Sound and Automated Synthesis of Digital Stabilizing Controllers for Continuous Plants*

Alessandro Abate[1], Iury Bessa[2], Dario Cattaruzza[1], Lucas Cordeiro[1,2], Cristina David[1], Pascal Kesseli[1] and Daniel Kroening[1]

[1]University of Oxford, Oxford, United Kingdom [2]Federal University of Amazonas, Manaus, Brazil

ABSTRACT

Modern control is implemented with digital microcontrollers, embedded within a dynamical plant that represents physical components. We present a new algorithm based on counter-example guided inductive synthesis that automates the design of digital controllers that are correct by construction. The synthesis result is sound with respect to the complete range of approximations, including time discretization, quantization effects, and finite-precision arithmetic and its rounding errors. We have implemented our new algorithm in a tool called DSSynth, and are able to automatically generate stable controllers for a set of intricate plant models taken from the literature within minutes.

Keywords

Digital control synthesis, CEGIS, finite-word-length representation, time sampling, quantization

1. INTRODUCTION

Modern implementations of embedded control systems have proliferated with the availability of low-cost devices that can perform highly non-trivial control tasks, with significant impact in numerous application areas such as environmental control and robotics [4, 17]. Correct control is non-trivial, however. The problem is exacerbated by artifacts specific to digital control, such as the effects of finite-precision arithmetic, time discretization, and quantization noise introduced by A/D and D/A conversion. Thus, programming expertise is a key barrier to broad adoption of correct digital controllers, and requires considerable knowledge outside of the expertise of many control engineers.

Beyond classical a-posteriori validation in digital control, there has been plenty of previous work aiming at *verifying* a given designed controller, which however broadly lack automation. Recent work has studied the stability of digital controllers considering implementation aspects, i.e., fixed-point arithmetic and the word length [8]. They exploit advances in bit-accurate verification of C programs to obtain a verifier for software-implemented digital control.

By contrast, we leverage a very recent step-change in the automation and scalability of *program synthesis*. Program synthesis engines use a specification as the starting point, and subsequently generate a sequence of candidate programs from a given template. The candidate programs are iteratively refined to eventually satisfy the specification. Program synthesizers implementing Counter-Example Guided Inductive Synthesis (CEGIS) [34] are now able to generate programs for highly non-trivial specifications with a very high degree of automation. Modern synthesis engines combine automated testing, genetic algorithms, and SMT-based automated reasoning [1, 11].

By combining and applying state-of-the-art synthesis engines we present a tool that automatically generates digital controllers for a given continuous plant model that are correct by construction. This approach delivers a high degree of automation, promises to reduce the cost and time of development of digital control dramatically, and requires considerably less expertise than a-posteriori verification. Specifically, we synthesize stable, software-implemented embedded controllers along with a model of a physical plant. Due to the complexity of such closed-loop systems, in this work we focus on linear models with known configurations, and perform parametric synthesis of stabilizing digital controllers (further closed-loop performance requirements are left to future work).

Our work addresses challenging aspects of the control synthesis problem. We perform digital control synthesis over a hybrid model, where the plant exhibits continuous behavior whereas the controller operates in discrete time and over a quantized domain. Inspired by a classical approach [4], we translate the problem into a single digital domain, i.e., we model a digital equivalent of the continuous plant by evaluating the effects of the quantizers (A/D and D/A converters) and of time discretization. We further account for uncertainties in the plant model. The resulting closed-loop system is a program with a loop that operates on bit-vectors encoded using fixed-point arithmetic with finite word length (FWL). The three effects of 1. uncertainties, 2. FWL representation and 3. quantization errors are incorporated into the model, and are taken into account during the CEGIS-based synthesis of the control software for the plant.

In summary, this paper makes the following original contributions.

*Supported by EPSRC grant EP/J012564/1, ERC project 280053 (CPROVER) and the H2020 FET OPEN SC^2.

HSCC '17, April 18–20, 2017, Pittsburgh, PA, USA.

ACM ISBN 978-1-4503-4590-3/17/04.

DOI: http://dx.doi.org/10.1145/3049797.3049802

- We automatically generate *correct-by-construction* digital controllers using an inductive synthesis approach. Our application of program synthesis is non-trivial and addresses challenges specific to control systems, such as the effects of quantizers and FWL. In particular, we have found that a two-stage verification engine that continuously refines the precision of the fixed-point representation of the plant yields a speed-up of two orders of magnitude over a conventional one-stage verification engine.
- Experimental results show that DSSynth is able to efficiently synthesize stable controllers for a set of intricate benchmarks taken from the literature: the median runtime for our benchmark set considering the faster engine is 48 s, i.e., half of the controllers can be synthesized in less than one minute.

2. PRELIMINARIES

2.1 Discretization of the Plant

The digital controllers synthesized using the algorithm we present in this paper are typically used in closed loops with continuous (physical) plants. Thus, we consider continuous dynamics (the plant) and discrete parts (the digital controller). In order to obtain an overall model for the synthesis, we discretize the continuous plant and particularly look at the plant dynamics from the perspective of the digital controller.

As we only consider transfer function models, and require a z-domain transfer function $G(z)$ that captures all aspects of the continuous plant, which is naturally described via a Laplace-domain transfer function $G(s)$. The continuous model of the plant must be discretized to obtain the corresponding coefficients of $G(z)$.

