Divergent Stutter Bisimulation Abstraction for Controller Synthesis with Linear Temporal Logic Specifications

Sahar Mohajerani [a] Robi Malik [b] Andrew Wintenberg [c] Stéphane Lafortune [c] Necmiye Ozay [c]

[a] *Department of Electrical Engineering, Chalmers University of Technology, Göteborg, Sweden*

[b] *Department of Software Engineering, University of Waikato, Hamilton, New Zealand*

[c] *Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA*

Abstract

This paper proposes a method to synthesise controllers for systems with possibly infinite number of states that satisfy a specification given as an $\mathrm{LTL}_{\setminus\circ}$ formula. A common approach to handle this problem is to first compute a finite-state abstraction of the original state space and then synthesise a controller for the abstraction. This paper proposes to use an abstraction method called *divergent stutter bisimulation* to abstract the state space of the system. As divergent stutter bisimulation factors out *stuttering steps*, it typically results in a coarser and therefore smaller abstraction, at the expense of not preserving the temporal "next" operator. The paper leverages results about divergent stutter bisimulation from model checking and shows that divergent stutter bisimulation is a sound and complete abstraction method when synthesising controllers subject to specifications in $\mathrm{LTL}_{\setminus\circ}$.

Key words: Control synthesis; Computational issues; Controller constraints and structure; Abstraction; LTL specification.

1 Introduction

Dynamic system models are often used for safety-critical applications, where formal verification and synthesis are of great importance (Belta et al., 2017). This paper is concerned with *controller synthesis*, where the control logic is automatically computed from a model of the system with possibly infinite number of states and a specification of the desired behaviour in temporal logic. In this context, the state space is typically abstracted and partitioned to produce a finite transition system, and then finite-state machine synthesis methods are applied (Ramadge, 1989; Kloetzer and Belta, 2008).

The most commonly used abstraction method for this purpose is *bisimulation* (Milner, 1989). Bisimulation is a strong behavioural equivalence of transition systems, which preserves all temporal logic properties (Baier and Katoen, 2008). There exist polynomial-time algorithms to calculate a bisimilar abstraction of a finite-state system (Fernandez, 1990). Bisimulation can also be applied to continuous state spaces (Pappas, 2003; Belta et al., 2017; Megawati and van der Schaft, 2016), but the bisimulation algorithms are only guaranteed to terminate for specific classes of continuous systems (Alur et al., 2000).

To overcome these difficulties, the literature proposes various alternatives to bisimulation. *Approximate bisimulation* relaxes bisimulation by allowing a bounded mismatch between the behaviours of the abstract and concrete system (Girard and Pappas, 2007). A further relaxation is obtained by considering (approximate) simulation and feedback refinement relations. These are sound

⋆ The work of the first author was supported by the Swedish Research Council under grants 2016-00529 and 2016-06204, while working as postdoc at the University of Michigan. The work of the third author was supported in part by US NSF grant CNS-1738103 and an Early Career Faculty grant from NASA's Space Technology Research Grants Program. The research of the fourth author was supported in part by US NSF grant CNS-1738103 and the work of fifth author was supported by an Early Career Faculty grant from NASA's Space Technology Research Grants Program..

Email addresses: `mohajera@chalmers.se` (Sahar Mohajerani), `robi@waikato.ac.nz` (Robi Malik), `awintenb@umich.edu` (Andrew Wintenberg), `stephane@umich.edu` (Stéphane Lafortune), `necmiye@umich.edu` (Necmiye Ozay).

but not necessarily complete abstraction methods, in the sense that the nonexistence of a controller for the abstract system does not imply the nonexistence of a controller for the concrete system (Tabuada, 2009; Zamani et al., 2011; Belta et al., 2017; Reissig et al., 2016). *Dual-simulation* (Wagenmaker and Ozay, 2016) produces a coarser abstraction than bisimulation; it uses overlapping subsets rather than quotient sets. Moreover, the dual-simulation algorithm avoids the set difference operation, so that it preserves the convexity of the regions in the abstracted state space. Unlike bisimulation, it does not preserve all temporal logic properties, but it preserves the results of controller synthesis when applied to linear temporal logic (LTL).

Bisimulation and its variants consider all transitions as significant. The potential for abstraction can be increased by factoring out so-called *stuttering steps* where the system remains in the same region of the state space without changing any of the propositions relevant for the specification. By combining sequences of stuttering steps with the next non-stuttering step in a single transition, a coarser abstraction is produced. However, the temporal "next" operator is no longer preserved because the number of steps between two states may change. Accordingly, *over-approximations* (Liu et al., 2013; Nilsson et al., 2017) have been proposed to synthesise controllers for continuous-time systems. The approach is sound for specifications in $\mathrm{LTL}_{\setminus\circ}$, the LTL fragment without the "next" operator, but not complete in general.

This paper considers another approach to factor out stuttering steps, called *divergent stutter bisimulation* (Baier and Katoen, 2008). Divergent stutter bisimulation preserves $\mathrm{CTL}^*_{\setminus\circ}$, the fragment of the Computation Tree Logic CTL^* without the "next" operator (Baier and Katoen, 2008). An efficient algorithm to compute abstractions based on divergent stutter bisimulation exists and has been used to simplify state spaces (Groote and Vaandrager, 1990; Groote et al., 2017). Once an abstraction and a quotient system are constructed using divergent stutter bisimulation, *verification* can be done using existing tools (Clarke et al., 1999; Baier and Katoen, 2008) and gives the same result as for the original system. Yet, *synthesis* is more difficult, because it is not immediately clear how a controller synthesised for the abstracted system can be used to control the original system. A single transition of the abstract controller has to be implemented by a sequence of several transitions (including stuttering steps) in the original system.

This paper shows that *divergent stutter bisimulation* is a sound and complete abstraction method when synthesising for specifications in $\mathrm{LTL}_{\setminus\circ}$. It is shown that a controller for the abstracted system exists if and only if such a controller exists for the original system with possibly infinite number of states, and it is shown how the controller for the original system can be constructed from the abstract controller.

The divergent stutter bisimulation algorithm and the algorithm to construct a controller for the original system from the abstract controller have been implemented in TuLiP (Filippidis et al., 2016) and applied to discrete-time linear systems from (Hussien and Tabuada, 2018; Wagenmaker and Ozay, 2016). It is shown that the abstracted system from divergent stutter bisimulation is smaller compared to bisimulation. As expected, lifting the control from the discrete abstraction to the original system is more laborious than it would be using only bisimulation.

This paper is organised as follows. Section 2 gives a brief background on modelling and linear temporal logic and abstraction. Next, Section 3 explains divergent stutter bisimulation and reviews the algorithm to partition a state space while preserving divergent stutter bisimulation. Section 4 shows how the abstracted controller is used to construct a controller for the original system and proves that the abstraction method is sound and complete. Section 5 applies the algorithm to examples, and Section 6 gives concluding remarks.

2 Preliminaries

This section gives a brief overview of notations used throughout the paper. Most of the following definitions are adopted from Baier and Katoen (2008).

2.1 Finite and Infinite Strings

Let X be a set. The sets of finite and infinite strings of symbols from X are denoted by X^* and X^ω, respectively. The combined set of finite and infinite strings over X is $X^\infty = X^* \cup X^\omega$. The *empty string* is $\varepsilon \in X^*$, and the set of nonempty finite strings is $X^+ = X^* \setminus \{\varepsilon\}$. The *concatenation* of strings $s \in X^*$ and $t \in X^\infty$ is written as st. The notation s^k refers to the string obtained by concatenating $k \geq 0$ copies of string $s \in X^*$. A string $s \in X^*$ is called a *prefix* of $t \in X^\infty$, written $s \sqsubseteq t$, if there exists $u \in X^\infty$ such that $su = t$.

An infinite ascending sequence $s_0 \sqsubseteq s_1 \sqsubseteq s_2 \sqsubseteq \cdots$ of finite strings $s_i \in X^*$ has a unique least upper bound $s \in X^\infty$ with $s_i \sqsubseteq s$ for all $i \geq 0$, called its *closure* and denoted by $\mathrm{clo}\{\, s_i \mid i \geq 0 \,\}$. For $s \in X^\infty$, the *duplicate-free string* $\mathrm{uniq}(s) \in X^\infty$ is obtained from s by removing all elements that are equal to their predecessor. For example, $\mathrm{uniq}(001322233\cdots) = 01323$. Removing duplicates from an infinite string may result in a finite or infinite string.

2.2 Transition Systems and Problem Formulation

Definition 1. A *transition system* is a tuple $G = \langle Q, Q^\circ, \rightarrow, \Pi, \models\rangle$ where Q is a set of states, $Q^\circ \subseteq Q$ is the set of initial states, $\rightarrow \subseteq Q \times Q$ is the *state transition* *relation*, Π is a set of atomic propositions, and $\models \subseteq Q \times \Pi$ is the satisfaction relation. G is called *finite* if Q is finite.

Definition 2. A finite path fragment s in a transition system $G = \langle Q, Q^\circ, \rightarrow, \Pi, \models \rangle$ is a finite state sequence $x_0 \cdots x_n$ such that there are transitions $x_{i-1} \rightarrow x_i$ for all $0 < i \leq n$. An infinite path fragment s is an infinite state sequence $x_0 x_1 \cdots$ such that $x_{i-1} \rightarrow x_i$ for all $0 < i$. If $x_0 \in Q^\circ$ then the path fragment is called a *path* in G.

A path in a transition system describes a possible behaviour of the system.

