 (left). Here, the diamond-enclosed number on the edges indicates the iteration of the while loop (in line 5-21 of Alg. 2) in which this transition and its child are added to the tree. When comparing Fig. 4 (left) and the KA-abstraction $\widehat{S}^\mathsf{K}$ of this example (Fig. 3 (bottom left)), we see that the third component of all tuples generated by KAM coincides with the abstract states generated by KA in the same iteration (i.e., in a state with the same distance from the initial state).

Having thus created all the children for a node $\langle v, q, c\rangle$, if c is a proper subset of q, the next step in KAM is to check if q, the current guess for the observation equivalence class for c, needs to be refined. Refinement is performed by the function Refine (Alg. 2, line 15) and works similarly to the bisimulation algorithm.

In contrast to the usual bisimulation algorithm, $\textsc{Refine}(\langle\cdot, q, c\rangle)$ only splits a block q based on its possible successors in the tree if this split respects c, thereby avoiding the splitting of indistinguishable states, which caused the non-termination issue discussed in Sec. 4.2. One can intuitively think of $s \subseteq X$ computed in line 27 of Alg. 2 as the set of all states which are equivalent to c in terms of their one-step observable future. However, in contrast to the bisimulation algorithm, KAM only adds s to Cover but does not add its complement $q \setminus s$ (see line 29). This is due to the fact that this operation might not respect the currently available cells and again split indistinguishable states. If $q \setminus s$ is indeed needed, it will be discovered by another call to Refine.

Summarizing the above description, we see that Refine refines the Cover set based on the one-step future of the computed cell. Given this refinement, all previously obtained relations between cells and blocks need to be re-evaluated as $s \subset q$ implies that s is now the minimal cover of c, if c was previously related to q in $\mathtt{EXP}_\mathsf{X}$ (see line 31). Thus, KAM updates its guess on the set of states possibly external language equivalent to a state in c. This, however, might imply new block splits in cell/block pairs reaching c, which have been checked for refinement in previous iterations of the algorithm. This is taken care of by the recursive call to Refine in line 35. Note that the recursion always moves up to the parent in the tree, and thus it eventually terminates. One can show that after the recursive call to Refine terminates, we always have a single minimal cover box q for every cell c computed so far. That is, given the relation $\widetilde{\alpha}(c) = \{q \in \mathtt{Cover} \mid \langle c, q\rangle \in \mathtt{EXP}_\mathsf{X}^{\downarrow}\}$ for $\mathtt{EXP}_\mathsf{X}^{\downarrow} := \{\langle q, c\rangle \mid \exists v \,.\, \langle v, q, c\rangle \in \mathtt{EXP}_\mathsf{X}\}$, we have $|\widetilde{\alpha}(c)| = 1$ (see Lem. A.2 in the appendix for a formal proof).

Algorithm 2 KAM: Knowledge Abstraction and Minimization

Require: $S = (X, X_0, U, F, Y, H)$

1: $\mathtt{Cover} \leftarrow \{q \in 2^X \setminus \{\emptyset\} \mid \exists y \in Y \,.\, q = H^{-1}(y)\}$;
2: $\mathtt{EXP}_\Gamma \leftarrow \emptyset$;
3: $\mathtt{EXP}_\mathsf{X} \leftarrow \{\langle H(c), q, c\rangle \mid q \in \mathtt{Cover} \wedge c = q \wedge c \cap X_0 \neq \emptyset\}$;
4: $\mathtt{EXP}_\mathsf{F} \leftarrow \emptyset$;
5: **while** $\mathtt{EXP}_\Gamma \neq \{\langle q, c\rangle \mid \exists \nu \,.\, \langle \nu, q, c\rangle \in \mathtt{EXP}_\mathsf{X}\}$ **do**
6: $\quad \mathtt{EXP}_\Gamma \leftarrow \{\langle q, c\rangle \mid \exists \nu \,.\, \langle \nu, q, c\rangle \in \mathtt{EXP}_\mathsf{X}\}$;
7: $\quad$ **for** $\langle \nu, q, c\rangle \in \mathtt{EXP}_\mathsf{X}$ s.t. $|\nu|$ is maximal **do**
8: $\quad\quad$ **for** $u \in U, y \in Y$ **do**
9: $\quad\quad\quad \nu' = \nu u y$;
10: $\quad\quad\quad c' = F(c, u) \cap H^{-1}(y) \neq \emptyset$;
11: $\quad\quad\quad Q' = \{q' \in \mathtt{Cover} \mid c' \subseteq q' \text{ and } q' \text{ is minimal}\}$;
12: $\quad\quad\quad \mathtt{EXP}_\mathsf{X} \leftarrow \mathtt{EXP}_\mathsf{X} \cup \{\langle \nu', q', c'\rangle \mid q' \in Q'\}$;
13: $\quad\quad\quad \mathtt{EXP}_\mathsf{F} \leftarrow \mathtt{EXP}_\mathsf{F} \cup \{(\langle \nu, q, c\rangle, u, \langle \nu', q', c'\rangle) \mid q' \in Q'\}$;
14: $\quad\quad$ **end for**
15: $\quad\quad$ **if** $c \subset q$ **then** Refine($\langle \nu, q, c\rangle$);
16: $\quad\quad$ **end if**
17: $\quad$ **end for**
18: $\quad \widehat{S} \leftarrow$ Extract($\mathtt{EXP}_\mathsf{X}, \mathtt{EXP}_\mathsf{F}$);
19: $\quad$ **if** TermCond() == true **then return** $\widehat{S}$;
20: $\quad$ **end if**
21: **end while**
22: **return** $\widehat{S}$;
23: **function** Refine($\langle \nu, q, c\rangle$)
24: $\quad$ **for** $u \in U$ **do**
25: $\quad\quad \mathtt{PostQ}_u \leftarrow \bigcup\{q' \in \mathtt{Cover} \mid (\langle \nu, q, c\rangle, u, \langle \cdot, q', \cdot\rangle) \in \mathtt{EXP}_\mathsf{F}\}$;
26: $\quad$ **end for**
27: $\quad s \leftarrow \{x \in q \mid \forall u \in U \,.\, F(x, u) \subseteq \mathtt{PostQ}_u\}$;
28: $\quad$ **if** $s \subset q$ **then**
29: $\quad\quad \mathtt{Cover} \leftarrow \mathtt{Cover} \cup \{s\}$;
30: $\quad\quad$ **for** all $\langle \tilde{\nu}, \tilde{q}, \tilde{c}\rangle \in \mathtt{EXP}_\mathsf{X}$ s.t. $\tilde{q} = q$ **do**
31: $\quad\quad\quad$ **if** $\tilde{c} \subset s$ **then** change $\langle \tilde{\nu}, q, \tilde{c}\rangle$ to $\langle \tilde{\nu}, s, \tilde{c}\rangle$ in $\mathrm{EXP}_{\Gamma, X, F}$;
32: $\quad\quad\quad$ **end if**
33: $\quad\quad$ **end for**
34: $\quad\quad$ **for** all $(\langle \tilde{\nu}', \tilde{q}', \tilde{c}'\rangle, \cdot, \langle \tilde{\nu}, \tilde{q}, \tilde{c}\rangle) \in \mathtt{EXP}_\mathsf{F}$ s.t. $\tilde{q} = s \wedge \tilde{c}' \subset \tilde{q}'$ **do**
35: $\quad\quad\quad$ Refine($\langle \tilde{\nu}', \tilde{q}', \tilde{c}'\rangle$);
36: $\quad\quad$ **end for**
37: $\quad$ **end if**
38: **end function**
39: **function** Extract($\mathtt{EXP}_\mathsf{X}, \mathtt{EXP}_\mathsf{F}$)
40: $\quad \widehat{X} \leftarrow \{q \in 2^X \mid \langle \cdot, q, \cdot\rangle \in \mathtt{EXP}_\mathsf{X}\}$;
41: $\quad \widehat{X}_0 \leftarrow \{X_0 \cap H^{-1}(y) \in 2^X \setminus \{\emptyset\} \mid y \in Y\}$;
42: $\quad \widehat{F} \leftarrow \{(q, u, q') \mid \langle \cdot, q, \cdot\rangle, u, \langle \cdot, q', \cdot\rangle) \in \mathtt{EXP}_\mathsf{F}\}$;
43: $\quad \widehat{H}(\widehat{x}) = y$ if $y \in H(\widehat{x})$;
44: $\quad$ **return** $\widehat{S} = (\widehat{X}, \widehat{X}_0, U, \widehat{F}, Y, \widehat{H})$;
45: **end function**