Among the discretization methods in the literature [17], we consider the sample-and-hold processes in complex systems [20]. On the other hand, the ZOH discretization models the exact effect of sampling and DAC interpolation over the plant.

ASSUMPTION 1. *The sample-and-hold effects of the ADC and the presence of the ZOH of the DAC are synchronized, namely there is no delay between sampling the plant output at the ADC and updating the DAC accordingly. The DAC interpolator is an ideal ZOH.*

LEMMA 1. *[4] Given a synchronized ZOH input and sample-and-hold output on the plant, with a sample time T satisfying the Nyquist criterion, the discrete pulse transfer function $G(z,T)$ is an exact z-domain representation of $G(s)$, and can be computed using the following formula:*

$$G(z,T) = (1-z^{-1})\mathcal{Z}\left\{\mathcal{L}^{-1}\left\{\frac{G(s)}{s}\right\}_{t=kT}\right\}. \quad (1)$$

In this study, for the sake of brevity, we will use the notation $G(z)$ to represent the pulse transfer function $G(z,T)$. Lemma 1 ensures that the poles and zeros match under the $\mathcal{Z}\left\{\mathcal{L}^{-1}\left\{\cdot\right\}_{t=kT}\right\}$ operations, and it includes the ZOH dynamics in the $(1-z^{-1})$ term. This is sufficient for stability studies over $G(s)$ [15], i.e., if there is any unstable pole (in the complex domain $\Re\{s\} > 0$), the pulse transfer function in (1) will also present the same number of unstable poles ($|z| > 1$) [17].

2.2 Model Imprecision, Finite Word Length Representation and Quantization Effects

Let $C(z)$ be a digital controller and $G(z)$ be a discrete-time representation of the plant, given as

$$C(z) = \frac{C_n(z)}{C_d(z)} = \frac{\beta_0 + \beta_1 z^{-1} + ... + \beta_{M_C} z^{-M_C}}{\alpha_0 + \alpha_1 z^{-1} + ... + \alpha_{N_C} z^{-N_C}}, \quad (2)$$

$$G(z) = \frac{G_n(z)}{G_d(z)} = \frac{b_0 + b_1 z^{-1} + ... + b_{M_G} z^{-M_G}}{a_0 + a_1 z^{-1} + ... + a_{N_G} z^{-N_G}}. \quad (3)$$

where $\vec{\beta}$ and $\vec{\alpha}$ are vectors containing the controller's coefficients; similarly, $\vec{b}$ and $\vec{a}$ denote the plant's coefficients; and finally $N_{(\cdot)}$ and $M_{(\cdot)}$ indicate the order of the polynomials, and we require in particular that $N_G \geq M_G$.

Uncertainties in $G(z)$ may appear owing to: 1. uncertainties in $G(s)$ (we denote the uncertain continuous plant by $\hat{G}(s) = \frac{G_n(s)+\Delta_p G_n(s)}{G_d(s)+\Delta_p G_d(s)}$ to explicitly encompass the effects of the uncertainty terms $\Delta_p G_{(\cdot)}(s)$) arising from tolerances/imprecision in the original model; 2. errors in the numerical calculations due to FWL effects (e.g., coefficient truncation and round-off, which will be denoted as $\Delta_b G_n(s), \Delta_b G_d(s)$); and 3. errors caused by quantization (which we model later as as external disturbances ν_1 and ν_2). These uncertainties are parametrically expressed by additive terms, eventually resulting in an uncertain model $\hat{G}(z)$, such that:

$$\hat{G}(z) = \frac{G_n(z) + \Delta G_n(z)}{G_d(z) + \Delta G_d(z)}, \quad (4)$$

which will be represented by the following transfer function:

$$\hat{G}(z) = \frac{\hat{b}_0 + \hat{b}_1 z^{-1} + ... + \hat{b}_{M_G} z^{-M_G}}{\hat{a}_0 + \hat{a}_1 z^{-1} + ... + \hat{a}_{N_G} z^{-N_G}}. \quad (5)$$

Notice that, due to the nature of the methods we use for the stability check, we require that the parametric errors in the plant have the same polynomial order as the plant itself (indeed, all other errors described in this paper fulfill this property). We also remark that, due to its native digital implementation, there are no parametric errors $(\Delta_p C_n(z), \Delta_p C_d(z))$ in the controller. Thus $\hat{C}(z) \equiv C(z)$.

We introduce next a notation based on the coefficients of the polynomial to simplify the presentation. Let $\mathcal{P}^N$ be the space of polynomials of order N. Let $P \in \mathcal{P}^{M,N}$ be a rational polynomial $\frac{P_n}{P_d}$, where $P_n \in \mathcal{P}^M$ and $P_d \in \mathcal{P}^N$. For a vector of coefficients

$$\vec{P} \in \mathbb{R}^{N+M+2} = [n_0\ n_1\ \ldots\ n_M\ d_0\ d_1\ \ldots\ d_N\ ]^T \quad (6)$$

and an uncertainty vector

$$\Delta\vec{P} \in \mathbb{R}^{N+M+2} = [\Delta n_0\ \ldots\ \Delta n_M\ \Delta d_0\ \ldots\ \Delta d_N\ ]^T \quad (7)$$

we write

$$\hat{\vec{G}} = \vec{G} + \Delta\vec{G}, \text{ where} \quad (8)$$

$$\vec{G} \in \mathbb{R}^{N_G+M_G+2} = [b_0\ \ldots\ b_{M_G}\ a_0\ \ldots\ a_{N_G}\ ]^T,$$

$$\Delta\vec{G} \in \mathbb{R}^{N_G+M_G+2} = [\Delta b_0\ \ldots\ \Delta b_{M_G}\ \Delta a_0\ \ldots\ \Delta a_{N_G}\ ]^T.$$

In the following we will either manipulate the transfer functions $G(z)$, $C(z)$ directly, or work over their respective coefficients $\vec{G}$, $\vec{C}$ in vector form.