Definition 3. Let $G = \langle Q, Q^\circ, \rightarrow, \Pi, \models \rangle$ be a transition system. The labelling function $L\colon Q \rightarrow 2^\Pi$ is defined by $L(x) = \{\, \pi \in \Pi \mid x \models \pi \,\}$.

The labelling function L relates to each state $x \in Q$ a set of atomic proposition that are satisfied by state x. The labelling function L is extended to finite or infinite path fragments $s = x_0 x_1 \cdots \in Q^\infty$ by applying it to each state of the path fragment, $L(s) = L(x_0)L(x_1)\cdots$.

This paper is concerned with the construction of *controllers* that restrict a transition system so that it only enters certain states.

Definition 4 (Controller). Let $G = \langle Q, Q^\circ, \rightarrow, \Pi, \models \rangle$ be a transition system. A controller for G is a function $C\colon Q^* \rightarrow 2^Q$.

The controller takes as argument a path, representing the history of all states visited in the path. If $x_0, \ldots, x_n \in Q$ are states ($n \geq 0$), then the path composed of these states is written $x_0 \cdots x_n \in Q^*$. The idea is that, after visiting states $x_0, \ldots, x_n \in Q$, the controlled system is allowed to enter a state $x_{n+1} \in Q$ if and only if $x_{n+1} \in C(x_0 \cdots x_n)$.

Definition 5. Let $G = \langle Q, Q^\circ, \rightarrow, \Pi, \models \rangle$ be a transition system, and let $C\colon Q^* \rightarrow 2^Q$ be a controller for G. The controlled system, G under the control of C, is $C/G = \langle Q^+, Q^\circ \cap C(\varepsilon), \rightarrow_{|C}, \Pi, \models_{|C} \rangle$, where

(i) $x_0 \cdots x_n \rightarrow_{|C} x_0 \cdots x_n x_{n+1}$ if and only if $x_n \rightarrow x_{n+1}$ in G and $x_{n+1} \in C(x_1 \cdots x_n)$,
(ii) $x_0 \cdots x_n \models_{|C} \pi$ if and only if $x_n \models \pi$.

In Def. 5, the states of the closed-loop system, C/G, are paths of G that are accepted by C. The transitions of the closed-loop system have the from of $x_1 \cdots x_n \rightarrow_{|C} x_1 \cdots x_{n+1}$, where $x_1 \cdots x_n$ is path of G and a state of the controlled system C/G. From a state $x_1 \cdots x_n$ of C/G, the next state $x_1 \cdots x_{n+1}$ can be reached if $x_n \rightarrow x_{n+1}$ is a transition in G and $x_{n+1} \in C(x_1 \cdots x_n)$.

Accordingly, the paths in C/G are strings of the states of C/G, i.e., strings of strings. The following definition adds a more convenient notation to also consider strings over states of G as paths in C/G.

Definition 6. Let $G = \langle Q, Q^\circ, \rightarrow, \Pi, \models \rangle$, and let C be a controller for G. A string $s \in Q^\infty$ is said to be a *path in C/G* if s is a path in G, and for every prefix $rx \sqsubseteq s$ with $r \in Q^*$ and $x \in Q$ it holds that $x \in C(r)$.

Another important concern in the following is *divergence*, where a transition system stays in the same state indefinitely.

Definition 7. Let $G = \langle Q, Q^\circ, \rightarrow, \Pi, \models \rangle$, and let C be a controller for G. A state $x \in Q$ is *divergent* in G if $x \rightarrow x$. A finite path $s = rx$ with $r \in Q^*$ and $x \in Q$ is divergent in C/G if its last state x is divergent in G and $x \in C(sx^k)$ for all $k \geq 0$.

2.3 Linear Temporal Logic

This paper considers requirement specifications written in *Linear Temporal Logic* (Baier and Katoen, 2008). Specifically, the fragment considered is $\mathrm{LTL}_{\setminus\circ}$, which does not include the *next* operator $\circ$. $\mathrm{LTL}_{\setminus\circ}$ formulas φ over the set of atomic propositions Π are formed according to the syntax $\varphi ::= \pi \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi_1 \mid \varphi_1 \,\mathcal{U}\, \varphi_2$, where $\pi \in \Pi$, and φ_1 and φ_2 are $\mathrm{LTL}_{\setminus\circ}$ formulas. $\mathcal{U}$ is the *until* operator, which is also used to define the more common operators *finally*, $\Diamond\, \varphi = \text{true} \,\mathcal{U}\, \varphi$ and *always*, $\Box\, \varphi = \neg \Diamond \neg\varphi$.

Definition 8. Let $G = \langle Q, Q^\circ, \rightarrow, \Pi, \models \rangle$ be a transition system and let φ be an $\mathrm{LTL}_{\setminus\circ}$ formula over Π. An infinite path fragment $s \in Q^\infty$ is defined to *satisfy* φ, written $s \models \varphi$, recursively as follows:

- $s \models \pi$ if $s = x_0 x_1 \cdots$ and $x_0 \models \pi$;
- $s \models \varphi_1 \wedge \varphi_2$ if both $s \models \varphi_1$ and $s \models \varphi_2$;
- $s \models \neg\varphi_1$ if $s \models \varphi_1$ does not hold;
- $s \models \varphi_1 \,\mathcal{U}\, \varphi_2$ if $s = x_0 x_1 \cdots$ and there exists $k \geq 0$ such that $x_i x_{i+1} \cdots \models \varphi_1$ for $i = 0, \ldots, k-1$ and $x_k x_{k+1} \cdots \models \varphi_2$.

The transition system G is said to satisfy φ, written $G \models \varphi$, if $s \models \varphi$ holds for every infinite path s in G.

According to this semantics, a transition system satisfies an $\mathrm{LTL}_{\setminus\circ}$ formula if the formula holds on every infinite path. The definition does not cover paths that visit a deadlock state, i.e., a state without outgoing transitions. It is common to rule out this case so that all paths can be extended to an infinite path.

Definition 9. Let $G = \langle Q, Q^\circ, \rightarrow, \Pi, \models \rangle$ be a transition system. Then G is called *deadlock free* if, $Q^\circ \neq \emptyset$ and for every reachable state $x \in Q$ there exists a state $y \in Q$ such that $x \rightarrow y$.

This paper is concerned with the *synthesis* problem, which is to compute a controller enforcing a specification given as an LTL$_{\setminus\circ}$ formula for a given transition system. This problem can now be defined formally as follows.

Problem 1. Given a transition system G and an LTL$_{\setminus\circ}$ formula φ, find a controller C such that C/G is deadlock-free and $C/G \models \varphi$.

There are several methods to construct a controller that enforces an LTL$_{\setminus\circ}$ formula on a finite transition system. This can be done by transforming the LTL$_{\setminus\circ}$ formula to a Rabin or a Büchi automaton, taking a product of the automaton with the transition system, and then solving the control problem using existing methods (Ramadge, 1989; Kloetzer and Belta, 2008).

2.4 Abstraction

For systems with very large or infinite state spaces, finding a solution to Problem 1 becomes intractable. One approach to handle the synthesis problem in such cases is the use of *abstraction techniques*, where the state space is partitioned to produce an equivalent synthesis problem that can be solved more easily.

A common way to partition a transition system is to identify and group equivalent states. Given a set X, a relation $\approx\ \subseteq X \times X$ is an *equivalence relation* on X if it is reflexive, symmetric, and transitive. The *equivalence class* of $x \in X$ is $[x] = \{\, x' \in X \mid x \approx x' \,\}$, and $X/{\approx} = \{\, [x] \mid x \in X \,\}$ is the set of all equivalence classes modulo $\approx$.

Given an equivalence relation $\approx$ on the state set Q of a transition system, a *quotient* transition system can be constructed by grouping all equivalent states into a single abstract state, i.e., using $Q/{\approx}$ as a reduced state set. For the quotient transition system to preserve liveness properties, it is important to distinguish whether or not it is possible for the system behaviour to remain within an equivalence class indefinitely.

Definition 10. Let $G = \langle Q, Q^\circ, \rightarrow, \Pi, \models \rangle$ be a transition system, and let $\approx\ \subseteq Q \times Q$ be an equivalence relation. A state $x \in Q$ is *$\approx$-divergent* in G if there exists an infinite path fragment $x x_1 x_2 \cdots$ in G such that $x \approx x_i$ for all $i > 0$. An equivalence class[1] $\tilde{x} \in Q/{\approx}$ is $\approx$-divergent in G if it contains a $\approx$-divergent state.

Definition 11. Let $G = \langle Q, Q^\circ, \rightarrow, \Pi, \models \rangle$ be a transition system, and let $\approx\ \subseteq Q \times Q$ be an equivalence relation. The *divergence-respecting quotient transition system* is $G /\!\!/ {\approx} = \langle \tilde{Q}, \tilde{Q}^\circ, \tilde{\rightarrow}, \Pi, \tilde{\models} \rangle$ where,

- $\tilde{Q} = Q/{\approx}$;
- $\tilde{Q}^\circ = \{\, \tilde{x}^\circ \in \tilde{Q} \mid \tilde{x}^\circ \cap Q^\circ \neq \emptyset \,\}$;
- $\tilde{x} \mathrel{\tilde{\rightarrow}} \tilde{x}$ if $\tilde{x}$ is divergent;
- $\tilde{x} \mathrel{\tilde{\rightarrow}} \tilde{y}$ for $\tilde{x} \neq \tilde{y}$, if there exist $x \in \tilde{x}$ and $y \in \tilde{y}$ such that $x \rightarrow y$;
- $\tilde{x} \mathrel{\tilde{\models}} \pi$ if there exists $x \in \tilde{x}$ such that $x \models \pi$.