Example 4.5. For the example in Sec. 4.3, we see that for the tuple t_0 we have $c = q$ as $X_A = \{a_1\}$, hence, Refine is not called in the first iteration of KAM. In its second iteration, it computes the leaves t_{21} and t_{22} in the main while loop and then checks the *parent node* t_1 for refinement. For this, it computes all cover cells reachable by b_1 (which is PostQ $= \bigcup\{X_B, X_C\}$ and then computes all states in $q = X_B$ with the same reachable cover blocks, which is $s = X_B$. As $q = s$, no split occurs and a new iteration of the main while loop starts. After the computation of the leaves $t_{31} - t_{33}$ KAM checks the parent node t_{21} for refinement. Here we obtain PostQ $= X_D$ and $s = X_C^{odd} = \{c_{2i+1}\}_{i \in \mathbb{N}}$. As $s \subset q = X_C$ the cell X_C^{odd} is added to Cover. As there is no other node in the tree with a cell component contained in X_C^{odd}, we only update the block component of t_{21} (indicated by the red dotted arrow pointing to it in Fig. 4) and schedule all its predecessors for refinement. Therefore, node t_1 is checked for refinement again. Given the new cover cell X_C^{odd} we now obtain PostQ $= \bigcup\{X_B, X_C^{odd}\}$ and $s = X_B^{odd}$. This updates the cover element of t_1 and t_{33}. This schedules only t_{22} for refinement, as t_0 does not fulfill the condition that $c \subset q$. Now it can be observed that checking t_{22} for refinement still gives PostQ $= \bigcup\{X_B, X_C\}$ as we have not yet added the cover element $X_B^{even} = X_B \setminus X_B^{odd}$. This is due to the fact that we do not know whether this element is indeed needed and respects the constructed state subsets. We therefore leave node t_{22} unchanged and proceed to the forth iteration of the main while loop. This computes the leaves $t_{41} - t_{45}$. It should be noted that during this computation we now have the new cover cell X_C^{odd} available and KAM uses this smaller cover cell to correctly tack the equivalence class for t_{44} (indicated in green in Fig. 4). Now the only interesting refinement check is on t_{32} which discovers the new cover element X_C^{even} and induces the further refinement of node t_{22} introducing the cover cell X_B^{even}. This updates t_{22} and t_{45}. Due to space constraints, we do not depict the constructed tree further. It should however be noted that t_{43} clusters e_2^l and e_2^r into a single cell, as these states are not distinguishable based on the past observations. Therefore, calling Refine on t_{43} in the next iteration of KAM will not refine the equivalence class X_E as PostQ $= \bigcup\{X_F, X_G\}$ and we therefore obtain $s = X_E$. The same happens for nodes d_i^l and d_j^r. This prevents the non-termination issue of the bisimulation algorithm for this example.

After exploration and refinement, KAM extracts an abstraction $\widehat{S}$ via the function Extract in line 19. Intuitively, Extract projects the tree in $\mathtt{EXP}_\mathsf{F}$ to the blocks in the current Cover set which are reachable. It thereby "forgets" the forward-computed cells and only retains their observation-equivalent generalizations s. For the example in Sec. 4.3 the abstraction extracted after the fifth iteration of KAM is depicted in Fig. 4 (right). It can be observed that Fig. 4 (right) coincides with the abstraction $\widehat{S}$ in Fig. 3 (top right) up to a renaming of states.

Termination. Intuitively, KAM should terminate if Cover stabilizes. Then, all distinguishable subsets which are observation-equivalent have been discovered, and hence, imply $\mathrm{Ext}(S) = \mathrm{Ext}(\widehat{S})$. That is, we would ideally like to have TermCond() == true in line 19 iff Cover has stabilized. Unfortunately, even if we observe that Cover has not changed in the current iteration, we do not know if it will never change again. This is because KAM bases its search for cover splits on the already constructed state-subsets. There might be a very long input/output event sequence which only causes a subset split after a long exploration phase. As the

state space of S is infinite, we cannot check if this will ever happen. Interestingly, this is also true for fully initialized systems (i.e., where $X = X_0$). Thus, this termination check is undecidable.

One interesting special case where termination is decidable occurs if the KA algorithm (Alg. 1) terminates (which is for example always the case if X is finite). In this case, one can show that $\mathtt{EXP}_\Gamma = \mathtt{EXP}_\chi{}^\downarrow$ holds in the l-th iteration of Alg. 2 iff $\Gamma = \widehat{X}$ holds in the l-th iteration of Alg. 1 (see Lem. A.3 in the appendix for a formal proof of this statement). While Cover might have stabilized earlier, we know it has surely stabilized by then.

Finite-State Abstractions. The termination condition discussed above aims on computing a sound *finite-state realization* of the external behavioral closure of S which might not exist. Indeed, for arbitrary non-linear dynamical systems there rarely ever exists an exact finite-state realization in this sense, even if their input and output sets are finite. Therefore, as the name suggests, abstraction-based controller synthesis is usually only aiming at computing a finite-state *abstraction* which is accurate enough to synthesize an abstract controller for the given specification.

In this context, it is interesting to investigate whether the system $\widehat{S}^\#$ computed in line 18 of Alg. 2 after running the while loop in line 5-21 finitely often, is indeed a sound abstraction of S in the sense of Def. 3.1 and therefore allows for abstraction based control in the sense of Cor. 3.4. Interestingly, this is only true if KAM has already explored all possible output events which are reachable in S at least once when terminated. This is for example trivially satisfied if $X_0 = X$. Additionally, whenever Cover stabilizes after a finite number of iterations, KAM indeed computes a sound realization of S. This is formalized in the following theorem.

Theorem 4.6. *Let S be a system, $S^\star$ its external trace system and $\widehat{S}^\#$ an abstract system extracted in line 18 of* KAM(S) *in some iteration. Further, let $Y^\# = \{y \in Y \mid \exists \langle q, c\rangle \in \mathtt{EXP}_\Gamma \,.\, \widehat{H}(q) = y\}$ and $\mathrm{Reach}(Y) = \{y \in Y \mid \exists \rho \in \mathrm{EPrefs}(S) \,.\, y = \mathrm{Last}(\rho)\}$. If $Y^\# = \mathrm{Reach}(Y)$ it holds that $S^\star \preccurlyeq_\alpha \widehat{S}^\#$ with $\alpha = \mathrm{LastX}_{\widehat{S}^\#}$. Further, if* Cover *has stabilized, we additionally have $S^\star \cong_\alpha \widehat{S}^\#$.*

In order to prove Thm. 4.6, we first prove Prop. 4.7 below which formalizes the intuition that, under the given premises, the cell/block pairs $\langle q, c\rangle \in \mathtt{EXP}_\chi{}^\downarrow$ available when extracting $\widehat{S}^\#$ in line 18 of Alg. 2 actually induce a sound abstraction relation between $\widehat{S}^\mathsf{K}$ and $\widehat{S}^\#$. I.e., we always have $\widehat{S}^\mathsf{K} \preccurlyeq_{\widetilde{\alpha}} \widehat{S}^\#$ for

$$\widetilde{\alpha}(c) := \{q \in \widehat{X}^\# \mid \langle q, c\rangle \in \mathtt{EXP}_\chi{}^\downarrow\}. \tag{1}$$

Further, Prop. 4.7 shows that $\widehat{S}^\mathsf{K} \cong_{\widetilde{\alpha}} \widehat{S}^\#$ if $\widehat{S}^\mathsf{K}$ is finite-state (and thereby Cover has stabilized from Lem. A.3 in the appendix). With this result Thm. 4.6 becomes a simple corollary of Prop. 4.7 and Prop. 4.1 by utilizing the compositionality of sound abstractions (see Prop. A.1 in the appendix for a formal proof).