A typical digital control system with a continuous plant and a discrete controller is illustrated in Figure 1. The DAC and ADC converters introduce quantization errors (notice

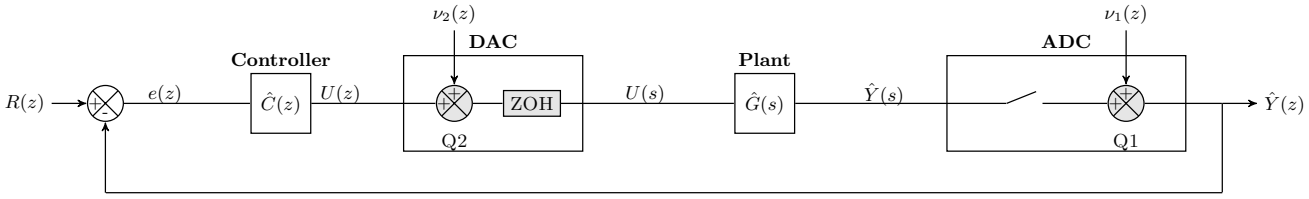


Figure 1: Closed-loop digital control system (cf. Section 2.2 for notation)

that each of them may have a different FWL representation than the controller), which are modeled as disturbances $\nu_1(z)$ and $\nu_2(z)$; $G(s)$ is the continuous-time plant model with parametric additive uncertainty $\Delta_p G_n(s)$ and $\Delta_p G_d(s)$ (as mentioned above); $R(z)$ is a given reference signal; $U(z)$ is the control signal; and $\hat{Y}(z)$ is the output signal affected by the disturbances and uncertainties in the closed-loop system The ADC and DAC may be abstracted by transforming the closed-loop system in Figure 1 into the digital system in Figure 2, where the effect of ν_1 and ν_2 in the output $Y(z)$ is additive noise.

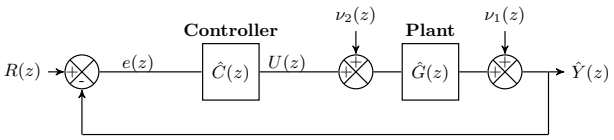


Figure 2: Fully digital equivalent to system in Figure 1

In Figure 2, two sources of uncertainty are illustrated: parametric uncertainties model the errors (which are represented by $\Delta_p\vec{G}$), and uncertainties for the quantizations in the ADC and DAC conversions (ν_1 and ν_2), which are assumed to be non-deterministic. Recall that we discussed how the quantization noise is an additive term, which means it does not enter parametrically in the transfer function. Instead, we later show that the system is stable given these non-deterministic disturbance.

The uncertain model may be rewritten as a vector of coefficients in the z-domain using equation (8) as $\vec{\hat{G}} = \vec{G} + \Delta_p\vec{G}$. The parametric uncertainties in the plant are assumed to have the same order as the plant model, since errors of higher order can move the closed-loop poles by large amounts, thus preventing any given controller from stabilizing such a setup. This is a reasonable assumption since most tolerances do not change the architecture of the plant.

Direct use of controllers in fixed-point representation.

Since the controller is implemented using finite representation, $C(z)$ also suffers disturbances from the FWL effects, with roundoffs in coefficients that may change closed-loop poles and zeros position, and consequently affect its stability, as argued in [8].

Let $\hat{C}(z)$ be the digital controller transfer function represented using this FWL with integer size I and fractional size F. The term I affects the range of the representation and is set to avoid overflows, while F affects the precision and the truncation after arithmetic operations. We shall denote the FWL domain of the coefficients by $\mathbb{R}\langle I, F\rangle$ and define a function

$$\mathcal{F}_n\langle I,F\rangle(P \in \mathcal{P}) : \mathcal{P}^n \to \mathcal{P}^n\langle I,F\rangle \tag{9}$$

$$\triangleq \tilde{P} \in \mathcal{P}^n\langle I,F\rangle : c_i \in \vec{P} \wedge \tilde{c}_i \in \vec{\tilde{P}} = \mathcal{F}_0\langle I,F\rangle(c_i),$$

where $\mathcal{P}^n$ is the space of polynomials of n-th order, $\mathcal{P}^n\langle I,F\rangle$ is the space of polynomials with coefficients in $\mathbb{R}\langle I,F\rangle$, and (as a special case) $\mathcal{F}_0\langle I,F\rangle(x)$ returns the element $\tilde{x} \in \mathbb{R}\langle I,F\rangle$ that is closest to the real parameter x.

