Steady Abstractions for CPS Controller Synthesis*

Francisco Palau Romero Rüdiger Ehlers

***Abstract*— Designing correct-by-construction controllers for cyber-physical systems is difficult. Synthesis aids the controller designer by automating this process once the controller's specification and the dynamics of the environment have been formalized. Most classes of system dynamics have undecidable synthesis problems, which is commonly mitigated by computing a discrete abstraction of the system dynamics and synthesizing a controller that works correctly under all environment behaviors captured by the abstraction. Abstractions can be made at varying levels of granularity. If the granularity is too fine, the abstraction grows too large to be suitable for current synthesis approaches. If the granularity is too coarse, it is insufficient for computing a controller that satisfies the specification.**

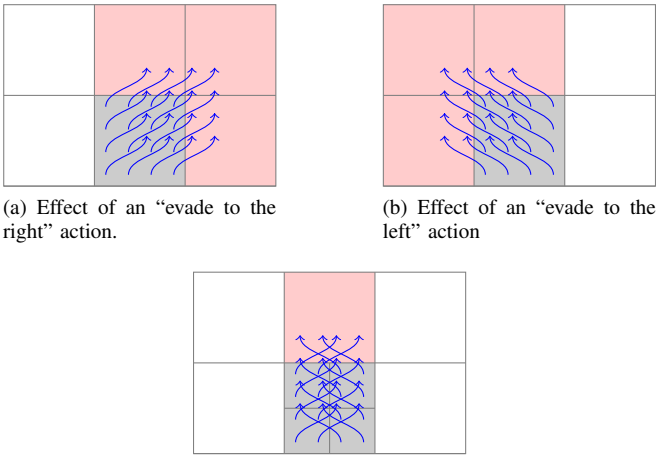
In this paper, we present a new abstraction computation process that facilitates working with coarser abstractions in synthesis while retaining the controllability of the system. Our approach takes a fine-grained abstraction and processes it to a coarser abstraction in which the controller is only offered actions that minimize the amount of non-determinism in the abstraction. We demonstrate that our approach helps to find a good balance between precision and controllability in CPS controller synthesis on two case studies.

I. Introduction

The commonly used approach to synthesize controllers for cyber-physical systems with at least moderately complex dynamics and specifications is to first compute a discrete abstraction of the system's physical dynamics, and to synthesize a controller that works correctly in an environment whose behavior is captured by the discrete abstraction. In order for the synthesized controller to be deployable in the field, the abstraction must *alternatingly simulate* [1], [2] the real system.

Alternating simulation abstractions can be built at various degrees of granularity. Coarse abstractions have few discrete states, which reduces the computation time of the synthesis process in which the abstraction is to be used. However, a coarse abstraction makes it more difficult to control the system, as the abstraction needs to *overapproximate* the possible behavior of the physical environment. If an abstraction is too coarse, then due to overapproximation, it contains spurious traces that prevent the controllability of the system. To counter this problem, more fine-grained abstractions can be computed. Since this increases the number of discrete states in an abstraction, this makes the synthesis process computationally more difficult, and is thus avoided in practice whenever possible. Hence, physical environments that require a very fine abstraction in order to be controlled are currently out of reach for CPS controller synthesis approaches that work with discrete abstractions.

The authors are with the University of Bremen, Germany.



(a) Effect of an "evade to the right" action.

(b) Effect of an "evade to the left" action

(c) Composite *steady* action

Fig. 1: Motivating example for steady abstractions.

In this paper, we present a new approach to compute alternatingly simulating abstractions that are relatively coarse, yet capture the main characteristics of finer abstractions. Our approach is agnostic to the concrete system dynamics as it post-processes a relatively fine-grained abstraction to a coarser one. The main idea is that when making the abstraction coarser, we replace the actions by *steadier actions*; these are the ones that minimize the number of discrete states reached under the action, and we use the fine abstraction to enumerate such steady actions. Figure 1 demonstrates how this process can lead to coarser abstractions that are small, yet useful. Shown are parts of a two-dimensional abstraction of the dynamics of a *unicycle*, where the system state is characterized by the X and Y positions of the unicycle in the workspace and the current heading and speed. Figure 1a demonstrates the effect of applying an "evade to the right" action from the middle cell (for some fixed current speed and heading) in the abstraction. The arrows depict how the abstracted system evolves from the states represented by the cell, and the abstraction needs to have all four marked abstraction cells as successors in order to alternatingly simulate the real-world physical system. Figure 1b shows the same setting for an "evade to the left action". Again, the discrete abstraction needs to have four successor states. For computing the steady abstraction in Figure 1c, we start with a finer abstraction and then postprocess it to a steadier coarse abstraction. Here, the abstraction is finer by a factor of two, so every cell has multiple subcells. In each subcell, we choose actions such that the overall number of states in the

abstraction is minimal. In this example, we can use "evade to the right" actions in the left subcells, and "evade to the left" actions in the right subcells. This strategic choice minimizes the number of states reached in the abstraction under the abstract action and is thus used in our steady abstraction.

Our approach is particularly interesting for postprocessing alternatingly similar time- and state-quantized abstractions of linear and non-linear systems. These are relatively easy to compute (for example using toolkits such as SCOTS [3] or PESSOA [4]), but difficult to use in synthesis, as for all but the simplest system dynamics, they exhibit a high degree of non-determinism. Our approach mitigates this problem by computing abstractions with a lower degree of non-determinism from finer abstractions. It thus helps with getting synthesis of controllers for complex dynamics more scalable. We demonstrate the effectiveness of our abstraction processing technique on two case studies with ω-regular specifications. The first case study features vehicle dynamics and a specification that has both safety and liveness components, while the second one uses moon lander dynamics and a reachability specification.

II. Related Work

Discrete abstractions have been proven to be useful for computing controllers for linear and non-linear systems [5], [5], [6], [7], [8], [9], [10], [11], [12]. While in order to be useful for control, cyber-physical system dynamics abstractions need to over-approximate the system's behavior, there exist several concretizations of the concept that differ in the type of simulation relation that they build between the concrete and abstract systems.