Like a standard quotient transition system, the divergence-respecting quotient includes a transition $\tilde{x} \mathrel{\tilde{\rightarrow}} \tilde{y}$ between two equivalence classes if the original transition system has a transition $x \rightarrow y$ between some states of these classes, and an equivalence class satisfies a proposition if one of its states satisfies that proposition. The difference to a standard quotient lies in the treatment of selfloop transitions: a selfloop $\tilde{x} \mathrel{\tilde{\rightarrow}} \tilde{x}$ is only included in the divergence-respecting quotient if $\tilde{x}$ is a divergent equivalence class.

For simplicity of notation, in the following, the tilde superscripts of the transition and satisfaction relations of quotient transition systems will be omitted, thus identifying $\tilde{\rightarrow}$ with $\rightarrow$ and $\tilde{\models}$ with $\models$. Further, the following definition is used to relate path fragments of a transition system to those of its quotient.

Definition 12. Let $G = \langle Q, Q^\circ, \rightarrow, \Pi, \models \rangle$ be a transition system, and let $\approx\ \subseteq Q \times Q$ be an equivalence relation. For a (finite or infinite) path fragment $s = x_0 x_1 x_2 \cdots \in Q^\infty$, the term $[s] = [x_0][x_1][x_2] \cdots \in (Q/{\approx})^\infty$ denotes the path fragment of equivalence classes that appear in s.

3 Stutter Equivalence

This paper addresses the problem of abstracting a transition system before calculating a controller to enforce an LTL$_{\setminus\circ}$ specification. Without the temporal "next" operator, the logic LTL$_{\setminus\circ}$ cannot distinguish how often states with the same labels are repeated. This leads to the idea of *stutter equivalence* (Baier and Katoen, 2008). Transitions between states with equal labels are called *stuttering steps*, and *stutter bisimulation* considers transition systems as equivalent if they have the same infinite paths while factoring out the stuttering steps. To preserve liveness properties, this relation is refined to *divergent stutter bisimulation* (Baier and Katoen, 2008).

3.1 Stutter Equivalent Paths

Definition 13. Let $G_i = \langle Q_i, Q_i^\circ, \rightarrow_i, \Pi, \models_i \rangle$ be transition systems, and let s_i be path fragments in G_i, for $i = 1, 2$. Then s_1 and s_2 are *stutter equivalent* if $\mathrm{uniq}(\mathrm{L}(s_1)) = \mathrm{uniq}(\mathrm{L}(s_2))$ and s_1 and s_2 are either both finite or both infinite.

According to Def. 3, $\mathrm{L}(s)$ is the sequence of labellings visited on the path fragment s, and $\mathrm{uniq}(\mathrm{L}(s))$ is the

[1] Here and in the following, $\tilde{x} \in Q/{\approx}$ is an unspecified equivalence class, which may or may not be equal to the equivalence class $[x]$ containing x.

same sequence after removing duplicates. Therefore, stutter equivalent paths have the same sequences of atomic propositions, while the number of states repeated by stuttering steps, i.e., transitions whose source and target states satisfy exactly the same propositions, does not need to be equal. It is known that stutter equivalent paths satisfy the same $\mathrm{LTL}_{\setminus\circ}$ formulas.

Proposition 1. (Baier and Katoen, 2008) Let $G_i = \langle Q_i, Q_i^\circ, \to_i, \Pi, \models_i\rangle$ be transition systems, and let $s_i \in Q_i^\omega$ be path fragments in G_i, for $i = 1, 2$. If s_1 and s_2 are stutter equivalent, then $s_1 \models \varphi$ if and only if $s_2 \models \varphi$ for any $\mathrm{LTL}_{\setminus\circ}$ formula φ.

3.2 Divergent Stutter Bisimulation

Definition 14. Let $G = \langle Q, Q^\circ, \to, \Pi, \models\rangle$ be a transition system. A relation $\approx\, \subseteq Q \times Q$ is a *divergent stutter bisimulation* equivalence on G if for all $x_1, x_2 \in Q$ such that $x_1 \approx x_2$ the following conditions hold:

(i) $\mathrm{L}(x_1) = \mathrm{L}(x_2)$,
(ii) if there exists $x_1' \in Q$ such that $x_1 \to x_1'$ and $x_1' \not\approx x_2$ then there exists a finite path fragment $x_2 y_1 \dots y_n x_2'$ such that $n \geq 0$ and $x_1 \approx y_i$ for $i = 1, \dots, n$ and $x_1' \approx x_2'$;
(iii) if there exists $x_2' \in Q$ such that $x_2 \to x_2'$ and $x_2' \not\approx x_1$ then there exists a finite path fragment $x_1 z_1 \dots z_n x_1'$ such that $n \geq 0$ and $x_2 \approx z_i$ for $i = 1, \dots, n$ and $x_2' \approx x_1'$;
(iv) x_1 is $\approx$-divergent if and only if x_2 is $\approx$-divergent.

Condition (i) requires that equivalent states are labelled with the same propositions, i.e., the equivalence relation is proposition-preserving. According to conditions (ii) and (iii), for two states to be considered as equivalent, if one of them reaches a state in a different equivalence class then the other must reach an equivalent state either directly or after some stuttering steps. The last condition (iv) means that two states are only equivalent if either both exhibit divergent paths or none of them does.

Theorem 2. (Baier and Katoen, 2008) Let $G = \langle Q, Q^\circ, \to, \Pi, \models\rangle$ be a transition system, and let $\approx$ be a divergent stutter bisimulation on G. If $s \in Q^\omega$ is a path in G then there exists a stutter equivalent infinite path $\tilde{s}$ in $G/\!\!/\!\approx$.

Theorem 2 shows that, if a transition system G is abstracted to a divergence-respecting quotient $G/\!\!/\!\approx$ with respect to a divergent stutter bisimulation, then for any infinite path in the original system G there exists a stutter equivalent infinite path in the abstraction $G/\!\!/\!\approx$. By Prop. 1, these paths satisfy the same $\mathrm{LTL}_{\setminus\circ}$ properties.

3.3 Divergent Stutter Bisimulation Algorithm

There exists a polynomial-time algorithm to calculate the coarsest divergent stutter bisimulation relation of a transition system (Groote and Vaandrager, 1990; Groote et al., 2017), which is summarised in this section. The algorithm performs *partition refinement* starting from an initial partition consisting of regions determined by the propositions. Regions are split repeatedly when they contain states that cannot reach the same successors. The splits are performed by calculating the sets of *predecessors* of a region and then intersecting all regions with this predecessor set.

Definition 15. Let $G = \langle Q, Q^\circ, \to, \Pi, \models\rangle$ be a transition system. The set of predecessors of state $x \in Q$ is $\mathrm{Pre}(x) = \{\, y \in Q \mid y \to x \,\}$. The set of predecessors of a set of states $X \subseteq Q$ is $\mathrm{Pre}(X) = \bigcup_{x \in X} \mathrm{Pre}(x)$. The set of *stutter predecessors* of a state set $T \subseteq Q$ within P is

$$\mathrm{PPre}(T, P) = \{\, y \in P \mid \text{there exists a finite path fragment } y y_1 \cdots y_n x \text{ in } G \text{ with } n \geq 0 \text{ and } y_1, \dots, y_n \in P \text{ and } x \in T \,\}\ . \tag{1}$$

The set of *divergent states* within P is

$$\mathrm{Div}(P) = \{\, y \in P \mid \text{there exists an infinite path fragment } y y_1 y_2 \cdots \text{ in } G \text{ such that } y_k \in P \text{ for all } k \geq 1 \,\}\ . \tag{2}$$

The stutter predecessors of T within P are states in the region P that can reach a state in T directly or after a finite number of steps within P. The divergent states within P are states from where an infinite number of transitions is possible while staying within P. The sets of stutter predecessors and divergent states can be characterised as fixed points. Given the recursive definition

$$\mathrm{PPre}^0(T, P) = \mathrm{Pre}(T) \cap P\ ; \tag{3}$$
$$\mathrm{PPre}^{i+1}(T, P) = \mathrm{Pre}(\mathrm{PPre}^i(T, P)) \cap P\ ; \tag{4}$$

it is clear that

$$\mathrm{PPre}(T, P) = \bigcup_{i \geq 0} \mathrm{PPre}^i(T, P)\ ; \tag{5}$$
$$\mathrm{Div}(P) = \bigcap_{i \geq 0} \mathrm{PPre}^i(P, P)\ . \tag{6}$$

For finite-state systems, these state sets can be computed in a finite number of steps, but for infinite systems the calculation of $\mathrm{PPre}(T, P)$ or $\mathrm{Div}(P)$ may fail to terminate.