Similarly, $\mathcal{F}_{n,m}\langle I,F\rangle(\cdot) : \mathcal{P}^{n,m} \to \mathcal{P}^{n,m}\langle I,F\rangle$ applies the same effect to a ratio of polynomials, where $\mathcal{P}^{n,m}$, $\mathcal{P}^{n,m}\langle I,F\rangle$ are rational polynomial domains.

Thus, the perturbed controller model $\tilde{C}(z)$ may be obtained from the original model $\hat{C}(z) = C(z) = \frac{C_n(z)}{C_d(z)}$ as follows:

$$\tilde{C}(z) = \mathcal{F}_{M_C,N_C}\langle I,F\rangle(C(z)) = \frac{\mathcal{F}_{M_C}\langle I,F\rangle(\hat{C}_n(z))}{\mathcal{F}_{N_C}\langle I,F\rangle(\hat{C}_d(z))}. \tag{10}$$

In the case of a digitally synthesized controller (as it is the case in this work), $\tilde{C}(z) \equiv \hat{C}(z) \equiv C(z)$ because the synthesis is performed directly using FWL representation. In other words, we synthesize a controller that is already in the domain $\mathbb{R}\langle I,F\rangle$ and has therefore no uncertainties entering because of FWL representations, that is, $\Delta_b C_n(z) = \Delta_b C_d(z) = 0$.

Fixed-point computation in program synthesis.

The program synthesis engine uses fixed-point arithmetic. Specifically, we use the domain $\mathbb{R}\langle I,F\rangle$ for the controller's coefficients and the domain $\mathbb{R}\langle I_p,F_p\rangle$ for the plant's coefficients, where I and F, as well as I_p and F_p, denote the number of bits for the integer and fractional parts, respectively, and where it is practically motivated to consider $\mathbb{R}\langle I_p,F_p\rangle \supseteq \mathbb{R}\langle I,F\rangle$.

Given the use of fixed-point arithmetic, we examine the discretization effect during these operations. Let $\tilde{C}(z)$ and $\tilde{G}(z)$ be transfer functions represented using fixed-point bit-vectors.

$$\tilde{C}(z) = \frac{\tilde{\beta}_0 + \tilde{\beta}_1 z^{-1} + ... + \tilde{\beta}_{M_C} z^{-M_C}}{\tilde{\alpha}_0 + \tilde{\alpha}_1 z^{-1} + ... + \tilde{\alpha}_{N_C} z^{-N_C}}, \tag{11}$$

$$\tilde{G}(z) = \frac{\tilde{b}_0 + \tilde{b}_1 z^{-1} + ... + \tilde{b}_{M_G} z^{-M_G}}{\tilde{a}_0 + \tilde{a}_1 z^{-1} + ... + \tilde{a}_{N_G} z^{-N_G}}. \tag{12}$$

Recall that since the controller is synthesized in the $\mathbb{R}\langle I,F\rangle$ domain, $\tilde{C}(z) \equiv \hat{C}(z) \equiv C(z)$. However, given a real plant $\hat{G}(z)$, we need to introduce $\tilde{G}(z) = \mathcal{F}_{M_G,N_G}\langle I_p,F_p\rangle(\hat{G}(z))$, where

$$\tilde{G}(z) = \frac{(\hat{b}_0 + \Delta_b\hat{b}_0) + ... + (\hat{b}_{M_G} + \Delta_b\hat{b}_{M_G})z^{-M_G}}{(\hat{a}_0 + \Delta_b\hat{a}_0) + ... + (\hat{a}_{N_G} + \Delta_b\hat{a}_{N_G})z^{-N_G}}$$

$$\vec{\tilde{G}} = \vec{\hat{G}} + \Delta_b\vec{G} = \vec{G} + \Delta_p\vec{G} + \Delta_b\vec{G}, \tag{13}$$

where $\Delta_b c_i = \tilde{c}_i - \hat{c}_i$, and $\Delta_b G$ represents the plant uncertainty caused by the rounding off effect. We capture the global uncertainty as $\Delta\vec{G} = \Delta_p\vec{G} + \Delta_b\vec{G}$.

2.3 Closed-Loop Stability Verification under Parametric Uncertainties, FWL Representation and Quantization Noise

Sound synthesis of the digital controller requires the consideration of the effect of FWL on the controller, and of quantization disturbances in the closed-loop system. Let the quantizer $Q1$ (ADC) be the source of a white noise ν_1, and $Q2$ (DAC) be the source of a white noise ν_2. The following equation models the system in Figure 1, including the parametric uncertainties $\Delta\vec{G}$ and the FWL effects on the controller $\tilde{C}(z)$:

$$\hat{Y}(z) = \nu_1(z) + \hat{G}(z)\tilde{C}(z)R(z) + \hat{G}(z)\nu_2(z) - \hat{G}(z)\tilde{C}(z)\hat{Y}(z). \tag{14}$$

The above can be rewritten as follows:

$$\hat{Y}(z) = H_1(z)\nu_1(z) + H_2(z)\nu_2(z) + H_3(z)R(z), \tag{15}$$

where

$$H_1(z) = \frac{1}{1 + \hat{G}(z)\tilde{C}(z)},$$

$$H_2(z) = \frac{\hat{G}(z)}{1 + \hat{G}(z)\tilde{C}(z)},$$

$$H_3(z) = \frac{\hat{G}(z)\tilde{C}(z)}{1 + \hat{G}(z)\tilde{C}(z)}.$$

Assumption 2. *The quantization noises ν_1 (from $Q1$) and ν_2 (from $Q2$) are uncorrelated white noises and their amplitudes are always bounded by the half of quantization step [4], i.e., $|\nu_1| \leq \frac{q_1}{2}$ and $|\nu_2| \leq \frac{q_2}{2}$, where q_1 and q_2 are the quantization steps of ADC and DAC, respectively.*