The use of exact alternating simulation relations [13], [14] for controller synthesis assumes full and precise observation of the system to be controlled. Approximate alternating simulation relations [15], [16] are a generalization of this concept for controlling systems under a measurement imprecision ϵ and under disturbances [17]. Feedback refinement relations [18] are another refinement of alternating simulation relations that is orthogonal to approximate simulation. Controllers that have been synthesized using a discrete system abstraction that is in relation to the physical system by a feedback refinement relation do not need to observe the full physical system state, and only need quantized state information. Furthermore, for abstractions that are related to the concrete system by feedback refinement, the structure of the abstraction ensures that controller actions can be used as control input without an additional mapping into the concrete system domain. When processing a discrete abstraction that has a feedback refinement relation with the environment with the approach that we present in this paper, the former of these two properties is retained, but the latter of these properties is not.

To the best of the authors' knowledge, no rigorous post-processing approach for alternating simulation relations and their refinements has been proposed so far. For this initial treatment, we focus on classical alternating simulation relations to keep the presentation concise, and defer a closer look at the adaptation of our approach to ϵ-approximate alternating simulation relations to future work.

III. Preliminaries

a) Basics: Given some set K, we denote the set of finite strings from characters in K as K^* and the set of infinite strings over K as K^ω. Given some function $f : K \to M$ for some sets K and M, we also view f as a relation such that for some $k \in K$ and $m \in M$, we have $(k, m) \in f$ if and only if $f(k) = m$. The composition between two relations R and R' is defined as $R \circ R' = \{(x, y) \mid \exists z.(x, z) \in R \wedge (z, y) \in R')\}$. Given some function $f : K \to M$, its inverse f^{-1} is a function from M to 2^K such that $f^{-1}(m) = \{k \in K \mid f(k) = m\}$ for all $m \in M$.

b) Discrete-time systems: We consider discrete-time system dynamics of the form

$$y = f(x, u, d),$$

where $y \in X$ and $x \in X$ are the states after and before a step of the system's execution, $u \in U$ is a control signal, $d \in D$ is the disturbance, and $f : X \times U \times D \rightharpoonup X$ is an arbitrary partial function. We can also represent f as a set of *transitions* $T \subseteq X \times U \times X$ by defining $T = \{(x, u, y) \mid \exists d \in D.f(x, u, d) = y\}$. We also call (X, U, T) a *system description*. We say that a word $w = (w_0^X, w_0^U)(w_1^X, w_1^U) \ldots \in (X \times U)^\omega \cup (X \times U)^*$ is a trace of (X, U, T) if for every $i \in \mathbb{N}$ (smaller than its length if w is infinite), we have $(w_i^X, w_i^U, w_{i+1}^X) \in T$. We only consider maximal finite traces in the following, i.e., those that cannot be extended to a longer trace. We also define $\delta_{X,T}(x) = \{u \in U \mid \exists y \in X.(x, u, y) \in T\}$ for all $x \in X$.

We say that a triple (X', U', T') is an *alternating simulating abstraction* of a system description (X, U, T) if X' and U' are sets, $T' \subseteq X' \times U' \times X'$, and there exists a function $R_X : X \to X'$ such that for all $x \in X$ and $u' \in U'$, there exist a $u \in U$ such that for all $y \in X$ with $(x, u, y) \in T$, we have $(R_X(x), u', R_X(y)) \in T'$. We call R_X the *alternating simulation relation* between (X, U, T) and (X', U', T').[1]

Alternating simulation relations can also be nested. If a system (X', U', T') alternatingly simulates a system (X, U, T) by a relation R_X, and a system (X'', U'', T'') alternatingly simulates (X', U', T') by a relation R'_X, then $R'_X \circ R_X$ is an alternating simulation relation between (X, U, T) and (X'', U'', T'') ([1], Proposition 4.23).

Feedback refinement relations [18] are a refinement of alternating simulation relations. A triple (X', U', T') is a *feedback refinement abstraction* of some system (X, U, T) if there exists a *feedback refinement relation* $R \subseteq X \times X'$ such that $U' \subseteq U$, we have $\delta_{X,T}(x) \supseteq \delta_{X',T'}(x')$ for every $(x, x') \in R$, and for every $(x, x') \in R$ and $u \in \delta_{X',T'}(x')$, we have $\{y' \in X' \mid \exists y \in X.(y, y') \in R, (x, u, y) \in T\} \subseteq \{y' \in X' \mid (x', u, y') \in T'\}$.

[1] In other works employing alternating simulation relations, f_X may also be a set-valued function. Not considering this possibility here is for notational convenience only.

c) Controller Synthesis: Given a system (X, U, T), some set of initial states $X_0 \subseteq X$ and some specification $L \subseteq (X \times U)^\omega$, the synthesis problem is to compute a *strategy* $s : X^* \to U$ such that for all $x_0 \in X_0$ and all traces $w = (w_0^X, w_0^U)(w_1^X, w_1^U) \dots$ of (X, U, T), if $w_0^X = x_0$ and for all $i \in \mathbb{N}$, we have $w_{i+1}^U = s(w_0^X w_1^X \dots w_i^X)$, then $w \in L$. A specification that admits a strategy for some given system (X, U, T) is called *realizable*, otherwise it is *unrealizable*.

If a strategy can be represented with a finite number of states, then we say that it can be implemented in a *controller* that simulates the strategy, and hence controls the physical system to satisfy the specification.

Controllers can also be computed over abstractions. In this case, we modify the specification L to a specification $L' \subseteq (X' \times U')^\omega$ such that for all $w' = (w_0'^X, w_0'^U)(w_1'^X, w_1'^U) \dots \in L'$ and all $w = (w_0^X, w_0^U)(w_1^X, w_1^U) \dots \in L$, if we have that $(w_i, w_i') \in R_X$ for all $i \in \mathbb{N}$, then $w \in L$. Given a specification L', a strategy implementing it over an abstraction is a function $s : X'^* \to U'$ such that for all $x_0 \in X_0$ and all traces $w' = (w_0'^X, w_0'^U)(w_1'^X, w_1'^U) \dots$ of the system, if $(x_0, w_0'^X) \in R_X$ and $w_i'^U = s(w_0'^X \dots w_i'^X)$ for all $i \in \mathbb{N}$, then $w' \in L'$.