Proposition 4.7. *Given the premises of Thm. 4.6, it holds that $\widehat{S}^\mathsf{K} \preccurlyeq_{\widetilde{\alpha}} \widehat{S}^\#$ with $\widetilde{\alpha}$ as in* (1)*. Further, if* Cover *has stabilized, we additionally have $\widehat{S}^\mathsf{K} \cong_{\widetilde{\alpha}} \widehat{S}^\#$.*

Proof. To simplify notation we use $\widetilde{S} := \widehat{S}^\mathsf{K}$ and $\widehat{S} := \widehat{S}^\#$.

▶ We first show that equality holds for (A1) and (A3) from Def. 3.1.

▹ (A1): Observe that line 1 in Alg. 1 and line 41 in Alg. 2 literally match. Further, for all $\widehat{x} \in \widehat{X}_0$ we have that $\langle \varepsilon, \widehat{x}, \widehat{x}\rangle$ is in the initial cover set (line 1 in Alg. 2) and thereby $\langle \widehat{x}, \widehat{x}\rangle \in \mathtt{EXP}_\chi{}^\downarrow$, as we have assumed X_0 to respect H. As Alg. 2 always maintains $\widetilde{x} \subseteq \widehat{x}$ for any $\langle \widehat{x}, \widetilde{x}\rangle \in \mathtt{EXP}_\chi$ and all elements in Cover only get refined, we see that there is no other $\widehat{x}' \in \widehat{X}$ related to $\widetilde{x} \in \widetilde{X}$. We therefore have $\widetilde{\alpha}(\widetilde{X}_0) = \widehat{X}_0$. ▹ (A3): It is easy to see that for all $\widehat{x} \in \widehat{X}$ holds that $x, x' \in \widehat{x}$ implies $H(x) = H(x') = \widehat{H}(\widehat{x})$. As $\widetilde{x} \subseteq \widehat{x}$ for all $\langle \widehat{x}, \widetilde{x}\rangle \in \mathtt{EXP}_\chi$, we have $\widetilde{H}(\widetilde{x}) = \widehat{H}(\widehat{x})$ for all related states.

▶ Now we show that (A2) holds with equality for all $\langle \widetilde{x}, \widehat{x}\rangle \in \mathtt{EXP}_\Gamma$ (possibly a subset of $\mathtt{EXP}_\chi{}^\downarrow$). For this, observe that $\widehat{S}$ is extracted in the last iteration of the while loop in line 5-21 of Alg. 2 and therefore the recursive function Refine was applied to all $\langle \widehat{x}, \widetilde{x}\rangle \in \mathtt{EXP}_\Gamma$ with $\widetilde{x} \subset \widehat{x}$ and has terminated. We can therefore utilize Lem. A.2 in the appendix implying $|\widetilde{\alpha}(\widetilde{x})| = 1$ for all $\widetilde{x}$ present in $\mathtt{EXP}_\Gamma$.

▹ (A2) for $\mathtt{EXP}_\Gamma$: Pick $\widetilde{x} \in \widetilde{X}$, $u \in U$ and $\widetilde{x}'_y = F(\widetilde{x}, u) \cap H^{-1}(y)$. Further, define $Y' = \{y \in Y \mid \widetilde{x}'_y \neq \emptyset\}$ and let Q' contain all $\widehat{x}' \in \widehat{X}$ s.t. $\langle \widehat{x}', \widetilde{x}'_y\rangle \in \mathtt{EXP}_\chi$ and $y \in Y'$. Using the same argument as in the proof of Prop. 4.1 we have $\widetilde{F}(\widetilde{x}, u) = \bigcup_{y \in Y'} \{\widetilde{x}'_y\}$, and therefore, by definition, $\widetilde{\alpha}(\widetilde{F}(\widetilde{x}, u)) = Q'$. Now one can verify, by looking at line 10, 13 and 25 of Alg. 2, that $Q' = \mathtt{PostQ}_u(\langle \widehat{x}, \widetilde{x}\rangle)$ for $\{\widehat{x}\} = \widetilde{\alpha}(\widetilde{x})$. Further, we extract $\widehat{S}$ after all covers have been refined. With this we know that $F(\widehat{x}, u) = \mathtt{PostQ}_u(\langle \widehat{x}, \widetilde{x}\rangle)$, as otherwise there would exists a refinement $s \subset \widehat{x}$ in the sense of line 27 in Alg. 2. This further implies that for all $\langle \widehat{x}, \widetilde{x}_1\rangle, \langle \widehat{x}, \widetilde{x}_2\rangle \in \mathtt{EXP}_\Gamma$ we have that $\mathtt{PostQ}_u(\langle \widehat{x}, \widetilde{x}_1\rangle) = \mathtt{PostQ}_u(\langle \widehat{x}, \widetilde{x}_2\rangle)$. With this it follows that $Q' = \widehat{F}(\alpha(\widetilde{x}), u)$. This implies $\widetilde{\alpha}(\widetilde{F}(\widetilde{x}, u)) = \widehat{F}(\widetilde{\alpha}(\widetilde{x}), u)$.

▶ It remains to show that (A2) holds (with equality for a stable cover and with inclusion for an unstable one) for tuples $\langle \widehat{x}, \widetilde{x}\rangle \in \mathtt{EXP}_\chi{}^\downarrow \setminus \mathtt{EXP}_\Gamma$. First, one can verify that $\langle \widehat{x}, \widetilde{x}\rangle \in \mathtt{EXP}_\chi{}^\downarrow \setminus \mathtt{EXP}_\Gamma$ if (a) a tuple $\langle \sigma, \widehat{x}, \widetilde{x}\rangle$ is added to $\mathtt{EXP}_\chi$ in the last iteration of the while loop before extracting $\widehat{S}^\#$, and (b) if there exists no tuple $\langle \widehat{x}', \widetilde{x}\rangle \in \mathtt{EXP}_\Gamma$ for an arbitrary $\widehat{x}'$. While (a) is obvious, we show that (b) also holds. It follows from Lem. A.2, that after completing every iteration of the while-loop in line 21 it holds for every $\widetilde{x}$ already constructed, that there exists a unique $\widehat{x}'$ s.t. $\langle \widehat{x}', \widetilde{x}\rangle \in \mathtt{EXP}_\Gamma$. Now assume that $\langle \sigma, \widehat{x}, \widetilde{x}\rangle$ is added to $\mathtt{EXP}_\chi$ via line 11 of Alg. 2. Then we know that $\widehat{x}' = \widehat{x}$, as $\widehat{x}'$ is the unique minimal element of Cover covering $\widetilde{x}$ and, hence, $\langle \widehat{x}, \widetilde{x}\rangle \notin \mathtt{EXP}_\chi{}^\downarrow \setminus \mathtt{EXP}_\Gamma$.