A discrete-time dynamical system is said to be Bounded-Input and Bounded-Output (BIBO) stable if bounded inputs necessarily result in bounded outputs. This condition holds true over an LTI model if and only if every pole of its transfer function lies inside the unit circle [5]. Analyzing Eq. (15), the following proposition provides conditions for the BIBO stability of the system in Figure 1, with regards to the exogenous signals $R(z)$, ν_1, and ν_2, which are all bounded (in particular, the bound on the quantization noise is given by Assumption 2).

Proposition 1. *[8, 15] Consider a feedback closed-loop control system as given in Figure 1 with a FWL implementation of the digital controller $\tilde{C}(z) = \mathcal{F}_{M_C,N_C}\langle I, F\rangle(C(z))$ and uncertain discrete model of the plant from (6), (7)*

$$\hat{G}(z) = \frac{\hat{G}_n(z)}{\hat{G}_d(z)}, \quad \vec{\hat{G}} = \vec{G} + \Delta_p\vec{G}.$$

Then $\hat{G}(z)$ is BIBO-stable if and only if:

- *the roots of characteristic polynomial $S(z)$ are inside the open unit circle, where $S(z)$ is:*

$$S(z) = \tilde{C}_n(z)\hat{G}_n(z) + \tilde{C}_d(z)\hat{G}_d(z); \tag{16}$$

- *the direct loop product $\tilde{C}(z)\hat{G}(z)$ has no pole-zero cancellation on or outside the unit circle.*

Proposition 1 provides necessary (and sufficient) conditions for the controller to stabilize the closed-loop system, considering plant parametric uncertainties (i.e., $\Delta_p\vec{G}$), quantization noises (ν_1 and ν_2) and FWL effects in the control software. In particular, note that the model for quantization noise enters as a signal to be stabilized: in practice, if the quantization noise is bounded, the noise may be disregarded if the conditions on Proposition 1 are satisfied.

If the verification is performed using FWL arithmetic, the above equations must use $\tilde{G}(z)$ instead of $\hat{G}(z)$. The former will provide sufficient conditions for the latter to be stabilized.

3. AUTOMATED PROGRAM SYNTHESIS FOR DIGITAL CONTROL

3.1 Overview of the Synthesis Process

In order to synthesize closed-loop digital control systems, we use a program synthesis engine. Our program synthesizer implements Counter-Example Guided Inductive Synthesis (CEGIS) [34]. We start by presenting its general architecture followed by describing the parts specific to closed-loop control systems. A high-level view of the synthesis process is given in Figure 3. Steps 1 to 3 are performed by the user and Steps A to D are automatically performed by our tool for Digital Systems Synthesis, named DSSynth.

CEGIS-based control synthesis requires a formal verifier to check whether a candidate controller meets the requirements when combined with the plant. We use the Digital-System Verifier (DSVerifier) [19] in the verification module for DSSynth. It checks the stability of closed-loop control systems and considers finite-word length (FWL) effects in the digital controller, and uncertainty parameters in the plant model (plant intervals) [8].

Given a plant model in ANSI-C syntax as input (Steps 1–3), DSSynth constructs a non-deterministic model to represent the plant family, i.e., it addresses plant variations as interval sets (Step A), and formulates a function (Step B) using implementation details provided in Steps 2 and 3 to calculate the controller parameters to be synthesized (Step C). Note that DSSynth synthesizes the controller for the desired numerical representation and realization form. Finally, DSSynth builds an intermediate ANSI-C code for the digital system implementation, which is used as input for the CEGIS engine (Step D).

This intermediate ANSI-C code model contains a specification ϕ for the property of interest (i.e., robust stability) and is passed to the Counterexample-Guided Inductive Synthesis (CEGIS) module of CBMC [9], where the controller is marked as the input variable to synthesize. CEGIS employs an iterative, counterexample-guided refinement process, which is explained in detail in Section 3.2. CEGIS reports a successful synthesis result if it generates a controller that is safe with respect to ϕ. In particular, the ANSI-C code model guarantees that a synthesized solution is complete and sound with respect to the stability property ϕ, since it does not depend on system inputs and outputs. In the case of stability, the specification ϕ consists of a number of assumptions on the polynomial coefficients, following Jury's Criteria, as well as the restrictions on the representation of these coefficients as discussed in detail in Section 3.3.

3.2 Architecture of the Program Synthesizer

The input specification provided to the program synthesizer is of the form $\exists\vec{P}.\forall\vec{x}.\sigma(\vec{x},\vec{P})$ where $\vec{P}$ ranges over functions,

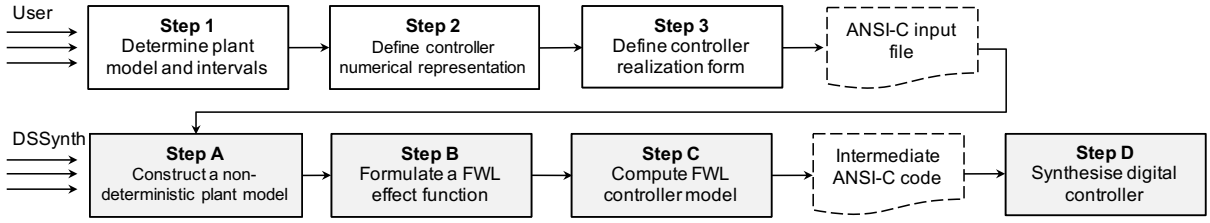


Figure 3: Overview of the synthesis process

$\vec{x}$ ranges over ground terms and σ is a quantifier-free formula. We interpret the ground terms over some finite domain $\mathcal{D}$.