Whenever a strategy for the abstraction is found, the alternating simulation between the original system and the abstraction guarantees the existence of a corresponding strategy for the original system [17], [1]. If L' belongs to the class of ω-regular properties (see, e.g., [19]), then if a strategy to control L' on the abstraction exists, it is finite-state and can be implemented as a controller for the original system.

By the definition of a strategy (over an abstraction), a strategy must avoid to reach the set of *bad states*, i.e., the largest set of states X'_{bad} such that there is no strategy enforcing $L' = (X' \times U')^\omega$ from any state in X'_{bad}. Bad states represent, for instance, situations in which there is no way to control the system not to leave its workspace boundaries.

Note that the complexity of checking the realizability of a specification L' over some abstraction (X', U', T') is superlinear in $|X'|$ for most specification classes. For example, the class of generalized reactivity(1) specifications has a complexity that is quadratic or cubic in the size of X' [20], depending on whether the transition relation is stored in explicit or symbolic form. Thus, keeping the size of an abstraction small is of importance for the efficiency of the synthesis process.

d) Satisfiability Solving: The Boolean satisfiability problem (SAT) is the problem of finding a satisfying assignment to the variables of a Boolean formula, whenever it exists. Nowadays, heuristic SAT algorithms are widely used to solve problems with several thousand of variables, which is sufficient for many practical SAT problems [21].

Modern SAT solvers typically require the input formula to be given in conjunctive normal form (CNF). A *clause* is a disjunction of *literals*, and these literals are either a variable or its negation. A Boolean formula in CNF is then a conjunction of clauses. For example, the Boolean formula $f(a, b, c) = (\neg a \vee b) \wedge (a \vee c)$ is in CNF form.

Many modern solvers can also be run in *incremental* mode, where a SAT instance is gradually built and can already be checked for satisfiability before the instance is finalized. Incremental SAT solving is more efficient than rebuilding the SAT instance from scratch every time its satisfiability is to be checked, as so-called *learned clauses* are kept between the solver runs, but clauses can only be added between the SAT solver runs and not removed. Some solvers furthermore support *assumed* literals [22]. The solver then searches only for solutions that in addition to the clauses also satisfy all assumed literals.

IV. Problem Definition

In the following, we assume that we have a procedure to compute a finite-state discrete abstraction (X', U', T') from some system description (X, U, T). For the simplicity of presentation, we assume that $X = [0, d_1] \times \dots \times [0, d_n]$ for some $d_1, \dots, d_n \in \mathbb{R}$ and $n \in \mathbb{N}$, and that we have $X' = \{0, \dots, d_1'\} \times \dots \times \{0, \dots, d_n'\}$ for some $d_1', \dots, d_n' \in \mathbb{N}$ as state space of the abstraction.

We want to reduce the *degree of non-determinism* of (X', U', T'), which is defined as follows:

Definition 1: Given an abstract system (X', U', T') with the set of bad states X'_{bad}, we define the *degree of non-determinism* of T' as:

$$\mathsf{don}(T') = \max_{x' \in X' \setminus X'_{bad}} \min_{u' \in U', \exists y' : (x', u', y') \in T'} (|\{y' \in X' : (x', u', y') \in T'\}|).$$

Abstractions with a high degree of non-determinism are difficult to control, as a strategy needs to work correctly however the non-determinism is resolved at runtime. We removed the bad states from consideration in this definition, as no controller (for any specification) can make use of them without violating its specification in the long run. Whenever the high degree of non-determinism leads to the *unrealizability* of a specification under an abstraction, this can be solved by making the abstraction finer (e.g., by increasing the values $d_1', \dots, d_n'$). While the degree of non-determinism of the abstraction typically becomes higher with this change, a finer abstraction enables the controller to base its decisions on more precise data of the current system state. As finer abstractions lead to higher computation times in controller synthesis, we want to postprocess them to a coarser abstraction whose degree of non-determinsm is smaller than if we computed a coarser, non-postprocessed abstraction.

Definition 2: Let (X, U, T) be a (discrete-time) discrete-state system (or a discrete abstraction of a discrete-time system) with the state space $X = \{0, \dots, d_1\} \times \dots \times \{0, \dots, d_n\}$.

Let furthermore a sequence of *compression factors* $c_1, \dots, c_n \in \mathbb{N}$ be given such that for all $i \in \{1, \dots, n\}$, c_i divides d_i, and (X', U', T') be a discrete system description with $X' = \{0, \dots, \frac{d_1}{c_1}\} \times \dots \times \{0, \dots, \frac{d_n}{c_n}\}$. We define a mapping $m : X \to X'$ such that $m(x_1, \dots, x_n) = (\lfloor \frac{x_1}{c_1} \rfloor, \dots, \lfloor \frac{x_n}{c_n} \rfloor)$ for all $(x_1, \dots, x_n) \in X$. We also define $\theta(x', a) = \{m(y) \mid y \in X, \exists x \in m^{-1}(x'), (x, a(x), y) \in T\}$ for all $x' \in X'$ and all $a : m^{-1}(x') \to U$.

We say that (X', U', T') is a *steady abstraction* of (X, U, T) if for every $x' \in X'$ and $u' \in U'$, for $succ = \{y' \in X' \mid (x', u', y') \in T'\}$, there exists an $a : m^{-1}(x') \to U$ such that $succ = \theta(x', a)$, and there exists no assignment $a' : m^{-1}(x') \to U$ such that $\theta(x', a') \subset succ$.

In a steady abstraction, every available action minimizes the set of reachable abstraction states under the action. A finer abstraction is used to denote which abstraction states can be reached or avoided, and from each state x' in the steady abstraction, it must define actions for each state in $m^{-1}(x')$ that jointly minimize the set of reached states in the abstraction.