▹ (A2) for $\mathtt{EXP}_\chi{}^\downarrow \setminus \mathtt{EXP}_\Gamma$ with stabilized Cover: If Cover has stabilized no element in Cover will be further refined by Refine. In particular, this implies that $\widehat{x}$ is stable for any $\langle \widehat{x}, \widetilde{x}\rangle \in \mathtt{EXP}_\chi{}^\downarrow \setminus \mathtt{EXP}_\Gamma$. Further, a stable cover implies that there already exists another tuple $\langle \widehat{x}, \widetilde{x}'\rangle \in \mathtt{EXP}_\Gamma$ for which all outgoing transitions are contained in $\mathtt{EXP}_\mathsf{F}$. With this, we use the same reasoning as for $\mathtt{EXP}_\Gamma$ to construct Q' and to show that (A2) holds with equality.

▹ (A2) for $\mathtt{EXP}_\chi{}^\downarrow \setminus \mathtt{EXP}_\Gamma$ with unstable Cover: If the Cover is not stable, we cannot ensure that $\widehat{x}$ is stable for any $\langle \widehat{x}, \widetilde{x}\rangle \in \mathtt{EXP}_\chi{}^\downarrow \setminus \mathtt{EXP}_\Gamma$, i.e., would not be refined in the next iteration of the while loop. Further, we have to make sure that there exists another tuple $\langle \widehat{x}, \widetilde{x}'\rangle \in \mathtt{EXP}_\Gamma$. Now recall that we initialize Cover with the largest subsets $\widehat{x}_0^y \subseteq X$ that generate the same output y. As $Y^\# = \mathrm{Reach}(Y)$, we know that all initial cover cells $\widehat{x}_0^y$ with $y \in \mathrm{Reach}(Y)$ will be explored (and possibly refined) at least once in Alg. 2. As $\widehat{x} \in$ Cover and by construction $\widehat{x} \subseteq \widehat{x}_0^y$ for $y = H(\widehat{x}) \in \mathrm{Reach}(Y)$ we know that $\langle \widehat{x}, \widetilde{x}'\rangle \in \mathtt{EXP}_\Gamma$. With this we can use the same reasoning

as in the proof of (A2) for $\mathtt{EXP}_\Gamma$ to construct Q'. If it is stable, the argument reduces to the previous one. If it is not, we have $F(\widehat{x},u) \subset \mathtt{PostQ}_u(\langle \widehat{x},\widetilde{x}\rangle)$. With this, the same arguments as in the proof of (A2) for $\mathtt{EXP}_\Gamma$ show that (A2) holds with inclusion, i.e., $\widetilde{\alpha}(\widetilde{F}(\widetilde{x},u)) \subseteq \widehat{F}(\widetilde{\alpha}(\widetilde{x}),u)$ where $\widetilde{\alpha}(\widetilde{x})$ contains all minimal $\widehat{x}$'s covering $\widetilde{x}$. □

Proof of Thm. 4.6. As sound abstractions compose in the expected way (see Prop. A.1 in the appendix), we obtain a chain of sound abstractions $S^\star \preccurlyeq_{\mathrm{LastX}_{\widehat{S}^\mathsf{K}}} \widehat{S}^\mathsf{K} \preccurlyeq_{\widetilde{\alpha}} \widehat{S}^\sharp$ from Prop. 4.7 and Prop. 4.1, implying $S^\star \preccurlyeq_\alpha \widehat{S}$ with $\alpha = \widetilde{\alpha} \circ \mathrm{LastX}_{\widehat{S}^\mathsf{K}}$. It can be further observed from the tree-structure generated by KAM that every external prefix ν of S corresponds to a unique tuple $(q,c) \in \mathtt{EXP}_\mathsf{X}{}^\downarrow$. Further, the same external prefix ν reaches the state c of $\widehat{S}^\mathsf{K}$ and the state q of $\widehat{S}^\sharp$. As Prop. 4.7 shows that these states c and q are related via $\widetilde{\alpha}$, we have $\mathrm{LastX}_{\widehat{S}} = \widetilde{\alpha} \circ \mathrm{LastX}_{\widehat{S}^\mathsf{K}}$. With this, the first claim of Thm. 4.6 follows. The second claim follows similarly. □

Iterative ABCD with KAM. By combining Cor. 3.4 and Thm. 4.6 we can compute an output-feedback controller $C := \widehat{C} \circ \mathrm{LastX}_{\widehat{S}^\sharp} \in \mathcal{W}(S,\psi)$ from an abstract state-feedback controller $\widehat{C}^\dagger \in \mathcal{W}^\dagger(\widehat{S}^\sharp,\psi)$ whenever the latter synthesis problem allows for such a solution, i.e., $\mathcal{W}^\dagger(\widehat{S}^\sharp,\psi) \neq \emptyset$. Hence, ABCD with output feedback is sound in this case. Given that $\widehat{S}^\sharp$ is in general only known to abstract S, we are however losing completeness. That is, if $\mathcal{W}^\dagger(\widehat{S}^\sharp,\psi) = \emptyset$, it does not imply that there is no solution to the original synthesis problem $\langle S,\psi\rangle$.

We can however take an eager abstraction-refinement approach instead to retain relative completeness. That is, whenever $\mathcal{W}^\dagger(\widehat{S}^\sharp,\psi) = \emptyset$, we run KAM for some more steps, extract a new abstraction $\widehat{S}^{\sharp'}$, and again try to synthesize a controller. We give up, once an upper bound L on the iterations of KAM is reached. This eager approach relies on the insight that abstractions extracted after more iterations of KAM refine earlier abstractions as formalized in Thm. 4.8. Further, this abstraction-refinement procedure is relative complete. That is, if there is a topologically closed finite-state abstraction $\widehat{S}$ for which $\mathcal{W}(\widehat{S},\psi) \neq \emptyset$, there always exists a large enough L s.t. the abstraction $\widehat{S}^\sharp$ extracted from KAM in the L's iteration allows to solve the controller synthesis problem, i.e., $\mathcal{W}(\widehat{S}^\sharp,\psi) \neq \emptyset$.

Theorem 4.8. *Given the premises of Thm. 4.6, let $\widehat{S}^\sharp_{+1}$ be the system computed in line 18 of Alg. 2 after one more iteration of Alg. 2 after $\widehat{S}^\sharp$ was extracted. Then $\widehat{S}^\sharp_{+1} \preccurlyeq \widehat{S}^\sharp$.*

Proof. Let $\mathtt{EXP}_\Gamma$, $\mathtt{EXP}_\mathsf{X}{}^\downarrow$ and $\mathtt{EXP}'_\Gamma$, $\mathtt{EXP}_\mathsf{X}{}^{\downarrow'}$ be the sets computed when extracting $\widehat{S}^\sharp$ and $\widehat{S}^\sharp_{+1}$, respectively. Further let us define an abstraction map candidate α_{+1} using three cases. I.e., $q \in \alpha_{+1}(p)$ if there exists c s.t. either (a) $\langle q,c\rangle \in \mathtt{EXP}_\Gamma$ and $q = p$, or (b) $\langle q,c\rangle \in \mathtt{EXP}_\mathsf{X}{}^\downarrow \setminus \mathtt{EXP}_\Gamma$, $\langle p,c\rangle \in \mathtt{EXP}'_\Gamma$ and $p \subseteq q$, or (c) $\langle p,c\rangle \in \mathtt{EXP}_\mathsf{X}{}^{\downarrow'} \setminus \mathtt{EXP}'_\Gamma$ and there exists c' s.t. q is related to p as in (a) or (b).