Algorithm 1 uses these operations to calculate a partition that respects divergent stutter bisimulation. Line 1 computes the initial partition based on the propositions. It can be constructed as $\tilde{Q} = Q/\!\sim$ where the relation $\sim\, \subseteq Q \times Q$ is such that $x \sim y$, if $x \models p$ if and only if $y \models p$ for all $p \in \Pi$. Then the algorithm enters the loop, which repeatedly refines each region until all conditions in Def. 14 are satisfied. Line 4 checks for regions to be refined based on divergence. If a region P contains

Algorithm 1: Divergent stutter bisimulation

```
Input: Transition system $G = \langle Q, Q^\circ, \rightarrow, \Pi, \models \rangle$
Output: Coarsest partition $\tilde{Q}$
1  $\tilde{Q} \leftarrow$ Initial proposition-preserving partition;
2  $done \leftarrow$ false;
3  while $\neg done$ do
4      if $\exists P \in \tilde{Q} : \emptyset \subset \mathrm{Div}(P) \subset P$ then
5          $P_1 \leftarrow \mathrm{Div}(P)$;
6          $P_2 \leftarrow P \setminus P_1$;
7          $\tilde{Q} \leftarrow (\tilde{Q} \setminus \{P\}) \cup \{P_1, P_2\}$;
8      else if $\exists P, T \in \tilde{Q} : \emptyset \subset \mathrm{PPre}(T, P) \subset P$ then
9          $P_1 \leftarrow \mathrm{PPre}(T, P)$;
10         $P_2 \leftarrow P \setminus P_1$;
11         $\tilde{Q} \leftarrow (\tilde{Q} \setminus \{P\}) \cup \{P_1, P_2\}$;
12     else
13         $done \leftarrow$ true;
14 return $\tilde{Q}$;
```

both divergent and non-divergent regions, i.e., if the set $\mathrm{Div}(P)$ is nonempty and a proper subset of P, then it is refined by putting the divergent and non-divergent states into their own regions. Additionally, line 8 checks for splits based on stutter predecessors. Here it needs to be checked for each pair of regions P and T whether some of the states in P are stutter predecessors of T while others are not, in which case P is split. The algorithm terminates when no more splits are necessary. Then the result $\tilde{Q}$ represents the coarsest partition that respects divergent stutter bisimulation.

Example 1. (Wagenmaker and Ozay, 2016) Assume the transition relation

$$x(t+1) = 2x(t) + u(t) \ , \tag{7}$$

where $x(t) \in Q = [-1.5, 1.5]$ is the state, and $u(t) \in U = [-2, 2]$ is the control input. The initial partition consists of three regions, $s_1 = [-1.5, -1)$, $s_2 = [-1, 1)$, and $s_3 = [1, 1.5]$. For demonstration, consider the computation of $\mathrm{PPre}(s_3, s_2)$ during the execution of Algorithm 1. First, $\mathrm{PPre}^0(s_3, s_2) = \mathrm{Pre}(s_3) \cap s_2 = [-0.5, 1.75] \cap [-1, 1) = [0.5, 1)$, and then $\mathrm{PPre}^1(s_3, s_2) = \mathrm{Pre}(\mathrm{PPre}^0(s_3, s_2)) \cap s_2 = [-1.25, 1.875) \cap [-1, 1) = [-1, 1) = s_2$. Further iterations produce no change, so s_2 does not need to be split because of s_3. In fact, no regions are split at all, and the initial partition is already a divergent stutter bisimulation. Differently, partitioning based on bisimulation splits s_2 into $\mathrm{Pre}(s_3) \cap s_2 = [0.5, 1)$ and $[-1, 0.5)$, which require further splitting. Wagenmaker and Ozay (2016) show that there exists no finite bisimulation partition for this example. □

Like bisimulation, Algorithm 1 may fail to terminate when given an infinite state space. Yet there are cases such as the above example, where bisimulation fails to terminate while divergent stutter bisimulation yields a finite abstraction. It is clear and well-known that every bisimulation relation also is a divergent stutter bisimulation. It follows that Algorithm 1 terminates more often than bisimulation and always produces the same or a coarser abstraction.

4 Control Strategy

This section presents the main contributions of the paper and shows how divergent stutter bisimulation can be used for synthesis. After partitioning the state space using Algorithm 1, the synthesis problem can be solved for the abstract system. It remains to construct a controller that solves Problem 1 for the original system. Next, Section 4.1 describes the construction of this controller, and afterwards Sections 4.2 and 4.3 show that the method is sound and complete.

4.1 Controller Construction

Given a transition system G and $\mathrm{LTL}_{\setminus\circ}$ specification φ, the objective of this paper is to find a solution for Problem 1, i.e., a deadlock free controller that enforces φ on G. Using Algorithm 1, the transition system is replaced by a divergent stutter bisimilar abstraction $\tilde{G}$. Then a traditional synthesis procedure (Ramadge, 1989; Kloetzer and Belta, 2008) can be used to solve Problem 1 for the abstraction $\tilde{G}$, which results in a controller $\tilde{C}$ that enforces the specification φ on the abstraction $\tilde{G}$.

It follows from the results cited in Section 3.2 that the system G and its abstraction $\tilde{G}$ satisfy the same $\mathrm{LTL}_{\setminus\circ}$ properties, but this is only useful to verify that φ holds on the uncontrolled system. After synthesis, it remains to be shown that the existence of the controller for the abstract system implies the existence of a controller for the original system. Therefore, it is now shown how the abstract controller $\tilde{C}$ that enforces the specification on the abstract system $\tilde{G}$ can be used to design a controller C that enforces the same specification on the original system G. Its construction, which is given in Def. 16, works by considering the states in the original system G that correspond to the classes that form the abstraction $\tilde{G}$.

The controller C to be constructed for the original system observes a path $s = x_1 \cdots x_k$ and makes a control decision $C(s)$ of states allowed next. To base this decision on the abstract controller $\tilde{C}$, the path s is mapped to a path $\tilde{s}$ of the abstract system, and then the control decision $\tilde{C}(\tilde{s})$ is used to inform the choice of $C(s)$.

Consider a path $\tilde{s} = \tilde{x}_1 \cdots \tilde{x}_k \in (Q/{\approx})^+$ of the abstraction, which also is a state of the abstract controlled system $\tilde{C}/\tilde{G}$ by Def. 5. To use $\tilde{C}(\tilde{s})$ when making the control decision $C(s)$, it will be considered what classes can be reached from $\tilde{s}$ under control of $\tilde{C}$ after possible stuttering steps within $\tilde{x}_k$:

$$\tilde{S}^i(\tilde{s}) = \{\, \tilde{y} \in (Q/{\approx}) \mid \tilde{s}\tilde{x}_k^i\tilde{y} \text{ is a path in } \tilde{C}/\tilde{G} \text{ and } \tilde{y} \neq \tilde{x}_k \,\} \; ; \tag{8}$$

$$\tilde{S}(\tilde{s}) = \bigcup\nolimits_{i \geq 0} \tilde{S}^i(\tilde{s}) \; . \tag{9}$$

$\tilde{S}^i(\tilde{s})$ is the set of equivalence classes that can be reached in the abstract controlled system after executing $\tilde{s}$ and then staying i times in the last class $\tilde{x}_k$ of $\tilde{s}$, and $\tilde{S}(\tilde{s})$ is the set of equivalence classes reached after an arbitrary number of repetitions of the last class. This allows for abstract controllers that transition from $\tilde{x}_k$ to the next class immediately, or that keep the system in $\tilde{x}_k$ for some number of steps before transitioning.

Now consider a finite path $s \in Q^+$ in G, for which a control decision is to be made. This path is converted to a path $\tilde{s} \in (Q/{\approx})^+$ so that $\tilde{C}(\tilde{s})$ can be used to inform the control decision $C(s)$. The path $\tilde{s}$ must be constructed in such a way that $\tilde{s}$ is a path in $\tilde{C}/\tilde{G}$ and is stutter equivalent to s. This is achieved by repeating each of the equivalence classes in $[s]$ for the smallest number of times needed to form a path in $\tilde{C}/\tilde{G}$. Specifically, $\tilde{s} = \lfloor s \rfloor_{\tilde{C}}$ according to the following recursive definition, where $r \in Q^*$ and $x, y \in Q$:

$$\lfloor y \rfloor_{\tilde{C}} = [y] \; ; \tag{10}$$

$$\lfloor rxy \rfloor_{\tilde{C}} = \lfloor rx \rfloor_{\tilde{C}} \; , \qquad \text{if } x \approx y \; ; \tag{11}$$

$$\lfloor rxy \rfloor_{\tilde{C}} = \lfloor rx \rfloor_{\tilde{C}}[x]^i[y] \; , \quad \text{if } [y] \in \tilde{S}^i(\lfloor rx \rfloor_{\tilde{C}}) \; ; \tag{12}$$

where $i = \min\{\, i \geq 0 \mid [y] \in \tilde{S}^i(\lfloor rx \rfloor_{\tilde{C}}) \,\}$ in the last case. Also, $\lfloor s \rfloor_{\tilde{C}}$ is undefined if none of the above conditions applies. By construction, $\lfloor s \rfloor_{\tilde{C}}$ is stutter equivalent to s if it is defined, and it is prefix-preserving, i.e., $s \sqsubseteq t$ implies $\lfloor s \rfloor_{\tilde{C}} \sqsubseteq \lfloor t \rfloor_{\tilde{C}}$ if defined.

Now the set of successors outside of the current class that C allows after s is the union of the classes that can be reached under control of $\tilde{C}$ after $\lfloor s \rfloor_{\tilde{C}}$,

$$S(s) = \bigcup \tilde{S}(\lfloor s \rfloor_{\tilde{C}}) \; . \tag{13}$$

After observing $s \in Q^+$, the controller C guides the system to a state in $S(s)$, possibly after stuttering steps within the class of the final state of s. If $\tilde{C}$ permits divergence, then C allows it also, otherwise it follows a shortest path to a state in $S(s)$. These ideas are formalised in the following definition.