The following lemma establishes alternating simulation between a discrete-state system and a steady abstraction computed from it:

Lemma 1: If (X', U', T') is a steady abstraction of some discrete-time system (X, U, T), then (X', U', T') is an alternating simulation abstraction of (X, U, T).

Proof: By the definition of the concept of alternating simulation abstractions, we need to show that there exists a function $R_X : X \to X'$ such that for all $x \in X$ and $u' \in U'$, there exists a $u \in U$ such that for all $y \in X$ with $(x, u, y) \in T$, we have $(R(x), u', R(y)) \in T'$.

We choose $R_X = m$. Let $x \in X$ and $u' \in U'$ be arbitrary. To satisfy the definition of alternating simulation abstractions, we set $u = a(x)$ for the a function to which u' corresponds by the definition of steady abstractions. Let now $(x, u, y) \in T$. By the definition of Θ, we have that $\theta(x', a)$ contains $m(y)$ for all states $y \in X$ with $(x, u, y) \in T$ if $x' = m(x)$. Since $\{y' \in X' \mid (x', u', y') \in T'\} \supseteq \theta(x', a)$ (by the definition of *succ*), it follows that we have $(R_X(x), u', R_X(y)) \in T'$, which was to be proven. ∎

This lemma enables us to apply steady abstractions in the context of cyber-physical system control. If we compute an abstraction of a physical system that alternatingly simulates the physical system, and we post-process this abstraction to a steady abstraction, then by Proposition 4.23 of [1], we know that the steady abstraction alternatingly simulates the physical system as well. This enables us to implement a controller that we synthesize for the steady abstraction in the field.

Reissig et al. [18] refined the concept of alternating simulation relations to *feedback refinement relations*, which give rise to controllers that (1) only use quantized control input and (2) use the same control actions in physical system states that are mapped to the same state in the abstraction by the refinement relation. The former property is retained when processing a feedback refinement abstraction with our approach. This follows directly from the fact that a steady abstraction alternatingly simulates the abstraction from which it was computed. The second property of feedback refinement relations is however not retained in steady abstractions. This means that when implementing a controller computed with our approach in the field, it needs to be slightly larger than a controller synthesized for a feedback refinement abstraction, as it needs to include a look-up table for selecting the physical system actions for every action/state combination in the steady abstraction.

Note that the definition of a steady abstraction leaves the choice of actions in the steady abstraction relatively open. We will later discuss reasonable choices of actions.

V. Computing Steady Abstractions

As explained in Definition 2, we aim to compute a steady abstraction (X', U', T') of a discrete system description (X, U, T) for some compression factors $c_1, \ldots, c_n \in \mathbb{N}$. We assume that (X, U, T) alternatingly simulates some real-world physical system. Def. 2 already defines the structure of X' in a steady abstraction, so it suffices to compute U' and T'.

A post-processed abstraction (X', U', T') is steady if for every $x' \in X'$ and every $u' \in U'$, the set of successors for x' and u' in T' is as small as possible. More formally, for some abstract state x', we are searching for assignments $a : m^{-1}(x') \to U$ such that for no $a' : m^{-1}(x') \to U$, we have $\theta(x', a) \supset \theta(x', a')$. We can obtain such an assignment a by solving a sequence of satisfiability problems, which can be done with off-the-shelf satisfiability (SAT) solvers. We use the following two sets of Boolean variables:

- $V = \{v_{y'}\}_{y' \in \{m(y) \mid \exists x \in m^{-1}(x'), u \in U.(x,u,y) \in T\}}$ represent every state in X' that is reachable from x' under some arbitrary action assignment a',
- $B = \{b_{x,u}\}_{x \in m^{-1}(x'), u \in \delta_{X,T}(x)}$ represent an assignment a that we search for.

The main idea of the following encoding is to represent both a and $\theta(x', a)$ as a model of a Boolean formula in CNF, and to ensure that $\theta(x', a)$ is minimal by successively replacing a by assignments a' with $\theta(x', a') \subset \theta(x', a)$ until no such assignment a' can be found any more. The last assignment then represents a valid action in a steady abstraction.

In order to ensure that the model of the Boolean formula over $V \uplus B$ encodes a function $a : m^{-1}(x') \to U$, at least one variable for each subset $\{b_{x,u}\}_{u \in U}$ must evaluate to $\mathbf{true}$ for each $x \in m^{-1}(x')$. We use a clause $\bigvee_{u \in \delta_{X,T}(x)} b_{x,u}$ for every $x \in m^{-1}(x')$ to enforce this.

Then, we need clauses that enforce that for every $x \in m^{-1}(x')$ and for the encoded function a, all states $m(y)$ with $(x, a(x), y) \in T$ are part of the encoded set $\theta'(x, a)$. Adding a clause $\neg b_{x,u} \vee \bigvee_{y' \in \theta(x',u)} v_{y'}$ for every $x \in m^{-1}(x')$ and $u \in U$ is suitable for this task.

We use these clause types in an iterative SAT solving process that is described in Algorithm 1. In the main loop of the algorithm, we search for all assignments a that minimize the set $\theta(x', a)$. Whenever a new assignment a is found, it is tested whether another assignment a' can be found with $\theta(x', a') \subset \theta(x', a)$. To test this, in line 15, we force the next assignment a' to induce a strict subset of reachable states. The strictness requirement is implemented by adding a clause in line 17, while enforcing that $\theta(x', a') \subseteq \theta(x', a)$ holds is done in line 15 by requiring the SAT solver to yield solutions in which all variables in $V \setminus \{v_y \mid y \in \theta(x', a)\}$ have $\mathbf{false}$ values.