This definition induces the following three cases for the proof.
▹ (a) holds for (q,p): This implies $\langle q,c\rangle \in \mathtt{EXP}'_\Gamma$. It follows from the same arguments as used in the proof of Prop. 4.7 that equality holds for (A1)-(A4) in Def. 3.1 w.r.t. $\widetilde{S}$ both for $\widehat{S}^\sharp$ and $\widehat{S}^\sharp_{+1}$. As α_{+1} reduces to the identity map in this case, the claim trivially follows.

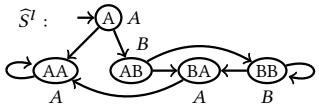

Figure 5: 2-complete abstraction of the system S in Fig. 2.

▹ (b) holds for (q,p): Then it follows again that equality holds for (A1)-(A4) in Def. 3.1 w.r.t. $\widetilde{S}$ for $\widehat{S}^\sharp_{+1}$ but it follows from Thm. 4.6 that only inclusion holds for (A3) w.r.t. $\widehat{S}^\sharp$. Formally, we fix c existentially quantified in the definition of case (b) before. Then we have $\widetilde{\alpha}_{+1}(\widetilde{F}(c,u)) = \widehat{F}_{+1}(\widetilde{\alpha}_{+1}(c),u)$ where $\widetilde{\alpha}_{+1}(c)$ contains the unique minimal p covering c and $\widetilde{\alpha}(\widetilde{F}(c,u)) \subseteq \widehat{F}(\widetilde{\alpha}(c),u)$ where $\widetilde{\alpha}(c)$ contains all minimal q's covering c. We have $p \subseteq q$ for all $q \in \widetilde{\alpha}(c)$ due to the additional refinement step run before extracting $\widehat{S}^\sharp_{+1}$. In particular, we have $\widetilde{\alpha}(c) = \alpha_{+1}(p)$. Hence, $\widetilde{\alpha}_{+1}(\widetilde{F}(c,u)) = \widehat{F}_{+1}(p,u)$ and $\widetilde{\alpha}(\widetilde{F}(c,u)) \subseteq \widehat{F}(\alpha_{+1}(p),u)$. Now define $C' = \widetilde{F}(c,u)$. If for all $c' \in C'$ case (a) or (b) holds, we have that $\widetilde{\alpha}_{+1}(c')$ maps to a unique p'. In this case it holds that $\alpha_{+1}(\widetilde{\alpha}_{+1}(\widetilde{F}(c,u))) = \widetilde{\alpha}(\widetilde{F}(c,u))$ and therefore $\alpha_{+1}(\widehat{F}_{+1}(p,u)) \subseteq \widehat{F}(\alpha_{+1}(p),u)$, what proves the statement. Now for any c'' for which case (c) applies there exists a c''' s.t. case (a) or (b) applies while $\widetilde{\alpha}(c'') = \widetilde{\alpha}(c''')$ and $\widetilde{\alpha}_{+1}(c'') = \widetilde{\alpha}_{+1}(c''')$. With this, the previous argument applies and the claim follows.
▹ (c) holds for (q,p): Fix c existentially quantified in the definition of (c) and recall that there exists c' s.t. $\widetilde{\alpha}(c) = \widetilde{\alpha}(c')$ and $\widetilde{\alpha}_{+1}(c) = \widetilde{\alpha}_{+1}(c')$ and case (a) or (b) applies for c'. Hence, without loss of generality we can replace c by c' and the claim follows. □

Remark 3. *The idea of abstraction-refinement for controller synthesis is also often applied in the context of l-complete abstractions [17, 20, 23, 29]. Similar to KAM, l-complete abstractions are constructed forward and generalize from initial observations to equivalence classes. Here, the equivalence classes collect states which share the same l-long external history (see e.g., Fig. 5 for an example with $l = 2$). l-complete abstractions are typically constructed from the external behavior of S and do not assume the state dynamics of S to be known. They thereby do not utilize the memory structure implicitly given by the state dynamics of S in their generalization step. Therefore, KAM generates tighter abstractions whenever the underlying state transition system is known, but l-complete abstractions are to be preferred if this is not the case.*

Symbolic Implementations. KAM differs from the simultaneous reachability and bisimulation minimization algorithm of Lee and Yannakakis [12] as it constructs an external language- (not bisimulation-) equivalent system. Hence, it only applies predecessor operations and intersection with outputs, but does not take set differences. This is in fact crucial in implementations. For example, for affine systems with polyhedral initial sets and output sets, one can implement the algorithm exactly using a convex polyhedral abstract domain, as both predecessor operators and intersections maintain convexity while set differences do not.

5 HYBRID SYSTEM EXAMPLES

We now present two continuous-state discrete-time hybrid system examples and show how our approach can be used to design abstractions useful for output-feedback control. Along the way, we

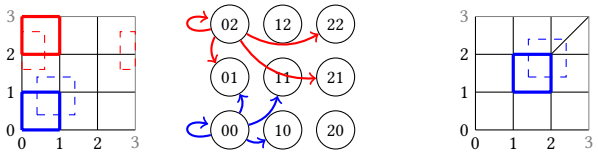

Figure 6: Graphical representation of Σ_1 (far left) and Σ_2 (far right), showing the state space X with the partition induced by the output maps H_1 and H_2, respectively. For Σ_1, $X_j = F(X_i, u_1)$ (dashed blue) indicates the reachable set of $X_i = H_1^{-1}(y_{00})$ (solid blue). Intersecting X_j with the partition generates transitions (blue) originating in y_{00} in the finite-state abstraction (middle). Similarly, $X_j = F(X_i, u_2)$ (dashed red) is reached from $X_i = H_1^{-1}(y_{02})$ (solid red) generating transitions (red) originating in y_{02} in the abstraction.

also compare our approach with several alternatives and show how state-of-the-art techniques for abstracting continuous-state systems, such as those implemented in SCOTS or Mascot [10, 22], can be incorporated in our approach.

Example 5.1. Consider a switched system Σ_1 with ▷ state space $X = [0,3) \times [0,3) \subset \mathbb{R}^2$; ▷ initial states $X_0 = X$; ▷ input space $U = \{u_1, u_2\}$ (corresponding to two controllable modes); ▷ output space $Y = \{y_{00}, y_{01}, y_{02}, y_{10}, y_{11}, y_{12}, y_{20}, y_{21}, y_{22}\}$; ▷ output function $H : x \mapsto y_{ij}$, where $i = \lfloor x_1/3 \rfloor$, $j = \lfloor x_2/3 \rfloor$ for all $x \in X$; and ▷ transition function F defined as

$$F(x, u_1) = mod_3\left(x + \begin{bmatrix}0.4\\0.4\end{bmatrix}\right),\ F(x, u_2) = mod_3\left(x - \begin{bmatrix}0.4\\0.4\end{bmatrix}\right),$$

where the function $mod_k : \mathbb{R}^n \to [0,k)^n$ wraps its input argument component-wise around the perimeter of its codomain; i.e., if $s = mod_k(x)$, then $s_i = x_i - k\lfloor \frac{x_i}{k} \rfloor$. In Fig. 6 (top left), state space X is shown, where the domain of H for all y is indicated by the large boxes with edge length 1. The dynamics of F are then interpreted as upward ($u = u_1$) and downward ($u = u_2$) discrete-time flows of points in X parallel to the diagonal connecting the lower left and top right corner of X. When the boundary of X is reached, the system continues to evolve in the block reached by wrapping X around its boundaries. Note that the only source of non-determinism in system Σ_1 is due to the initial condition not being a singleton, whereas the transition function is deterministic. We consider a specification ψ_1 stating that when starting in y_{00} the system should always eventually (re-)visit y_{00} and y_{22}.