Definition 16. Let $G = \langle Q, Q^\circ, \rightarrow, \Pi, \models \rangle$ be a transition system, let $\approx$ be an equivalence relation on Q, let $\tilde{G} = G /\!\!/ {\approx}$, and let $\tilde{C}$ be a controller for $\tilde{G}$. The controller $C \colon Q^* \to 2^Q$ for G is constructed as follows. First, for the empty path $s = \varepsilon$, we have $C(\varepsilon) = \bigcup \tilde{C}(\varepsilon)$. Second, for a nonempty path $s = x_0 \cdots x_n \in Q^+$, the value of $C(s)$ depends on divergence:

(i) If $[x_n]$ is not $\approx$-divergent in G or $\lfloor s \rfloor_{\tilde{C}}$ is divergent in $\tilde{G}$, then

$$C(s) = [x_n] \cup S(s) \; . \tag{14}$$

(ii) If $[x_n]$ is $\approx$-divergent in G and $\lfloor s \rfloor_{\tilde{C}}$ is not divergent in $\tilde{G}$, then

$$C(s) = \begin{cases} S(s) & \text{if } i = 0; \\ \mathrm{PPre}^{i-1}(S(s), [x_n]) & \text{otherwise;} \end{cases} \tag{15}$$

where $i = \min\{\, j \geq 0 \mid x_n \in \mathrm{PPre}^j(S(s), [x_n]) \,\}$.

In Def. 16, C allows as initial states all states in classes allowed as initial by $\tilde{C}$. Otherwise the decision depends on the equivalence class $[x_n]$ that corresponds to the last state of s.

In case (i), this class $[x_n]$ is not divergent or it is divergent in the abstract controlled system, i.e, $\lfloor s \rfloor_{\tilde{C}}[x_n][x_n]\cdots$ is a path in $\tilde{C}/\tilde{G}$, and then C allows the system to stay in $[x_n]$ or to transit to the classes $S(s)$ allowed as successors by $\tilde{C}$. Fig. 1 shows the case of a non-divergent class to the left, and in the middle class that is divergent in the abstract controlled system.

In case (ii), $[x_n]$ is a divergent class in the uncontrolled system but the abstract controller $\tilde{C}$ does not allow this divergence. The controller C prevents the divergent behaviour within $[x_n]$ and forces the system to transition to a different equivalence class eventually. The construction (15) ensures that the system transitions to another equivalence class permitted by the abstract controller when possible, and otherwise only allows transitions within $[x_n]$ that take the system closer to a state from where it is possible to leave $[x_n]$. This case is shown in Fig. 1 to the right.

4.2 Soundness

Soundness means that any controller synthesised by the abstraction-based method correctly solves Problem 1. Here it is shown that if there exists a solution $\tilde{C}$ to the synthesis problem for an abstracted system $\tilde{G}$, then the controller constructed according to Def. 16 solves the synthesis problem for the original system G.

For the controller to be correct, it must enforce its $\mathrm{LTL}_{\setminus\circ}$ specification and be deadlock free, which is shown in Props. 4 and 5 below. Both results depend on the following lemma, which shows that the path $\lfloor s \rfloor_{\tilde{C}}$ used in the construction of the controller C is a path in the abstract controlled system $\tilde{C}/\tilde{G}$ if s is a path in the original controlled system G.

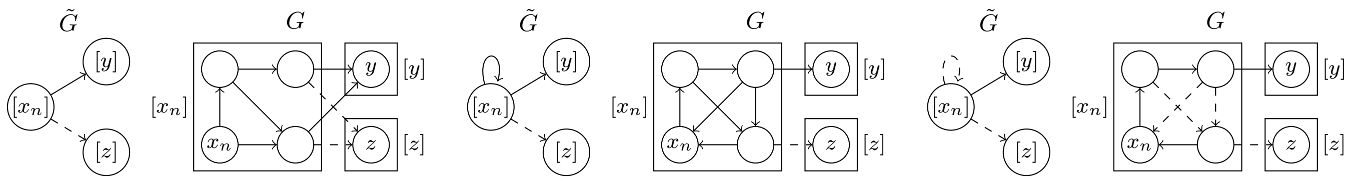


Fig. 1. Construction of a controller for the original system from the abstraction according to Def. 16, depending on whether $[x_n]$ is not divergent in G (left), $\lfloor s \rfloor_{\tilde{C}}$ is divergent in $\tilde{G}$ (middle), or $[x_n]$ is divergent in G and $\lfloor s \rfloor_{\tilde{C}}$ is not divergent in $\tilde{G}$ (right). Transitions disabled by the controller are shown as dashed arrows.

Lemma 3. Let G be a transition system, let $\approx$ be a divergent stutter bisimulation on G, let $\tilde{C}$ be a controller for $\tilde{G} = G/\!\!/\!\approx$, and let C be the controller constructed from $\tilde{C}$ according to Def. 16. If a finite string $s \in Q^+$ is a path in C/G, then $\lfloor s \rfloor_{\tilde{C}}$ is defined and is a path in $\tilde{C}/\tilde{G}$.

Proof. Let $s = x_0 \cdots x_n$, and let $s_k = x_0 \cdots x_k$ be the k-th prefix of s. It is shown by induction on $k = 0, \ldots, n$ that $\lfloor s_k \rfloor_{\tilde{C}}$ is defined and a path in $\tilde{C}/\tilde{G}$.

Base case: $k = 0$. As $s_0 = x_0$ is a path in C/G, it is clear that $x_0 \in Q^\circ$ is an initial state of G and $x_0 \in C(\varepsilon)$. Then $[x_0] \in \{\, [x^\circ] \mid x^\circ \in Q^\circ \,\} = \tilde{Q}^\circ$ is an initial state of $\tilde{G}$, and from $x_0 \in C(\varepsilon) = \bigcup \tilde{C}(\varepsilon)$ it follows that $[x_0] \in \tilde{C}(\varepsilon)$. This means that $[x_0]$ is an initial state of $\tilde{C}/\tilde{G}$, which implies that $\lfloor s_0 \rfloor_{\tilde{C}} = [x_0]$ is defined and a path in $\tilde{C}/\tilde{G}$.

Inductive step: Assume the claim holds for some $0 \le k < n$, i.e, $\lfloor s_k \rfloor_{\tilde{C}}$ is defined and a path in $\tilde{C}/\tilde{G}$. It must be shown that $\lfloor s_{k+1} \rfloor_{\tilde{C}}$ is defined and a path in $\tilde{C}/\tilde{G}$. As $k \ge 0$, the path s_{k+1} can be written as $s_{k+1} = s_k x_{k+1} = s_{k-1} x_k x_{k+1}$. As s and thus its prefix s_{k+1} is a path in C/G, it is clear that $x_{k+1} \in C(s_k)$ by Def. 6. Consider two cases.

If $x_k \approx x_{k+1}$, then it follows by (11) that $\lfloor s_{k+1} \rfloor_{\tilde{C}} = \lfloor s_{k-1} x_k x_{k+1} \rfloor_{\tilde{C}} = \lfloor s_{k-1} x_k \rfloor_{\tilde{C}} = \lfloor s_k \rfloor_{\tilde{C}}$ is defined and a path in $\tilde{C}/\tilde{G}$ by the inductive assumption.

Otherwise $x_{k+1} \notin [x_k]$, and as $x_{k+1} \in C(s_k)$ it follows in both cases (i) and (ii) of Def. 16 that $x_{k+1} \in S(s_k)$. (For case (ii), note that $\mathrm{PPre}^{i-1}(S(s_k), [x_k]) \subseteq [x_k]$.) By (13), there exists $\tilde{x} \in \tilde{S}(\lfloor s_k \rfloor_{\tilde{C}})$ such that $x_{k+1} \in \tilde{x}$, i.e., $[x_{k+1}] = \tilde{x}$. Then by (8)–(9), there exists $i \ge 0$ such that $\lfloor s_k \rfloor_{\tilde{C}} [x_k]^i \tilde{x} = \lfloor s_k \rfloor_{\tilde{C}} [x_k]^i [x_{k+1}]$ is a path in $\tilde{C}/\tilde{G}$. Choose i to be the smallest with this property. Then $\lfloor s_{k+1} \rfloor_{\tilde{C}} = \lfloor s_{k-1} x_k x_{k+1} \rfloor_{\tilde{C}} = \lfloor s_{k-1} x_k \rfloor_{\tilde{C}} [x_k]^i [x_{k+1}] = \lfloor s_k \rfloor_{\tilde{C}} [x_k]^i [x_{k+1}]$ by (12), so $\lfloor s_{k+1} \rfloor_{\tilde{C}}$ is defined and a path in $\tilde{C}/\tilde{G}$. □

Now it can be shown that the controller constructed according to Def. 16 enforces the same $\mathrm{LTL}_{\setminus\circ}$ properties on the original system as the abstract controller does.

Proposition 4. Let G be a transition system, let $\approx$ be a divergent stutter bisimulation on G, let φ be an $\mathrm{LTL}_{\setminus\circ}$ specification, let $\tilde{C}$ be a controller for $\tilde{G} = G/\!\!/\!\approx$ that enforces φ on $\tilde{G}$, and let C be the controller constructed from $\tilde{C}$ according to Def. 16. Then C enforces φ on G.