Note that the clause added in line 17 can remain in the SAT instance after a new assignment a has been found without

Algorithm 1 Algorithm to compute a steady abstraction from a given abstraction (X, U, T)

1: **function** COMPUTESTEADYABSTRACTION(X, U, T)
2: $X' \leftarrow \{m(x) \mid x \in X\}$
3: $U' \leftarrow \emptyset$
4: $T' \leftarrow \emptyset$
5: $\psi \leftarrow \mathbf{true}$
6: **for** $x' \in X'$ **do**
7: $V \leftarrow \{v_{y'}\}_{y' \in \{m(y) \mid (x,u,y) \in T, x \in m^{-1}(x'), u \in U\}}$
8: $B \leftarrow \{b_{x,u}\}_{x \in m^{-1}(x'), u \in \delta_{X,T}(x)}$
9: $\psi \leftarrow \psi \wedge \bigwedge_{x \in m^{-1}(x')} \bigvee_{u \in \delta_{X,T}(x)} b_{x,u}$
10: $\psi \leftarrow \psi \wedge \bigwedge_{x \in m^{-1}(x'), u \in \delta_{X,T}(x), y \in X, (x,u,y) \in T}$
11: $\neg b_{x,u} \vee v_{m(y)}$
12: $\psi \leftarrow \psi \wedge \bigvee_{v_{y'} \in V, y' \neq x'} v_{y'}$
13: **while** ψ is satisfiable **do**
14: $w \leftarrow$ model of ψ
15: **while** $\psi \wedge \bigwedge_{v \in V, w(v) = \mathbf{false}} \neg v$ is satisfiable **do**
16: $w \leftarrow$ model of ψ
17: $\psi \leftarrow \psi \wedge \bigvee_{v \in V, w(v) = \mathbf{true}} \neg v$
18: $succ \leftarrow \{y' \in X' \mid w(v_{y'}) = \mathbf{true}\}$
19: $act = \max\{u' \in \mathbb{N} \mid \exists y' \in X'.(x', u', y')$
20: $\in T'\} + 1$
21: $T' \leftarrow T' \cup \{(x', act, y') \mid y' \in succ\}$
22: $U' \leftarrow U' \cup \{act\}$
23: **return** (X', U', T')

making the search process non-incremental, as every other assignment a' found later in the search process needs to have $\theta(x', a') \not\supset \theta(x, a) \neq \emptyset$ in order to be a valid solution, hence the added clause needs to be fulfilled in all future solutions. The same clause also ensures that the next assignment a found in the main loop needs to have at least one different successor state, so that the main loop eventually terminates.

By the fact that Algorithm 1 computes action assignments a that minimize $\theta(x', a)$, it minimizes the degree of non-determinism (Definition 1) in the abstraction (X', U', T').

To speed up the algorithm, it makes sense to remove all *dominated* actions from (X, U, T) before running it. This means that whenever for some state $x \in X'$ and two actions $u, u' \in U$, we have $\{y \in X, (x, u, y) \in T\} \subset \{y \in X, (x, u', y) \in T\}$, we remove all transitions (x, u', y) from T, as action u is then a strictly better choice than u' in state x.

The complexity of Algorithm 1 is dominated by the number of invocations of the SAT solver. The *succ* sets found in the loop starting in line 13 are pairwise incomparable and subsets of $\kappa_{x'} = \{y' \in X' \mid \exists u' \in U'.(x', u', y') \in T'\}$ (for every state $x' \in X'$). Hence, the number of iterations of this loop is bounded by $\binom{|\kappa_{x'}|}{\lfloor |\kappa_{x'}|/2 \rfloor}$. The loop starting in line 15 is executed at most $|\kappa_{X'}|$ times per iteration of the outer loop.

Because the typically large number of actions and the number of iterations required in the algorithm, in the remainder of this section we present two modifications of the algorithm to reduce both of them while retaining the good properties of the abstraction.

A. Restricting the spreading of actions

In the first modification of Algorithm 1, we add a *two-stage filtering process* that reduces the set of actions with successors in T' to those that are most *steady*. In the first stage, we determine the maximum *spread* of the actions in (X, U, T). That is, we iterate over all $y_1, y_2 \in X$ for which there exist $u \in U$ and $x \in X$ with $(x, u, y_1) \in T$ and $(x, u, y_2) \in T$, and compute the spread of y_1 and y_2, which is formally defined as

$$\mathsf{spread}_X(y_1, y_2) = \left| \left\lfloor \frac{y_1}{(c_1, \ldots, c_n)} \right\rfloor - \left\lfloor \frac{y_2}{(c_1, \ldots, c_n)} \right\rfloor \right|,$$

where all operations (division, rounding, subtraction and absolute value taking) work element-wise on n-tuples. The maximum overall spread $maxSpread$ then is the element-wise maximum of the spread for each pair y_1, y_2, and it gives us an approximation of the unavoidable degree of non-determinism that (X, U, T) induces into (X', U', T'). As we are interested in actions in the steady abstraction that do not lead to unnecessary spread, for the first stage of the filtering process, we inject the following operation before line 13 of Algorithm 1:

$$\psi \leftarrow \psi \wedge \bigwedge_{y'_1, y'_2 \in X',\, |y'_1 - y'_2| \not\leq mSF \cdot maxSpread} (\neg v_{y'_1} \vee \neg v_{y'_2})$$

Here, $\leq$ is a component-wise comparison and $\cdot$ is the scalar multiplication. The constant mSF is user-provided and defines a limiting spreading factor. The smaller the user-provided factor is, the more successor state combinations are excluded. Note that mSF must always be ≥ 1 not to exclude all possible transitions, and in our experiments in the next section, we use $mSF = 1.5$. The added clauses reduce the number of iterations of the algorithm's main loop, as many action assignments a are excluded a-priori.