Let us first consider constructing an abstract system $\widehat{\Sigma}_1$ that has a feedback refinement relation (FRR) with Σ_1 by using forward simulation as, e.g., implemented in SCOTS. The main idea is to "grid" the state space into hyperboxes of size η in a way consistent with the outputs and treat each grid cell X_i as an abstract state. Then for each grid cell X_i and for each input u_i, post $F(X_i, u_i)$ is computed and a transition with input label u_i is added from the abstract state X_i to all abstract states X_j that have a non-empty intersection with the post. This process is illustrated in Fig. 6. Given the existence of an FRR from Σ_1 to $\widehat{\Sigma}_1$ (rendering $\widehat{\Sigma}_1$ a sound abstraction of Σ_1 for state-feedback control as discussed in Rem. 1) and the compositionality of sound abstractions (see Prop. A.1), we can use $\widehat{\Sigma}_1$ with any of the algorithms presented in Sec. 4 to construct an abstraction $\widehat{\Sigma}_1'$ which allows to solve the *output-feedback control problem* over Σ_1.

In order to apply this process, we need to select a grid size η when constructing $\widehat{\Sigma}_1$. We denote the resulting abstraction with $\widehat{\Sigma}_1^{(\eta)}$. We can start with $\eta = 1$ as discussed before. This, however induces non-determinism and it can be easily seen by inspecting Fig. 6 (middle), that there does not exist a controller in the abstraction that allows us to surely transition from y_{00} to y_{22} and back infinitely often—in the abstraction, applying the necessary input sequence might lead to visiting y_{02} instead of y_{22}. One can try a finer grid size, e.g., $\eta = 0.03$, but the problem still does not admit a solution for $\langle \widehat{\Sigma}_1^{(0.03)}, \psi_1 \rangle$. By inspection, the problem only has a solution if η is chosen such that 0.2 is an integer multiple of η. Here, 0.2 is the greatest common divisor of 0.4 (the increments the dynamics make) and 1 (the "fidelity" of the outputs). So, the set of grid sizes that gives a solution is a measure-zero set in $\mathbb{R}_{>0}$ and, in general, the "right" grid size is dictated by the dynamics and output map. Further, even if we use an automatic refinement tool like Mascot, the step size of the refinement of η is a design parameter and thus, the tool may not ever explore an integer multiple of 0.2.

We now turn to solving the output-feedback control problem $\langle \Sigma_1, \psi_1 \rangle$ by directly applying the algorithms discussed in Sec. 4 to Σ_1 without constructing $\widehat{\Sigma}_1$ first. For this example, all three algorithms (i.e., KA, KA with bisimulation quotient, and KAM) will produce the same abstraction. This is due to the fact that the dynamics of the system are such that the post and the pre operations over F cancel out. Therefore the forward and backward algorithms are essentially performing the same operations. Further, all of them terminate and generate a sound *realization*. Thus, these algorithms *automatically* figure out that the largest cover of X which merges states with the same future under any applied input sequence has size $\eta = 0.2$.

Example 5.2. We consider another switched system Σ_2 with the same dynamics as Σ_1 but with changed output space $Y_2 = \{y_{00}, \ldots, y_{21}, y_{22u}, y_{22l}\}$ s.t. H_2 maps the upper left and lower right triangle of y_{22} to y_{22u} and y_{22l}, respectively (see Fig. 6 (right) for an illustration). The specification ψ_2 requires to repeatedly visit y_{00} and either y_{22u} or y_{22l} infinitely often after starting in y_{00}.

Consider running KAM on Σ_2. First observe that we are now initializing KAM with the triangle shape domains of $H(y_{22l})$ and $H(y_{22u})$ in addition to the the boxed domains for all remaining outputs. This will result in little triangles right above and right below the diagonal of y_{33}, which collect reachable state subsets with the same output. However, in the remaining part of the state space, KAM will converge to the same rectangular grid as it does for Σ_1. The intuitive reason for this is that the post of any set $H^{-1}(y)$ with $y \notin \{y_{22l}, y_{22u}\}$ remains a box. Therefore, we can never distinguish whether we observe y_{22u} or y_{22l} if we transition to a box on the diagonal of y_{33}, no matter how fine we grid. Further, the post of any such box will be either $\{y_{22u}, y_{22l}\}$ again, $\{y_{00}\}$ (for $u = u_1$) or $\{y_{22}\}$ (for $u = u_2$). With this it is easy to see that boxes of size $\eta = 0.2$ are again the largest partition of X that form equivalence classes respecting observable subsets. KAM will therefore compute the same sound realization for Σ_2 as for Σ_1. If we however run KA

(with or without the bisimulation quotient) one would additionally chop every box of size $\eta = 0.2$ into an upper left and lower right triangle. This unnecessary doubles the state space of the abstraction, but still resulting in a sound realization.

Let us now consider computing an abstraction $\widehat{\Sigma}_2^{(\eta)}$ by forward simulation of Σ_2 first, using SCOTS. Then we immediately get into trouble, because we cannot find a rectangular grid that respects the output map, as needed to fulfill (A3) in Def. 3.1. This approach would therefore directly fail in this example.

Finally, consider a system Σ_3 which has an unbounded state space $X_3 = \mathbb{R}^2$ with transition function defined by F of Σ_1 but without the wrapping of its input argument. The output set Y_3 and the output function H_3 of Σ_3 are given by tiling the entire $\mathbb{R}^2$ space irregularly with the 3x3 blocks of observations Y_1 and Y_2 along with their respective output maps H_1 and H_2. We still have a finite set of inputs and outputs. By recalling that KAM produces the same sound realization for Σ_1 and Σ_2, we can use the same arguments as in the example of Sec. 4.3 to see that KAM will generate the same sound realization for Σ_3 as for Σ_1 and Σ_2, while all other algorithms will produce infinite-state abstractions. Admittedly, while the example distinguishes KAM from the other algorithms, it is not clear how to symbolically represent the algorithm in this case.

ACKNOWLEDGMENTS

This research was funded in part by the DFG project 389792660-TRR 248 and by the ERC under the Grant Agreement 610150. Ozay was supported in part by ONR grant N00014-18-1-2501, NSF grant ECCS-1553873, and an Early Career Faculty grant from NASA's Space Technology Research Grants Program.