Proof. Assume $s = x_0 x_1 \ldots$ is an arbitrary infinite path in C/G. It is to be shown that $s \models \varphi$. By Lemma 3, for every prefix $s_n = x_0 \cdots x_n$ of s, it holds that $\lfloor s_n \rfloor_{\tilde{C}}$ is defined and is a path in $\tilde{C}/\tilde{G}$. Therefore, as $\lfloor s \rfloor_{\tilde{C}}$ is prefix-preserving, there is an ascending sequence $\lfloor s_0 \rfloor_{\tilde{C}} \sqsubseteq \lfloor s_1 \rfloor_{\tilde{C}} \sqsubseteq \cdots$ of paths in $\tilde{C}/\tilde{G}$. The closure of this sequence may be finite or infinite.

If it is infinite, then let $\tilde{s} = \mathrm{clo}\{\, \lfloor s_i \rfloor_{\tilde{C}} \mid i \ge 0 \,\}$, which is an infinite path in $\tilde{C}/\tilde{G}$ and stutter equivalent to s.

Otherwise there exists $k \ge 0$ such that $\lfloor s_n \rfloor_{\tilde{C}} = \lfloor s_k \rfloor_{\tilde{C}}$ for all $n \ge k$. As $\lfloor s_n \rfloor_{\tilde{C}}$ is defined for all $n \ge 0$, it must have been constructed from (11) and $x_n \approx x_k$ for all $n \ge k$. As s is a path in G, it follows that s_k is divergent in C/G and x_k is divergent in G. The fact that s_k is divergent in C/G means that $C(s_k)$ cannot be constructed according to case (ii) of Def. 16, as this case only allows a finite number i of steps within $[x_k]$. Then case (i) of Def. 16 must be used, i.e., $\lfloor s_k \rfloor_{\tilde{C}}$ is divergent in $\tilde{C}/\tilde{G}$. This means by Def. 7 that $\tilde{s} = \lfloor s_k \rfloor_{\tilde{C}} [x_k][x_k] \cdots$ is an infinite path in $\tilde{C}/\tilde{G}$. Also, as $[x_n] = [x_k]$ for all $n \ge k$, it is clear that $\tilde{s}$ is stutter equivalent to s.

In both cases, there exists an infinite path $\tilde{s}$ in $\tilde{C}/\tilde{G}$ that is stutter equivalent to s. Since $\tilde{C}$ enforces φ on $\tilde{G}$, it follows that $\tilde{s} \models \varphi$. Since $\tilde{s}$ is stutter equivalent to s, based on Prop. 1 it holds that $s \models \varphi$. □

The second property required of a correct controller is that it is deadlock free. Therefore, the following proposition establishes that if the abstracted closed-loop system is deadlock free, then the controller constructed using Def. 16 produces a deadlock free closed-loop behaviour for the original system.

Proposition 5. Let G be a transition system, let $\approx$ be a divergent stutter bisimulation on G, let $\tilde{C}$ be a controller for $\tilde{G} = G/\!\!/\!\approx$ such that $\tilde{C}/\tilde{G}$ is deadlock free, and let C be the controller constructed from $\tilde{C}$ according to Def. 16. Then C/G is deadlock free.

Proof. Consider a reachable state s of C/G, which can be written as $s = x_0 \cdots x_n$, which must be a path in C/G. It follows by Lemma 3 that $\lfloor s \rfloor_{\tilde{C}}$ is defined and is a path in $\tilde{C}/\tilde{G}$. By construction, $\lfloor s \rfloor_{\tilde{C}}$ can be written as $\lfloor s \rfloor_{\tilde{C}} = \tilde{x}_0 \cdots \tilde{x}_k$ with $x_n \in \tilde{x}_k$.

First consider the case that $\lfloor s \rfloor_{\tilde{C}}$ is divergent in $\tilde{C}/\tilde{G}$, which by Def. 7 means that $\tilde{x}_0\tilde{x}_1 \cdots \tilde{x}_k\tilde{x}_k\tilde{x}_k \cdots$ is a path in $\tilde{C}/\tilde{G}$. This implies that $\tilde{x}_k$ is divergent in $\tilde{G}$, and as $\approx$ is a divergent stutter bisimulation on G, it follows that $[x_n] = \tilde{x}_k$ is $\approx$-divergent in G. Then there exists $x_{n+1} \in [x_n]$ such that $x_n \to x_{n+1}$ in G. Also $C(s)$ is constructed according to case (i) of Def. 16 for divergent $\lfloor s \rfloor_{\tilde{C}}$, and thus $x_{n+1} \in [x_n] \subseteq [x_n] \cup S(s) = C(s)$.

Now consider the case that $\lfloor s \rfloor_{\tilde{C}}$ is not divergent in $\tilde{C}/\tilde{G}$. Then, since $\tilde{C}/\tilde{G}$ is deadlock free, there must exist $i \geq 0$ and a class $\tilde{y} \neq \tilde{x}_k$ such that $\lfloor s \rfloor_{\tilde{C}}\tilde{x}_k^i\tilde{y}$ is a path in $\tilde{C}/\tilde{G}$. It follows from (8)–(9) that $\tilde{y} \in \tilde{S}(\lfloor s \rfloor_{\tilde{C}})$. Also, as $\lfloor s \rfloor_{\tilde{C}}\tilde{x}_k^i\tilde{y}$ is a path in $\tilde{C}/\tilde{G}$, it holds that $\tilde{x}_k \to \tilde{y}$ in $\tilde{G}$, which means that there exist states $x \in \tilde{x}_k$ and $y \in \tilde{y}$ such that $x \to y$ in G. Then $x_n \approx x$, and as $\approx$ is a divergent stutter bisimulation on G, there exists a path fragment

$$x_n \to x_{n+1} \to \cdots \to x_{n+m} \approx y \tag{16}$$

in G where $x_n, \ldots, x_{n+m-1} \in \tilde{x}_k$ and $x_{n+m} \in \tilde{y}$. Assume without loss of generality that (16) is a shortest path fragment with these properties. Note that, as $x_{n+m} \in \tilde{y}$ and $\tilde{y} \in \tilde{S}(\lfloor s \rfloor_{\tilde{C}})$, it follows from (13) that $x_{n+m} \in S(s)$. Now consider the two cases from Def. 16.

(i) $[x_n]$ is not divergent and $C(s) = [x_n] \cup S(s)$. Either it holds that $m > 1$ and $x_{n+1} \in [x_n]$ or $m = 1$ and $x_{n+1} = x_{n+m} \in S(s)$, so it follows that $x_{n+1} \in [x_n] \cup S(s) = C(s)$.

(ii) $[x_n]$ is divergent and $C(s)$ is defined by (15). It is first shown by induction on $j = 0, \ldots, m-1$ that $x_{n+m-j-1} \in \mathrm{PPre}^j(S(s), [x_n])$.

Base case: $j = 0$. As $x_{n+m-1} \in [x_n]$ and $x_{n+m-1} \to x_{n+m} \in S(s)$, it holds that $x_{n+m-1} \in \mathrm{Pre}(S(s)) \cap [x_n] = \mathrm{PPre}^0(S(s), [x_n])$.

Inductive step: Assume the claim $x_{n+m-j-1} \in \mathrm{PPre}^j(S(s), [x_n])$ holds for some $j < m-1$. Then since $x_{n+m-(j+1)-1} = x_{n+m-j-2} \to x_{n+m-j-1}$ in G and $x_{n+m-(j+1)-1} \in \tilde{x}_k = [x_n]$, it is clear that $x_{n+m-(j+1)-1} \in \mathrm{Pre}(\{x_{n+m-j-1}\}) \cap [x_n] \subseteq \mathrm{Pre}(\mathrm{PPre}^j(S(s), [x_n])) \cap [x_n] = \mathrm{PPre}^{j+1}(S(s), [x_n])$.

This completes the induction. If $m = 1$ then it follows with $j = 0$ that $x_n \in \mathrm{PPre}^0(S(s), [x_n])$, and thus $x_{n+1} = x_{n+m} \in S(s) = C(s)$ from (15). If $m > 1$ then it follows with $j = m-2$ that $x_{n+1} = \mathrm{PPre}^{m-2}(S(s), [x_n])$. Also, since (16) is a shortest path fragment, there does not exist $j < m-2$ such that $x_{n+1} \in \mathrm{PPre}^j(S(s), [x_n])$. Then it follows from (15) that $x_{n+1} \in \mathrm{PPre}^{m-2}(S(s), [x_n]) = C(s)$.

In all the cases there exists $x_{n+1} \in C(s)$ with $x_n \to x_{n+1}$. It follows that $s \to sx_{n+1}$ in C/G, which shows that C/G is deadlock free. □

The following theorem combines the results from Props. 4 and 5 to show that synthesis after divergent stutter bisimulation is sound.

Theorem 6 (Soundness). Let G be a transition system and φ be an $\mathrm{LTL}_{\setminus\circ}$ formula. Let $\tilde{C}$ be a controller for $\tilde{G} = G/\!\!/\!\approx$ such that $\tilde{C}/\tilde{G}$ is deadlock free and $\tilde{C}/\tilde{G} \models \varphi$. Let C be the controller constructed from $\tilde{C}$ according to Def. 16. Then C/G is deadlock free and $C/G \models \varphi$

Proof. Follows directly from Props. 4 and 5. □

Theorem 6 establishes that divergent stutter bisimulation can be used as an abstraction method before solving Problem 1. Moreover, Def. 16 can be used to construct a controller for the original system from the abstracted controller, which is a solution for Problem 1.