To further reduce the number of actions and as the second step of the filtering process, we post-process the abstraction (X', U', T') after the execution of Algorithm 1 by removing actions with a high degree of non-determinism that are not strictly necessary to reach any successor state. For every $x' \in X'$, we perform the following three steps:

1) First, we compute for every $u' \in U'$ with $T \cap (\{x'\} \times \{u'\} \times X') \neq \emptyset$ the spread of the action u' in x', which is defined as

$$\mathsf{spread}_{X'}(x', u') = \max_{\substack{y'_1, y'_2 \in X' . \{x'\} \times \\ \{u'\} \times \{y'_1, y'_2\} \subseteq T'}} \|y'_1 - y'_2\|_\infty,$$

where $\|\cdot\|_\infty$ denotes the Hamming distance. We order the actions u' by $\mathsf{spread}_{X'}(x', u')$.
2) Then, we iterate over all actions u' in ascending order of their $\mathsf{spread}_{X'}$ values and remove all actions u' for which all states y' with $(x', u', y') \in T'$ have already been found by an earlier action in the list, i.e., we have $(x', \hat{u}', y') \in T'$ for an earlier action $\hat{u}'$. The action $\hat{u}'$ can be different for every y' with $(x', u', y') \in T'$ in this checking step.

3) Finally, we restrict T' to only have actions u' for state x' that have not been filtered out in the previous step.

Note that this filtering process is sound as it only removes available actions for the controller, but never alters them.

B. Using the actions in the original abstraction as guidance

The second modification is guided by the available actions in the original abstraction and chooses assignments a that resemble the behavior of applying the same control input $u \in U$ from all states $x' \in X'$ that correspond to the same state $x \in X$ in the original abstraction. At the same time, we still reduce the set of successor states reachable under the steady action as much as possible, so we still search for actions a for which $\theta(x', a)$ is minimal (for every state x'). Figure 1 shows a simple example for this idea.

We implement the idea by introducing a third variable set $A = \{a_u \mid u \in U\}$. Then, after line 12 of Algorithm 1, we add the clause $\bigvee_{u \in U} a_u$ for ensuring that one action in the original abstraction is used as blueprint for the steady action. The further clauses $\bigwedge_{u \in U} \bigvee_{y' \in X', \nexists y \in X : (x,u,y) \in T \wedge y' = m(y)} (\neg a_u \vee \neg v_{y'})$ then prevent the selection of steady actions for which now all successors are reachable from all states in $m^{-1}(x')$ by the same action $u \in U$. Note that there is no need to enforce that at most one variable in A can have a $\mathbf{true}$ value, as whenever multiple variables have $\mathbf{true}$ values in an assignment, then we can just pick any of the encoded actions.

VI. Experiments

To evaluate the usefulness of our abstraction compression approach, we implemented it in a prototype tool that processes an alternating simulation abstraction of some system dynamics. We applied it to two case studies, one with a simple vehicle dynamics and another one with a simple moon lander dynamics.[2]

The computer used for the experiments has an Intel Core i5-5200U CPU (2.20 GHz) with 8 GB of RAM running Ubuntu Linux.

A. Simple Vehicle Example

The simple vehicle dynamics are of the form

$$\dot{x}^t = \begin{pmatrix} \sin(x_3) \\ \cos(x_3) \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ u_1 \end{pmatrix},$$

where $x^t = (x_1, x_2, x_3)^T$ is the state of the system at time t and $u^t = (u_1)^T$ is the control input at time t. In a state $(x_1, x_2, x_3)^T$, the components x_1 and x_2 denote the X- and Y-coordinates of the vehicle in a two-dimensional workspace, while x_3 represents the current heading. The controller is required to keep the position of the vehicle in the range $[0, 8]$ for both X- and Y-dimensions. The vehicle runs with a constant speed. The control input for changing the heading of the vehicle is restricted to $[\frac{-13}{24}\pi, \frac{13}{24}\pi]$.

[2]The implementation for the abstraction processor and our case studies can be found at `http://motesy.cs.uni-bremen.de/tools/abstractionProcessor`.

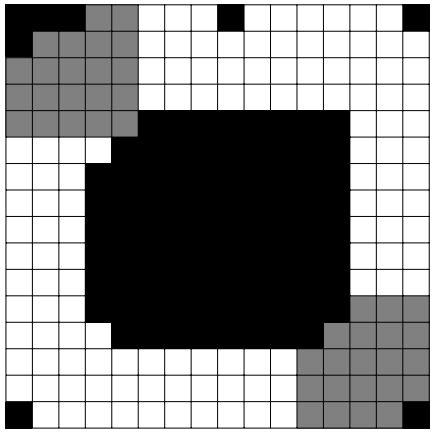


Fig. 2: Workspace of the vehicle example using a steady abstraction, where 93 of the 256 workspace cells were marked as obstacle cells.

We use the SCOTS framework [3] to compute a discrete-time system abstraction that gives rise to a feedback refinement relation between it and the concrete infinite-state system. We use a time step of $\tau = 1.0$, and discretize the workspace to 64 cells of size 0.125 units in each X- and Y-dimension and 128 intervals in the third (heading) dimension of size $\frac{\pi}{64}$. The input space is discretized to 13 intervals of size $\frac{\pi}{12}$.

Our implementation of the approach presented in the preceding section then processes this system abstraction to a smaller abstraction that alternatingly simulates the original system by compressing all dimensions by a factor of 4 each, which gives an abstraction with $D = 16 \times 16 \times 32$ states. As a comparison basis, we also consider

1) the relatively fine abstraction produced by SCOTS from which we compute the steady abstraction, as well as
2) the non-steady coarser abstraction that we obtain from SCOTS when asking directly for a discrete-state abstraction with D states.

Because the system dynamics are translation-invariant in the X- and Y-dimensions, our implementation executes the main loop of Algorithm 1 only once for every rotation value. The resulting transitions are then added to T' for all possible (x, y) coordinates as starting points, where we remove predecessor state/action combinations for which some of their transitions leave the workspace boundaries.

We store the abstraction as a *binary decision diagram*, which is a symbolic data structure that is capable of storing state and transition sets. The reactive synthesis tool `slugs` [23] is then used to synthesize controllers for the resulting abstraction. Both our prototype tool and `slugs` use the `CuDD` decision diagram library [24]. Our prototype tool furthermore uses `picosat` [25] version 965 or alternatively `lingeling` [26] in the `bbc-9230380-160707` satisfiability competition 2016 version as SAT solvers.