REFERENCES

[1] C. Belta, B. Yordanov, and E. A. Gol. *Formal methods for discrete-time dynamical systems*, volume 89. Springer, 2017.
[2] A. Bouajjani, J.-C. Fernandez, and N. Halbwachs. Minimal model generation. In R. Kurshan and E. Clarke, editors, *CAV 90: Computer-aided Verification*, Lecture Notes in Computer Science 531, pages 197–203. Springer-Verlag, 1990.
[3] K. Chatterjee, L. Doyen, T. A. Henzinger, and J. Raskin. Algorithms for omega-regular games with imperfect information. *Logical Methods in Computer Science*, 3(3), 2007.
[4] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL '77*, pages 238–252. ACM, 1977.
[5] R. Ehlers and U. Topcu. Estimator-based reactive synthesis under incomplete information. In *HSCC'15*, pages 249–258. ACM, 2015.
[6] D. Fan and D. C. Tarraf. Output observability of systems over finite alphabets with linear internal dynamics. *IEEE TAC*, 63(10):3404–3417, 2018.
[7] A. Girard, G. Pola, and P. Tabuada. Approximately bisimilar symbolic models for incrementally stable switched systems. *TAC*, 55(1):116–126, 2010.
[8] S. Haesaert, A. Abate, and P. M. Van den Hof. Correct-by-design output feedback of lti systems. In *CDC'15*, pages 6159–6164. IEEE, 2015.
[9] T. A. Henzinger, R. Majumdar, and J. Raskin. A classification of symbolic transition systems. *ACM Trans. Comput. Log.*, 6(1):1–32, 2005.
[10] K. Hsu, R. Majumdar, K. Mallik, and A.-K. Schmuck. Multi-layered abstraction-based controller synthesis for continuous-time systems. In *HSCC'18*, pages 120–129. ACM, 2018.
[11] M. Khaled and M. Zamani. pfaces: an acceleration ecosystem for symbolic control. In *HSCC'19*, pages 252–257. ACM, 2019.
[12] D. Lee and M. Yannakakis. Online minimization of transition systems. In *Proceedings of the 24th Annual Symposium on Theory of Computing*, pages 264–274. ACM Press, 1992.
[13] D. Luenberger. An introduction to observers. *IEEE TAC*, 16(6):596–602, 1971.
[14] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *STACS'95*, volume 900 of *LNCS*, pages 229–242. Springer, 1995.
[15] O. Mickelin, N. Ozay, and R. M. Murray. Synthesis of correct-by-construction control protocols for hybrid systems using partial state information. In *ACC'14*, pages 2305–2311. IEEE, 2014.
[16] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
[17] T. Moor and J. Raisch. Supervisory control of hybrid systems within a behavioural framework. *Systems & Control letters*, 38(3):157–166, 1999.
[18] R. Paige and R. Tarjan. Three partition-refinement algorithms. *SIAM Journal of Computing*, 16(6):973–989, 1987.
[19] J. Reif. The complexity of two-player games of incomplete information. *J. Computer and System Sciences*, 29:274–301, 1984.
[20] G. Reißig. Computing abstractions of nonlinear systems. *IEEE TAC*, 56(11):2583–2598, 2011.
[21] G. Reissig, A. Weber, and M. Rungger. Feedback refinement relations for the synthesis of symbolic controllers. *TAC*, 62(4):1781–1796, 2017.
[22] M. Rungger and M. Zamani. SCOTS: A tool for the synthesis of symbolic controllers. In *HSCC*, pages 99–104. ACM, 2016.
[23] A.-K. Schmuck and J. Raisch. Asynchronous l-complete approximations. *Systems & Control Letters*, 73:67–75, 2014.
[24] J. S. Shamma and K.-Y. Tu. Set-valued observers and optimal disturbance rejection. *IEEE TAC*, 44(2):253–264, 1999.
[25] P. Tabuada. *Verification and control of hybrid systems: a symbolic approach*. Springer, 2009.
[26] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 133–191. Elsevier, 1990.
[27] W. Thomas. On the synthesis of strategies in infinite games. In *STACS'95*, volume 900 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, 1995.
[28] J. Willems. The behavioral approach to open and interconnected systems. *IEEE Control Systems Magazine*, 27:46–99, 2007.
[29] J.-M. Yang, T. Moor, and J. Raisch. Local refinement of l-complete approximations for supervisory control of hybrid systems. *WODES'19*, 2018.
[30] X. Yin and S. Lafortune. A uniform approach for synthesizing property-enforcing supervisors for partially-observed discrete-event systems. *IEEE TAC*, 61(8):2140–2154, 2015.

A ADDITIONAL PROOFS

Proof of Thm. 3.2. We provide theis proof for the more general case of not fully enabled inputs.

For the first claim we pick $\pi = x_0u_0x_1u_1\ldots \in \mathrm{CPaths}(S, C^\dagger)$ with external sequence $\rho = y_0u_0y_1\ldots \in \mathrm{Ext}(S, C^\dagger)$ s.t. $y_k = H(x_k)$ for all $k \in \mathbb{N}$ and show $\rho \in \langle\!|\psi|\!\rangle$.

For $k = 0$, the definition of $\mathrm{CPaths}(S, C^\dagger)$ implies that $x_0 \in X_0$. Using (A1) we know that for all $\widehat{x}_0 \in \alpha(x_0)$ holds that $\widehat{x}_0 \in \widehat{X}_0$. We further have $H(x_0) = y_0$. Now it follows from (A3) that for all $\widehat{x}_0 \in \alpha(x_0)$ we have $y_0 \in \widehat{H}(\widehat{x}_0))$ and therefore $y_0 \in \mathrm{Ext}(\widehat{S}, \widehat{C}^\dagger)|_{[0;0]}$.

For $k > 0$ assume $\rho|_{[0;k-1]} \in \mathrm{Ext}(\widehat{S}, \widehat{C}^\dagger)|_{[0;k-1]}$ and show $\rho|_{[0;k]} \in \mathrm{Ext}(\widehat{S}, \widehat{C}^\dagger)|_{[0;k]}$. Let $\pi = x_0u_0x_1u_1\ldots x_{k-1} \in \mathrm{CPrefs}(S, C)$. Now pick any $\widehat{\pi} = \widehat{x}_0u_0\widehat{x}_1u_1\ldots\widehat{x}_{k-1} \in \alpha(\pi)$ and let $x_k = F(x_{k-1}, u_{k-1})$ and $\widehat{x}_k = \widehat{F}(x_{k-1}, u_{k-1})$ and observe that $\pi u_{k-1}x_k \in \mathrm{CPaths}(S, C^\dagger)$ and $\widehat{\pi}u_{k-1}\widehat{x}_k \in \mathrm{CPaths}(\widehat{S}, \widehat{C}^\dagger)$. Further, it follows from (A2) that $\alpha(x_k) \subseteq \widehat{x}_k$ and therefore $x_k \in \gamma(\widehat{x}_k)$. As $H(x_k) = y_k$, (A3) implies $y_k \in \widehat{H}(\widehat{x}_k)$ and, hence, $\rho|_{[0;k]} \in \mathrm{Ext}(\widehat{S}, \widehat{C}^\dagger)|_{[0;k]}$.

As $\mathrm{Ext}(\widehat{S})$ is topologically closed, so is $\mathrm{Ext}(\widehat{S}, \widehat{C}^\dagger)$. With this $\rho|_{[0;k]} \in \mathrm{Ext}(\widehat{S}, \widehat{C}^\dagger)|_{[0;k]}$ for all $k \in \mathbb{N}$ implies $\rho \in \mathrm{Ext}(\widehat{S}, \widehat{C}^\dagger)$. As $\widehat{C}^\dagger \in \mathcal{W}^\dagger(\widehat{S}, \psi)$, we have $\mathrm{Ext}(\widehat{S}, \widehat{C}^\dagger) \subseteq \langle\!|\psi|\!\rangle$ and, hence, $\rho \in \langle\!|\psi|\!\rangle$.

For the second claim, one can verify that $S \cong_\alpha^\gamma \widehat{S}$ implies $\widehat{S} \preccurlyeq_\gamma S$ and by this $C' \in \mathcal{W}^\dagger(S, \psi)$ implies $\widehat{C}' = C' \circ \gamma \in \mathcal{W}^\dagger(\widehat{S}, \psi)$ from the first part of this theorem. Hence, either $\mathcal{W}^\dagger(S, \psi) = \emptyset$ or $\mathcal{W}^\dagger(\widehat{S}, \psi) \neq \emptyset$. The "only if" part follows analogously from the inverse direction. □

The next proposition shows the compositionality of sound abstraction relations.

Proposition A.1. *Let* $(S_1, \lambda_1) \preccurlyeq_{\alpha_{12}} (S_2, \lambda_2)$ *and* $(S_2, \lambda_2) \preccurlyeq_{\alpha_{23}} (S_3, \lambda_3)$ *then* $(S_1, \lambda_1) \preccurlyeq_{\alpha_{13}} (S_3, \lambda_3)$ *with* $\alpha_{13} = \alpha_{23} \circ \alpha_{12}$.