The design of our synthesizer is given in Figure 4 and consists of two phases, SYNTHESIZE and VERIFY, which interact via a finite set of test vectors INPUTS that is updated incrementally. Given the aforementioned specification σ, the SYNTH procedure tries to find an existential witness $\vec{P}$ satisfying the specification $\sigma(\vec{x}, \vec{P})$ for all $\vec{x}$ in INPUTS (as opposed to all $\vec{x} \in \mathcal{D}$). If SYNTHESIZE succeeds in finding a witness $\vec{P}$, this witness is a candidate solution to the full synthesis formula. We pass this candidate solution to VERIFY, which checks whether it is a full solution (i.e., $\vec{P}$ satisfies the specification $\sigma(\vec{x}, \vec{P})$ for all $\vec{x} \in \mathcal{D}$). If this is the case, then the algorithm terminates. Otherwise, additional information is provided to the SYNTHESIZE phase in the form of a new counterexample that is added to the INPUTS set and the loop iterates again (the second feedback signal "Increase Precision" provided by the VERIFY phase in Figure 4 is specific to control synthesis and will be described in the next section).

Each iteration of the loop adds a new input to the finite set INPUTS that is used for synthesis. Given that the full set of inputs $\mathcal{D}$ is finite, this means that the refinement loop can only iterate a potentially very large, but finite number of times.

3.3 Synthesis for Control

Formal specification of the stability property.

Next, we describe the specific property that we pass to the program synthesizer as the specification σ. There are a number of algorithms in our verification engine that can be used for stability analysis [7, 8]. Here we choose Jury's criterion [4] in view of its efficiency and ease of integration within DSSynth: we employ this method to check the stability in the z-domain for the characteristic polynomial $S(z)$ defined in (16). We consider the following form for $S(z)$:

$$S(z) = a_0 z^N + a_1 z^{N-1} + \cdots + a_{N-1} z + a_N = 0, a_0 \neq 0.$$

Next, the following matrix $M = [m_{ij}]_{(2N-2)\times N}$ is built from $S(z)$ coefficients:

$$M = \begin{pmatrix} V^{(0)} \\ V^{(1)} \\ \vdots \\ V^{(N-2)} \end{pmatrix},$$

where $V^{(k)} = [v_{ij}^{(k)}]_{2\times N}$ such that:

$$v_{ij}^{(0)} = \begin{cases} a_{j-1}, & \texttt{if } i = 1 \\ 0 \\ v_{(1)(N-j+1)}, & \texttt{if } i = 2 \end{cases}$$

$$v_{ij}^{(k)} = \begin{cases} 0, & \texttt{if } j > n-k \\ v_{1j}^{(k-1)} - v_{2j}^{(k-1)} \cdot \frac{v_{11}^{(k-1)}}{v_{21}^{(k-1)}}, & \texttt{if } j \leq n-k \texttt{ and } i = 1 \\ v_{(1)(N-j+1)}^{k}, & \texttt{if } j \leq n-k \texttt{ and } i = 2 \end{cases}$$

and where $k \in \mathbb{Z}$ is such that $0 < k < N-2$. We have that [4] $S(z)$ is the characteristic polynomial of a stable system if and only if the following four conditions hold: $R_1 : S(1) > 0$; $R_2 : (1)^N S(1) > 0$; $R_3 : |a_0| < a_N$; $R_4 : m_{11} > 0 \wedge m_{31} > 0 \wedge m_{51} > 0 \wedge \ldots \wedge m_{(2N-3)(1)} > 0$. The stability property is then encoded by a constraint of the form: $\phi_{stability} \equiv (R_1 \wedge R_2 \wedge R_3 \wedge R_4)$.

The synthesis problem.

The synthesis problem we are trying to solve is the following: find a digital controller $\tilde{C}(z)$ that makes the closed-loop system stable for all possible uncertainties $\tilde{G}(z)$ (13). When mapping back to the notation used for describing the general architecture of the program synthesizer, the controller $\tilde{C}(z)$ denotes P and $\tilde{G}(z)$ represents x.

As mentioned above, we compute the coefficients for $\tilde{C}(z)$ in the domain $\mathbb{R}\langle I, F\rangle$, and those for $\tilde{G}(z)$ in the domain $\mathbb{R}\langle I_p, F_p\rangle$. While the controller's precision $\langle I, F\rangle$ is given, we can vary $\langle I_p, F_p\rangle$ such that $\mathbb{R}\langle I_p, F_p\rangle \supseteq \mathbb{R}\langle I, F\rangle$. As the cost of SAT solving increases with in the size of the problem instance, our algorithm tries to solve the problem first for small I_p, F_p, iteratively increasing the precision if it is insufficient.