We used the computed system dynamics abstraction for synthesizing a controller that requires the system to react to

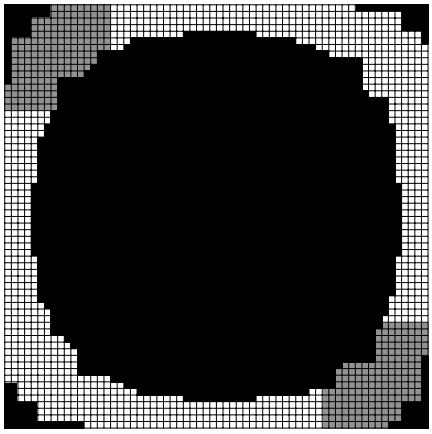


Fig. 3: Workspace of the vehicle example using a fine abstraction, where 2774 of the 4096 workspace cells were marked as obstacle cells.

a Boolean input signal: if the input signal eventually stays **false**, the vehicle has to infinitely often visit the lower right 5×5 cell region of the workspace, while if the input signal eventually stays **true**, the equally large upper left workspace region needs to be visited infinitely often. Since the controller does not know about the future evolution of the input signal, it has to eagerly move towards one of the two goal regions whenever the input signal value changes. Since the size of the relatively fine abstraction is 64×64, the target regions for its specification each have a size of 20×20.

The results of the experiments are shown in Table I. The specification is unrealizable when using the coarse abstraction of size D obtained directly from SCOTS. Computing the steady abstraction without any filtering was infeasible. When using one of the two modifications from Section V-A and Section V-B, computing a steady abstraction was much faster and the specification was realizable in both cases. Since the modified steady abstractions only use actions that are also present in the steady abstraction computed with any of the modifications from Section V-A and Section V-B, the same specification has to be realizable for the standard steady abstraction.

To determine the precision of the steady abstraction, we performed another experiment in which we gradually marked as many abstraction cells as static obstacles as possible without changing realizability, starting from the cells in the middle of the abstraction and ending in the corners. Figure 2 shows the resulting workspace (using the steady abstraction computed with the modification from Section V-A). All 256 realizability checks for this process together took 46 minutes. As it can be seen in Figure 2, a large obstacle (and various smaller ones) in the middle of the workspace can be tolerated without making the specification unrealizable. Note that our approach does not guarantee that an abstraction is symmetric, which explains the asymmetry of the figure. Figure 3 shows the resulting workspace after the same process when using the fine abstraction from which we computed the steady abstraction. As expected, more static obstacles can be added

TABLE I: Results of the experiments for the vehicle example using the SAT solver `picosat`. T.A. is the computation time for obtaining the abstraction, T.C. the time needed to check the realizability of the specification under the abstraction, D.N.D. is the degree of non-determinism according to Definition 1.

Abstraction Type	T.A.	T.C.	D.N.D
Coarse Abstraction	1.826s	6.546s	18
Fine Abstraction	4m4.758s	9m18.450s	24
Steady Abstraction	timeout $> 24h$	-	-
Steady Abstraction + Ext. A	4m41.454s	9.149s	12
Steady Abstraction + Ext. B	4m10.000s	8.815s	12

before the system becomes unrealizable. This is because 1) the actions in the steady abstraction are based on those available in the fine abstraction, and 2) if *any* state in $m^{-1}(x')$ for some state x' of the steady abstraction is a bad state, then x' is also automatically a bad state.

To evaluate the effect of the choice of SAT solver, we also computed the steady abstractions with `lingeling` as backend solver. The computation times (using the extensions from Section V-A and Section V-B) slightly increased to 5m42.872s and very slighty decreased to 4m6.737s respectively. We can see that the individual SAT queries given to the SAT solver are not difficult enough to benefit from `lingelings`'s more advanced preprocessing schemes.

B. Simple Moon Lander Example

For the second case study, we consider the dynamics of a moon lander:

$$\dot{x}^t = \begin{pmatrix} x_1 \\ x_2 \\ 1.0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ u_1 \\ u_2 \end{pmatrix},$$

where $x^t = (x_1, x_2, x_3, x_4)^T$ is the state of the system at time t and $u^t = (u_1, u_2)^T$ is the control input at time t. In a state $(x_1, x_2, x_3, x_4)^T$, the components x_1 and x_2 denote the Y- and X-coordinates of the lander in a two-dimensional workspace, while x_3 represents the current speed along the Y-axis and x_4 denotes the current speed along the X-axis. The control input components are restricted to $[-2, 0]$ and $[-1, 1]$, respectively, and represent the acceleration in the Y- and X-dimensions. The controller is required to keep the position of the lander in the range $[0, 10]$ along the Y-axis and $[0, 4.5]$ along the X-axis. The speed needs to be kept in the range $[-2.0625, 2.0625]$ in the Y-dimension and $[-1.0625, 1.0625]$ in the X-dimension.

Again we use the SCOTS framework [3] to compute an abstraction to be processed. We use a time step of $\tau = 1.0$ and discretize the workspace into 20 cells of size 0.5 units in the Y-dimension and 9 cells of size 0.5 in the X-dimension. There are 32 intervals in the third (Y-speed) dimension of size 0.125 and only 16 intervals in the X-speed dimension of size 0.125. The control input space is discretized to 33 intervals for the Y-acceleration and 16 for the X-dimension. For the steady abstractions, the compression factor is 2 for

TABLE II: Results of the experiments for the moon lander example. All labels are the same as in Table I.

Abstraction Type	T.A.	T.C.	D.N.D
Coarse Abstraction	1m8.552s	0m7.302s	24
Fine Abstraction	8m11.066s	57m28.550s	36
Steady Abstraction	timeout $> 24h$	-	-
Steady Abstraction + Ext. A	25m22.213s	3m53.306s	12
Steady Abstraction + Ext. B	11m28.337s	3m35.881s	12

the Y-axis position and 3 for the the Y-speed dimension. The other dimensions are not compressed. Overall, we get a steady abstraction with $D = 10 \times 9 \times 11 \times 16$ states.