4.3 Completeness

Completeness of a synthesis method means that if there exists a solution to the synthesis problem, then the method finds a solution. The following theorem shows that, if there exists a controller C for a system G, then there also exists a controller $\tilde{C}$ that enforces the same $\mathrm{LTL}_{\setminus\circ}$ specification on the divergent stutter bisimilar abstraction $\tilde{G}$.

Theorem 7 (Completeness). Let G be a transition system, let $\approx$ be a divergent stutter bisimulation on G, let φ be an $\mathrm{LTL}_{\setminus\circ}$ specification, and let C be a controller for G that enforces φ on G such that C/G is deadlock free. Then there exists a controller $\tilde{C}$ for $\tilde{G} = G/\!\!/\!\approx$ that enforces φ on $\tilde{G}$ such that $\tilde{C}/\tilde{G}$ is deadlock free.

Proof. Since C/G is deadlock free, from every initial state of C/G there exists an infinite path in C/G. Let $\tilde{Q}^\circ_C = \{\, [x^\circ] \mid x^\circ \in Q^\circ \cap C(\varepsilon) \,\}$ be the set of classes of initial states allowed by C. Then choose an initial class $\tilde{x}_0 \in \tilde{Q}^\circ_C$, and an initial state $x_0 \in \tilde{x}_0 \cap Q^\circ \cap C(\varepsilon)$, and an infinite path $s = x_0x_1 \cdots$ in C/G. As s is an infinite path in G, by Theorem 2, there exists a stutter equivalent infinite path $\tilde{s} = \tilde{x}_0\tilde{x}_1 \cdots$ in $\tilde{G}$. Now construct the controller $\tilde{C}$ such that $\tilde{C}(\varepsilon) = \{\tilde{x}_0\}$ and $\tilde{C}(\tilde{x}_0 \cdots \tilde{x}_{k-1}) = \tilde{x}_k$ for all $k \geq 0$. By construction, $\tilde{C}$ constrains the abstracted system $\tilde{G}$ such that it can only follow the infinite path $\tilde{s}$, which implies that $\tilde{C}/\tilde{G}$ is deadlock free. As s is a path in C/G, and C enforces φ on G, it holds that $s \models \varphi$. As $\tilde{s}$ is stutter equivalent to s, it follows by

Prop. 1 that $\tilde{s} \models \varphi$. Then, since the $\tilde{s}$ is the only infinite path in $\tilde{C}/\tilde{G}$, it follows that $\tilde{C}$ enforces φ on $\tilde{G}$. $\square$

Theorem 7 confirms that, whenever there exists a solution to Problem 1, the method of synthesis after divergent stutter bisimulation also finds a solution. Assume that there exists a controller C that enforces a given $\mathrm{LTL}_{\setminus\circ}$ specification on a given system G. Then by Theorem 7, there exists controller $\tilde{C}$ that enforces the same specification on the abstraction $\tilde{G}$. Given a complete procedure to solve the synthesis problem for $\tilde{G}$, it is possible to compute such a solution. This solution can then be converted to a controller for the original system using Def. 16, which by the results of Section 4.2 solves Problem 1 for the original system.

5 Examples

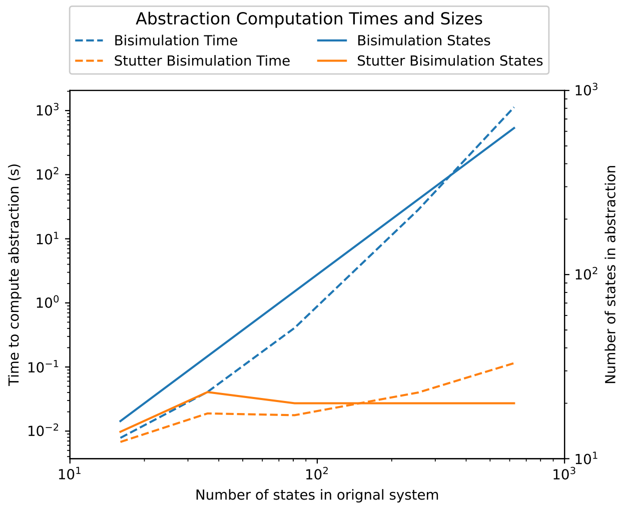
This section applies the synthesis and controller construction method using divergent stutter bisimulation to a variety of systems and compares the results to regular bisimulation.

As shown in Example 1, there are cases where the coarsest partition for bisimulation is infinite while a finite divergent stutter bisimulation partition exists. Even when bisimulation is applicable, stutter bisimulation can offer simpler partitions and faster computation. This is demonstrated with the next two examples featuring finite and continuous state spaces. In both cases the systems are abstracted, controllers are synthesised, and the closed-loop systems are simulated using TuLiP (Filippidis et al., 2016). All computations were performed on an Intel i7-4770K CPU with 32 GiB of RAM.

Example 2. Consider a system where K robots, numbered $1,\ldots,K$, navigate in a two-dimensional grid $G = \{0,\ldots,w\} \times \{0,\ldots,h\}$. The system state x consists of the robot locations $x^1,\ldots,x^K \in G$ and evolves according to

$$(x^i)^+ = x^i + u^i \text{ for } u^i \in \{-1,0,1\}^2 \,. \qquad (17)$$

The corners of the grid are special locations where the robots charge at home or perform tasks: $\mathsf{Home} = (0,0)$, $\mathsf{Task}_0 = (w,0)$, $\mathsf{Task}_1 = (0,h)$, $\mathsf{Task}_2 = (w,h)$. A robot is charged when it enters the home location, and a task is completed when a robot enters the task region. Each robot should be charged infinitely often and each task should be completed infinitely often by some robot, while no two robots should occupy the same location at the



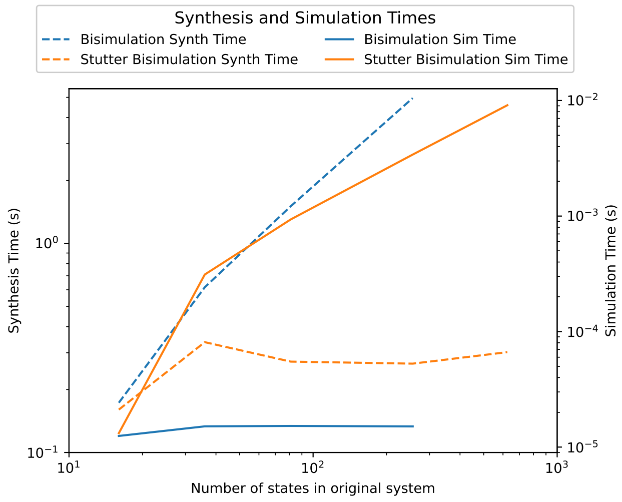


Fig. 2. Abstraction sizes and computation times (top) and synthesis and simulation times (bottom) from Example 2.

same time. This is modelled by the $\mathrm{LTL}_{\setminus\circ}$ formula

$$\square\neg C \wedge \bigwedge_{i=1}^{K} \square\lozenge H_i \wedge \bigwedge_{j=0}^{2} \square\lozenge T_j \,, \qquad (18)$$

where

$$x \models C \;\Leftrightarrow\; \exists i,j : i \neq j \wedge x^i = x^j \,, \qquad (19)$$

$$x \models H_i \;\Leftrightarrow\; x^i = \mathsf{Home} \,, \qquad (20)$$

$$x \models T_j \;\Leftrightarrow\; \exists i : x^i = \mathsf{Task}_j \,. \qquad (21)$$

Experiments are carried out for $K = 2$ robots and grid sizes $(w,h) \in \{(2,2),(2,3),(3,3),(3,4),(4,4),(5,5)\}$. For larger grids, the bisimulation algorithm does not converge within one hour. The states in this example are only bisimilar to themselves, so the coarsest bisimulation is the original partition defined by the propositions (19)–(21). For grid sizes 3×3 and above, it has 20 regions. Fig. 2 (top) shows the computation times, which are significantly faster for divergent stutter bisimulation.

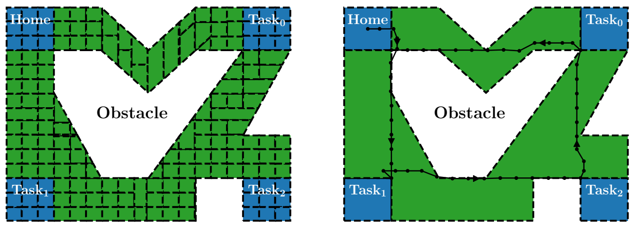


Fig. 3. The abstractions in Example 3 using bisimulation (left) and divergent stutter bisimulation (right). A path using the synthesised controller is shown for the divergent stutter bisimulation.

Next, controllers are synthesised for the abstracted systems and evaluated by simulating a path. At each step, the time to map the control decision from the abstract system to a control decision for the original system is measured and shown in Fig. 2 (bottom). The synthesis time for the divergent stutter bisimulation abstraction is significantly lower as expected from the smaller abstraction size. On the other hand, mapping control decisions from the abstract system to the original system is significantly faster for bisimulation as the construction of the divergent stutter bisimulation controller is nontrivial. □

Example 3. Consider a continuous analogue of Example 2 with one robot modelled by the linear system

$$x^+ = x + u, \;\; u \in \left[-\tfrac{1}{3}, \tfrac{1}{3}\right]^2 . \tag{22}$$

The state space Q is a subset of $[0,6] \times [0,4]$ with polygonal obstacles as shown in Fig. 3 removed. The atomic propositions are as in Example 2, and the satisfaction relation is defined similarly where the home and task locations are squares at the corners of Q.