3.4 The SYNTHESIZE and VERIFY phases

The SYNTHESIZE phase uses BMC to compute a solution $\tilde{C}(z)$. There are two alternatives for the VERIFY phase. The first approach uses interval arithmetic [28] to represent the coefficients $[c_i - \Delta_p c_i - \Delta_b c_i, c_i + \Delta_p c_i - \Delta_b c_i + (2^{-F_p})]$ and rounds outwards. This0 allows us to simultaneously evaluate the full collection of plants $\hat{G}(s)$ (i.e., all concrete plants $G(s)$ in the range $G(s) \pm \Delta_p G(s)$) plus the effects of numeric calculations. Synthesized controllers are stable for all plants in the family. Preliminary experiments show that a synthesis approach using this verification engine has poor performance and we therefore designed a second approach. Our experimental results in Section 4 show that the speedup yielded by the second approach is in most cases of at least two orders of magnitude.

The second approach is illustrated in Figure 4 and uses a two-stage verification approach: the first stage performs potentially unsound fixed-point operations assuming a plant precision $\langle I_p, F_p\rangle$, and the second stage restores soundness by validating these operations using interval arithmetic on the synthesized controller. In more detail, in the first stage, denoted by UNCERTAINTY in Figure 4, assuming a precision $\langle I_p, F_p\rangle$ we check whether the system is unstable for the current candidate solution, i.e., if $\neg\phi_{stability}$ is satisfiable for $S(z)$. If this is the case, then we obtain a counterexample $\tilde{G}(z)$,

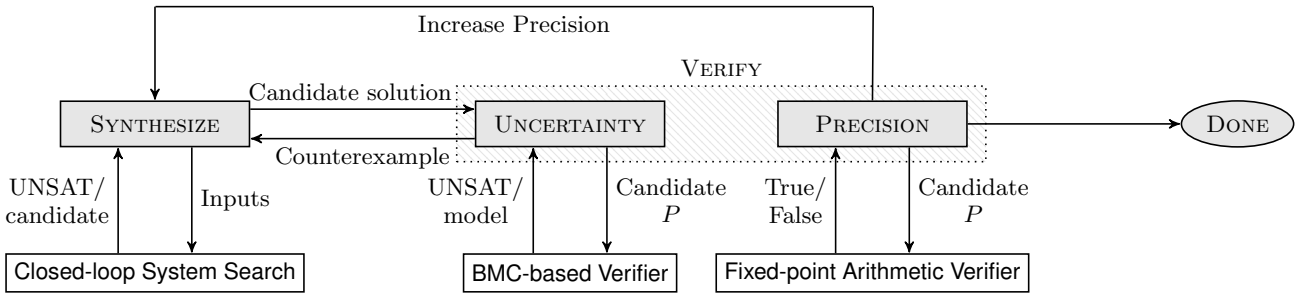


Figure 4: Counterexample-Guided Inductive Synthesis of Closed-loop Systems (Step D)

which makes the closed-loop system unstable. This uncertainty is added to the set INPUTS such that, in the subsequent SYNTHESIZE phase, we obtain a candidate solution consisting of a controller $C(z)$, which makes the closed-loop system stable for all the uncertainties accumulated in INPUTS.

If the UNCERTAINTY verification stage concludes that the system is stable for the current candidate solution, then we pass this solution to the second verification stage, PRECISION, which checks the propagation of the error in the fixed-point calculations using a Fixed-point Arithmetic Verifier based on interval arithmetic.

If the PRECISION verification returns *false*, then we increase the precision of $\langle I_p, F_p \rangle$ and re-start the SYNTHESIZE phase with an empty INPUTS set. Otherwise, we found a full sound solution for our synthesis problem and we are done.

In the rest of the paper, we will refer to the two approaches for the VERIFY phase as one-stage and two-stage, respectively.

3.5 Soundness

The SYNTHESISE phase generates potentially unsound candidate solutions. The soundness of the model is ensured by the VERIFY phase. If a candidate solution passes verification, it is necessarily sound.

The VERIFY phase has two stages. The first stage ensures that no counterexample plant with an unstable closed loop exists over finite-precision arithmetic. Since the actual plant uses reals, we need to ensure we do not miss a counterexample because of rounding errors. For this reason, the second verification stage uses an overapproximation with interval arithmetic with outward rounding. Thus, the first verification stage underapproximates and is used to generate counterexamples, and the second stage overapproximates and provides proof that no counterexample exists.

3.6 Illustrative Example

We illustrate our approach with a classical cruise control example from the literature [5]. It highlights the challenges that arise when using finite-precision arithmetic in digital control. We are given a discrete plant model (with a time step of 0.2 s), represented by the following z-expression:

$$G(z) = \frac{0.0264}{z - 0.9998}. \tag{17}$$

Using an optimization tool, the authors of [36] have designed a high-performance controller for this plant, which is characterized by the following z-domain transfer function:

$$C(z) = \frac{2.72z^2 - 4.153z + 1.896}{z^2 - 1.844z + 0.8496}. \tag{18}$$

The authors of [36] claim that the controller $C(z)$ in (18) stabilizes the closed-loop system for the discrete plant model $G(z)$ in (17). However, if the effects of finite-precision arithmetic are considered, then this closed-loop system becomes unstable. For instance, an implementation of $C(z)$ using $\mathbb{R}\langle 4, 16\rangle$ fixed-point numbers (i.e., 4 bits for the integer part and 16 bits for the fractional part) can be modeled as:

$$\tilde{C}(z) := \frac{2.7199859619140625z^2 - 4.1529998779296875z + 1.89599609375}{z^2 - 1.843994140625z + 0.8495941162109375}. \tag{19}$$

The resulting system, where $\tilde{C}(z)$ and $G(z)$ are in the forward path, is unstable. Notice that this is disregarding further approximation effects on the plant caused by quantization in the verifier (i.e., $\tilde{G}(z)$). Figure 5a gives the Bode diagram for the digital controller represented in (18): as the phase margin is negative, the controller is unstable when considering the FWL effects.