In the moon lander dynamics, the X- and Y-position dimensions also are translation-invariant, which again enables us to execute the main loop of Algorithm 1 only once for every possible combination of the X-speed and Y-speed values. For the specification, we require the lander to visit two goal regions (above the ground) while crossing the middle of the workspace at a height of at least 5, and finally landing on the lower workspace boundary.

The results of the experiments are shown in Table II. The specification is unrealizable when directly starting with a coarse abstraction of size D computed by SCOTS. The specification is realizable in all other cases.

VII. Conclusion

We have presented a novel method to postprocess cyber-physical system abstractions in order to reduce their sizes while retaining their controllability to a large extent. Our approach helps to combine good controllability in alternating simulation abstractions with a small abstraction size, which keeps the computation times of the synthesis process for which the abstractions are used short.

Our experiments show that the approach indeed computes abstractions that are "steady" – the system is able to avoid obstacles and to visit relatively small target regions, while the synthesis process is much faster than when using a more fine-grained abstraction. Due to these observations, we think that this method is a useful component in future approaches for CPS controller synthesis.

Acknowledgements

The Authors would like to thank Matthias Rungger for extensive help with the SCOTS framework.

This work was partially funded by the Institutional Strategy of the University of Bremen, funded by the German Excellence Initiative.

References

[1] Tabuada, P.: Verification and Control of Hybrid Systems - A Symbolic Approach. Springer (2009)

[2] Alur, R., Henzinger, T.A., Kupferman, O., Vardi, M.Y.: Alternating refinement relations. In: CONCUR '98: Concurrency Theory, 9th International Conference, 1998. (1998) 163–178

[3] Rungger, M., Zamani, M.: SCOTS: A tool for the synthesis of symbolic controllers. In: 19th International Conference on Hybrid Systems: Computation and Control, HSCC 2016. (2016) 99–104

[4] Jr., M.M., Davitian, A., Tabuada, P.: PESSOA: A tool for embedded controller synthesis. In: Computer Aided Verification, 22nd International Conference, CAV 2010. (2010) 566–569

[5] Mattila, R., Mo, Y., Murray, R.M.: An iterative abstraction algorithm for reactive correct-by-construction controller synthesis. In: 54th IEEE Conference on Decision and Control, CDC 2015. (2015) 6147–6152

[6] Fu, J., Dimitrova, R., Topcu, U.: Abstractions and sensor design in partial-information, reactive controller synthesis. In: American Control Conference, ACC 2014. (2014) 2297–2304

[7] Rungger, M., Stursberg, O.: On-the-fly model abstraction for controller synthesis. In: American Control Conference, ACC 2012. (2012) 2645–2650

[8] Cámara, J., Girard, A., Gößler, G.: Synthesis of switching controllers using approximately bisimilar multiscale abstractions. In: 14th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2011. (2011) 191–200

[9] Hahn, E.M., Norman, G., Parker, D., Wachter, B., Zhang, L.: Game-based abstraction and controller synthesis for probabilistic hybrid systems. In: Eighth International Conference on Quantitative Evaluation of Systems, QEST 2011. (2011) 69–78

[10] Girard, A.: Synthesis using approximately bisimilar abstractions: state-feedback controllers for safety specifications. In: 13th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2010. (2010) 111–120

[11] Mouelhi, S., Girard, A., Gößler, G.: Cosyma: a tool for controller synthesis using multi-scale abstractions. In: 16th International Conference on Hybrid Systems: Computation and Control, HSCC 2013. (2013) 83–88

[12] Alimguzhin, V., Mari, F., Melatti, I., Salvo, I., Tronci, E.: Linearising discrete time hybrid systems. IEEE Transactions on Automatic Control **62**(99) (2017) 5357–5364

[13] Tabuada, P.: Controller synthesis for bisimulation equivalence. Systems & Control Letters **57**(6) (2008) 443–452

[14] Glück, R., Möller, B., Sintzoff, M.: A semiring approach to equivalences, bisimulations and control. In: Relations and Kleene Algebra in Computer Science, 11th International Conference on Relational Methods in Computer Science, RelMiCS 2009, and 6th International Conference on Applications of Kleene Algebra, AKA 2009. (2009) 134–149

[15] Pola, G., Girard, A., Tabuada, P.: Approximately bisimilar symbolic models for nonlinear control systems. Automatica **44**(10) (2008) 2508–2516

[16] Girard, A., Pappas, G.J.: Approximation metrics for discrete and continuous systems. IEEE Trans. Automat. Contr. **52**(5) (2007) 782–798

[17] Pola, G., Tabuada, P.: Symbolic models for nonlinear control systems: Alternating approximate bisimulations. SIAM J. Control and Optimization **48**(2) (2009) 719–733

[18] Reissig, G., Weber, A., Rungger, M.: Feedback refinement relations for the synthesis of symbolic controllers. IEEE Trans. Automat. Contr. **62**(4) (2017) 1781–1796

[19] Farwer, B.: ω-automata. In: Automata, Logics, and Infinite Games: A Guide to Current Research, 2001. (2001) 3–20

[20] Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive(1) designs. In: VMCAI. (2006) 364–380

[21] Biere, A., Heule, M., van Maaren, H., Walsh, T., eds.: Handbook of Satisfiability. Volume 185 of Frontiers in Artificial Intelligence and Applications. IOS Press (2009)

[22] Eén, N., Sörensson, N.: An extensible SAT-solver. In: Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. (2003) 502–518

[23] Ehlers, R., Raman, V.: Slugs: Extensible GR(1) synthesis. In: Computer Aided Verification - 28th International Conference, CAV 2016. (2016) 333–339

[24] Somenzi, F.: CUDD: CU Decision Diagram package release 3.0.0 (2016)

[25] Biere, A.: Picosat essentials. Journal on Satisfiability, Boolean Modeling and Computation (JSAT) (2008)

[26] Biere, A.: Lingeling, Plingeling and Treengeling entering the SAT competition 2013. In Balint, A., Belov, A., Heule, M., Järvisalo, M., eds.: SAT Competition 2013. vol. B-2013-1 of Department of Computer Science Series of Publications B, University of Helsinki (2013) 51–52