Proof. We show that (A1)-(A3) in Def. 3.1 hold by using the observation that $\alpha_{12}(x_1) \subseteq X_2$ and $\alpha_{23}(x_2) \subseteq X_3$ for $x_1 \in X_1, x_2 \in X_2$. Further, we define γ_{ji} as the induced inverses of the respective α_{ij}.
▶ (A1) As $\alpha_{12}(X_{1,0}) \subseteq X_{2,0}$ and $\alpha_{23}(X_{2,0}) \subseteq X_{3,0}$ it follows that $\alpha_{23}(\alpha_{12}(X_{1,0})) \subseteq X_{3,0}$.
▶ (A2.1) As $\mathrm{Enab}_{S_3}(\alpha_{23}(x_2)) \subseteq \mathrm{Enab}_{S_2}(x_2)$ and $\mathrm{Enab}_{S_2}(\alpha_{12}(x_1)) \subseteq \mathrm{Enab}_{S_1}(x_1)$ for any $x_1 \in X_1$ and $x_2 \in X_2$ it follows that $\mathrm{Enab}_{S_3}(\alpha_{23}(\alpha_{12}(x_1))) \subseteq \mathrm{Enab}_{S_2}(\alpha_{12}(x_1)) \subseteq \mathrm{Enab}_{S_1}(x_1)$.
▶ (A2.2) As $\alpha_{12}(F_1(x_1,u)) \subseteq F_2(\alpha_{12}(x_1),u)$ and $\alpha_{23}(F_2(x_2,u)) \subseteq F_3(\alpha_{23}(x_2),u)$ for any $x_1 \in X_1$ and $x_2 \in X_2$ it follows that $\alpha_{23}(\alpha_{12}(F_1(x_1,u)) \subseteq \alpha_{23}(F_2(\alpha_{12}(x_1),u)) \subseteq (F_3(\alpha_{23}(\alpha_{12}(x_1)),u))$.
▶ (A3) As $\lambda_1(H_1(\gamma_{21}(x_2)) \subseteq \lambda_2(H_2(x_2))$ and $\lambda_2(H_2(\gamma_{32}(x_3)) \subseteq \lambda_3(H_3(x_3))$ for any $x_2 \in X_2, x_3 \in X_3$ it follows that $\lambda_1(H_1(\gamma_{21}(\gamma_{32}(x_3))) \subseteq \{\lambda_2(H_2(\gamma_{32}(x_3)))\} \subseteq \lambda_3(H_3(x_3))$. □

The following technicallemmas are used in the analysis of the KAM algorithm.

Lemma A.2. *After execution of the function Refine in line 15 of Alg. 2, it holds that $|\widetilde{\alpha}(\widetilde{x})| = 1$ for all $\widetilde{x}$ for which $\widetilde{\alpha}$ is defined.*

Proof. First observe that Refine is only called if $\widetilde{x} \subset \widehat{x}$. If $\widetilde{x} = \widehat{x}$ the claim is trivially satisfied as $\widehat{x}$ is the unique minimal element covering $\widetilde{x}$ in this case. As Q' in line 11 of Alg. 2 is chosen to be minimal, we have that $\langle \widehat{x}_1, \widetilde{x}\rangle, \langle \widehat{x}_2, \widetilde{x}\rangle \in \mathtt{EXP}_\Gamma$ with $\widehat{x}_1 \neq \widehat{x}_2$ implies $\widehat{x}_1 \not\subseteq \widehat{x}_2$ and $\widehat{x}_2 \not\subseteq \widehat{x}_1$ and in addition $\widetilde{x} \subset \widehat{x}_1$ and $\widetilde{x} \subset \widehat{x}_2$, so Refine is called. Further, as Alg. 2 is initialized with a cover which partitions the state space, we know that there exists a minimal $\widehat{x}$ which was split into $\widehat{x}_1 \subset \widehat{x}$ and $\widehat{x}_2 \subset \widehat{x}$ previously. This implies that there exists $\widetilde{x}_1$ and $\widetilde{x}_2$ s.t. $F(\widehat{x}_1,u) = \mathtt{PostQ}_u(\langle \widehat{x}_1, \widetilde{x}_1\rangle)$ and $F(\widehat{x}_2,u) = \mathtt{PostQ}_u(\langle \widehat{x}_2, \widetilde{x}_2\rangle)$ while there exists some $x_1 \in \widehat{x}_1 \setminus \widehat{x}_2$ s.t. $x \notin \mathtt{PostQ}_u(\langle \widehat{x}_2, \widetilde{x}_2\rangle)$ and, vise versa, there exists some $x_2 \in \widehat{x}_2 \setminus \widehat{x}_1$ s.t. $x \notin \mathtt{PostQ}_u(\langle \widehat{x}_1, \widetilde{x}_1\rangle)$, as otherwise the cover cell $\widehat{x}$ would not have been splitted. Now, consider $\widetilde{x}$ from before, and observe that $\widetilde{x} \in \widehat{x}_1 \cap \widehat{x}_2$ by definition. Further, the above reasoning implies $\widetilde{x} \subset \widetilde{x}_1$ and $\widetilde{x} \subset \widetilde{x}_2$, and $\mathtt{PostQ}_u(\langle \widehat{x}_1, \widetilde{x}\rangle) \subset \mathtt{PostQ}_u(\langle \widehat{x}_1, \widetilde{x}_1\rangle) = F(\widehat{x}_1,u)$ and $\mathtt{PostQ}_u(\langle \widehat{x}_2, \widetilde{x}\rangle) \subset \mathtt{PostQ}_u(\langle \widehat{x}_2, \widetilde{x}_2\rangle) = F(\widehat{x}_2,u)$ with proper containment in both cases. This introduces a contradiction to the assumption that Refine has terminated, as in this case we know that $\widehat{x}_1$ and $\widehat{x}_2$ cannot be further splitted, i.e., $F(\widehat{x}_1,u) = \mathtt{PostQ}_u(\langle \widehat{x}_1, \widetilde{x}\rangle)$ and $F(\widehat{x}_2,u) = \mathtt{PostQ}_u(\langle \widehat{x}_2, \widetilde{x}\rangle)$. The last equality only holds if $\widehat{x}_1 = \widehat{x}_2$ as in this case no x_1 and x_2 as above can be constructed. □

Lemma A.3. *If Alg. 1 terminates, there exists an iteration $l \in \mathbb{N}$ of Alg. 2 for which $\mathtt{EXP}_\Gamma = \mathtt{EXP}_\mathtt{X}{}^{\downarrow}$ holds.*

Proof. First, it can be verified that in every iteration of the while loops in both algorithms $\langle \cdot, c\rangle$ gets added to $\mathtt{EXP}_\mathtt{X}{}^{\downarrow}$ in Alg. 2 iff c gets added to $\widehat{X}$ in Alg. 1. This is due to the fact that the set of minimal blocks covering c is uniquely defined and refinements of any block are propagated through all sets $\mathrm{EXP}_{\Gamma,X,F}$ within Refine. Therefore, it cannot happen that a tuple $\langle q, c\rangle$ is added to $\mathtt{EXP}_\mathtt{X}{}^{\downarrow}$ if $\mathtt{EXP}_\mathtt{X}{}^{\downarrow}$ already contains a tuple $\langle q', c\rangle$. Thus, the termination conditions of the while loops coincide.

As Refine is a recursive function, we have to additionally prove that it terminates. To see this, observe that $\mathtt{EXP}_\mathtt{F}$ is a finite tree for every initial tuple $\langle \varepsilon, q, q\rangle$ and therefore only contains finite paths. Further, as every iteration of the while loop in line 5-21 of Alg. 2 only explores the current leaves of this tree, it adds new leaves to the tree and schedules leaves of the previous iteration for possible refinement. As the recursion of Refine in line 35 of Alg. 2 only schedules predecessors of these leaves and the tree is finite, it terminates. □