The abstraction algorithms are implemented for regions given by unions of polytopes whose predecessors can be computed with linear programming. The bisimulation algorithm is prematurely terminated by splitting only regions exceeding a minimum area, resulting in an abstraction with 158 regions that takes 349 s to compute. The divergent stutter bisimulation algorithm takes 2003 s and converges to the original partition. These abstractions are depicted in Fig. 3.

Next, when synthesising controllers, the computer runs out of memory during synthesis for bisimulation, while a controller is synthesised in 38 ms for the much smaller divergent stutter bisimulation abstraction. When simulating a path of the closed-loop system for the divergent stutter bisimulation abstraction, the average time to map control inputs for the abstraction to the original system is 0.81 ms. □

Even when divergent stutter bisimulation does not result in the original partition, the resulting abstraction can still be significantly less complex than abstractions using bisimulation, as shown in the next two examples.

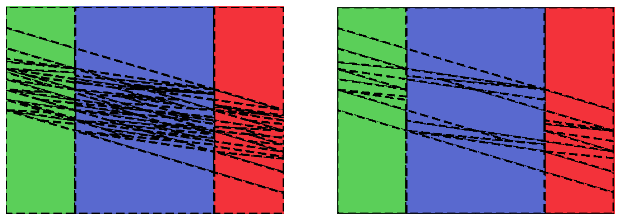


Fig. 4. The abstractions computed in Example 4 using bisimulation (left) and divergent stutter bisimulation (right).

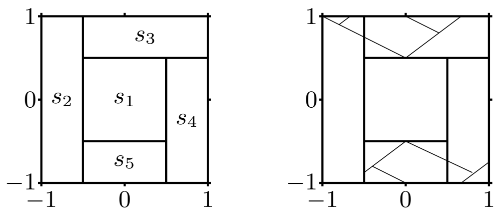


Fig. 5. The partition produced by divergent stutter bisimulation in Example 5.

Example 4. Consider the following continuous-state double-integrator system:

$$\begin{bmatrix} x^+ \\ v^+ \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ v \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \, , \tag{23}$$

where $x \in [-1,1]$, $v \in \mathbb{R}$, and $u \in [-0.5, 0.5]$. This models the position x and velocity v of a ball rolling on a one-dimensional table that can be tilted. The objective is for the ball to visit two regions $L = [-1,-0.5]$ and $R = [0.5, 1]$ repeatedly, which is expressed in LTL$_{\setminus\circ}$ as $\Box \Diamond l \wedge \Box \Diamond r$, where $(x,v) \models l$ if $x \le -0.5$ and $(x,v) \models r$ if $x \ge 0.5$. The bisimulation algorithm terminates in 48 s resulting in an abstraction with 82 regions, while the divergent stutter bisimulation algorithm converges in 200 s to an abstraction with 30 regions, as shown in Fig. 4. Although divergent stutter bisimulation takes longer to compute, it results in a smaller abstraction. □

Example 5. (Wagenmaker and Ozay, 2016) Consider the following two-dimensional linear system:

$$x^+ = \begin{bmatrix} 0.5 & 1 \\ 0.75 & -1 \end{bmatrix} x + u \, , \tag{24}$$

where $x \in [-1,1] \times [-1,1]$ is the state and $u \in [-1,1] \times [-1,1]$ is the control input. The initial partition consists of five regions, $s_1 = [-0.5, 0.5) \times [-0.5, 0.5)$, $s_2 = [-1, 0.5) \times [-1, 1]$, $s_3 = [-0.5, 1] \times [0.5, 1]$, $s_4 = [0.5, 1] \times [-1, 0.5)$, and $s_5 = [-0.5, 0.5) \times [-1, -0.5)$. Divergent stutter bisimulation converges to a partition of 15 regions as shown in Fig. 5, after 37 s computation time. The dual-simulation algorithm converges to a partition of 66 regions after 137 s, while the bisimulation algorithm does not terminate after 1 hour—it produces an approximate partition of 700 regions in 35 minutes. (Wagenmaker and Ozay, 2016). □

6 Conclusions

The abstraction method of *divergent stutter bisimulation* has been applied to reduce a state space before controller synthesis. Divergent stutter bisimulation is a well-known abstraction in the field of model checking, which preserves $\mathrm{CTL}^*_{\setminus\circ}$ properties. This paper leverages these results to simplify the task of controller synthesis. It is shown that a controller synthesised to satisfy any $\mathrm{LTL}_{\setminus\circ}$ specification on a reduced state space based on divergent stutter bisimulation can be converted back to a controller for the original system. This synthesis method is sound and complete relative to a sound and complete synthesis procedure for the abstract state space.

These results improve on bisimulation-based abstraction, because divergent stutter bisimulation results in coarser partitions and is more likely to terminate even for infinite state spaces. This abstraction can also be considered as more insightful for continuous systems as it ignores the number of steps needed to transition from one region to another.

In future work, the authors would like to combine divergent stutter bisimulation with dual-simulation (Wagenmaker and Ozay, 2016). Dual-simulation is related to simulation equivalence (Henzinger et al., 2005) and improves on bisimulation using covers instead of partitions, and could benefit from the abstraction of stuttering steps in the same way as bisimulation does. It would also be interesting to generalise the results about $\mathrm{LTL}_{\setminus\circ}$ specifications to $\mathrm{CTL}_{\setminus\circ}$ or $\mathrm{CTL}^*_{\setminus\circ}$, and to consider control under uncertainty or disturbance.

References

Alur, R., Henzinger, T.A., Lafferriere, G., and Pappas, G.J. (2000). Discrete abstractions of hybrid systems. *Proc. IEEE*, 88(7), 971–984. doi:10.1109/5.871304.

Baier, C. and Katoen, J.P. (2008). *Principles of Model Checking*. MIT Press.

Belta, C., Yordanov, B., and Gol, E.A. (2017). *Formal Methods for Discrete-Time Dynamical Systems*. Springer, 1 edition.

Clarke, Jr., E.M., Grumberg, O., and Peled, D.A. (1999). *Model Checking*. MIT Press.

Fernandez, J.C. (1990). An implementation of an efficient algorithm for bisimulation equivalence. *Sci. Comput. Programming*, 13, 219–236.

Filippidis, I., Dathathri, S., Livingston, S.C., Ozay, N., and Murray, R.M. (2016). Control design for hybrid systems with TuLiP: The temporal logic planning toolbox. In *2016 IEEE Int. Conf. Control Applications (CCA)*, 1030–1041. doi:10.1109/CCA.2016.7587949.

Girard, A. and Pappas, G.J. (2007). Approximation metrics for discrete and continuous systems. *IEEE Transactions on Automatic Control*, 52(5), 782–798.

Groote, J.F., Jansen, D.N., Keiren, J.J.A., and Wijs, A.J. (2017). An $O(m \log n)$ algorithm for computing stuttering equivalence and branching bisimulation. *ACM Trans. Computational Logic*, 18(13).

Groote, J.F. and Vaandrager, F.W. (1990). An efficient algorithm for branching bisimulation and stuttering equivalence. In *17th Int. Colloquium on Automata, Languages, and Programming, ICALP '90*, 626–638. Springer.

Henzinger, T.A., Majumdar, R., and Raskin, J.F. (2005). A classification of symbolic transition systems. *ACM Trans. Computational Logic*, 6(1), 1–32.

Hussien, O. and Tabuada, P. (2018). Lazy controller synthesis using three-valued abstractions for safety and reachability specifications. In *57th IEEE Conf. Decision and Control, CDC 2018*. IEEE.

Kloetzer, M. and Belta, C. (2008). A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Trans. Autom. Control*, 53(1), 287–297. doi:10.1109/TAC.2007.914952.

Liu, J., Ozay, N., Topcu, U., and Murray, R.M. (2013). Synthesis of reactive switching protocols from temporal logic specifications. *IEEE Transactions on Automatic Control*, 58(7), 1771–1785.

Megawati, N.Y. and van der Schaft, A. (2016). Bisimulation equivalence of differential-algebraic systems. *Int. J. Control*, 91(1), 45–56. doi:10.1080/00207179.2016.1266519.

Milner, R. (1989). *Communication and concurrency*. Series in Computer Science. Prentice-Hall.

Nilsson, P., Ozay, N., and Liu, J. (2017). Augmented finite transition systems as abstractions for control synthesis. *Discrete Event Dynamic Systems*, 27(2), 301–340.

Pappas, G.J. (2003). Bisimilar linear systems. *Automatica*, 39, 2035–2047.

Ramadge, P.J.G. (1989). Some tractable supervisory control problems for discrete-event systems modeled by Buchi automata. *IEEE Trans. Autom. Control*, 34(1), 10–19. doi:10.1109/9.8645.

Reissig, G., Weber, A., and Rungger, M. (2016). Feedback refinement relations for the synthesis of symbolic controllers. *IEEE Transactions on Automatic Control*, 62(4), 1781–1796.

Tabuada, P. (2009). *Verification and control of hybrid systems: a symbolic approach*. Springer Science & Business Media.

Wagenmaker, A.J. and Ozay, N. (2016). A bisimulation-like algorithm for abstracting control systems. In *54th Allerton Conf. Communication, Control and Computing*, 569–576. doi:10.1109/ALLERTON.2016.7852282.

Zamani, M., Pola, G., Mazo, M., and Tabuada, P. (2011). Symbolic models for nonlinear control systems without stability assumptions. *IEEE Transactions on Automatic Control*, 57(7), 1804–1809.