3.7 Program Synthesis for the Example

We now demonstrate how our approach solves the synthesis problem for the example given in the previous section. Assuming a precision of $I_p = 16$, $F_p = 24$, we start with an a-priori candidate solution with all coefficients zero (the controller performs FWL arithmetic, hence we use $\tilde{C}(z)$):

$$\tilde{C}(z) = \frac{0z^2 + 0z + 0}{0z^2 + 0z + 0}.$$

In the first VERIFY stage, the UNCERTAINTY check finds the following counterexample:

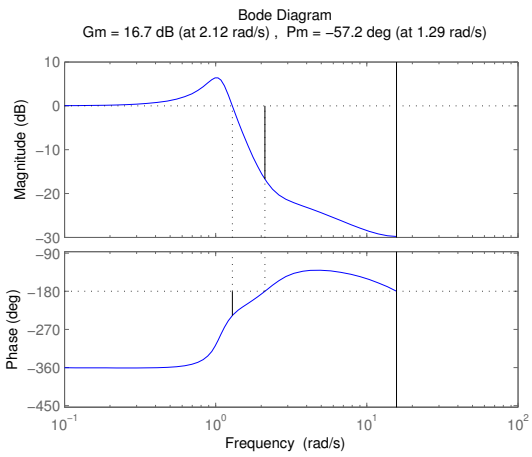
$$\tilde{G}(z) = \frac{0.026506}{1.000610z + 1.002838}.$$

We add this counterexample to INPUTS and initiate the SYNTHESIZE phase, where we obtain the following candidate solution:

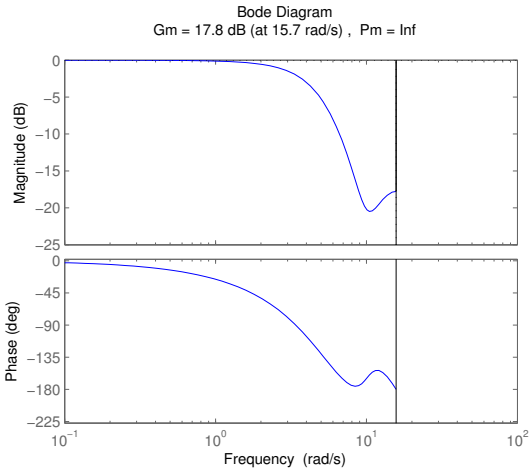
$$\tilde{C}(z) = \frac{12.402664z^2 - 11.439667z + 0.596756}{4.003906z^2 - 0.287949z + 0.015625}.$$

This time, the UNCERTAINTY check does not find any counterexample and we pass the current candidate solution to the PRECISION verification stage. We obtain the result *false*, meaning that the current precision is insufficient. Consequently, we increase our precision to $I_p = 20$, $F_p = 28$. Since the previous counterexamples were obtained at lower precision, we remove them from the set of counterexamples. Back in the SYNTHESIZE phase, we re-start the process with a candidate solution with all coefficients 0, as above. Next, the UNCERTAINTY verification stage provides the first counterexample at higher precision:

$$\tilde{G}(z) = \frac{0.026314}{0.999024z - 1.004785}.$$



(a) Original controller [36]



(b) Controller synthesized by DSSynth

Figure 5: Bode diagram for original controller in [36] and for newly synthesized closed-loop system

In the SYNTHESIZE phase, we get a new candidate solution that eliminates the new, higher precision counterexample:

$$\tilde{C}(z) = \frac{11.035202z^2+5.846100z+4.901855}{1.097901z^2+0.063110z+0.128357}.$$

This candidate solution is validated as the final solution by both stages UNCERTAINTY and PRECISION in the VERIFY phase. Figure 5 compares the Bode diagram using the digital controller represented by Eq. (18) from [36] (Figure 5a) and the final candidate solution from our synthesizer (Figure 5b). The DSSynth final solution is stable since it presents an infinite phase margin and a gain margin of 17.8 dB.

Figure 6 illustrates the step responses of the closed-loop system with the original controller represented by Eq. (18) (Figure 6a), the first (Figure 6b) and final (Figure 6c) candidate solutions provided by DSSynth. The step response in Figure 6a confirms the stability loss if we consider FWL effects. Figure 6b shows that the first candidate controller is able to stabilize the closed-loop system without uncertainties, but it is rejected during the PRECISION phase by DSSynth since this solution is not sound. Finally, Figure 6c shows a stable behavior for the final (sound) solution, which presents a lower settling time (hence the digitization effects).

4. EXPERIMENTAL EVALUATION

4.1 Description of the Benchmarks

The first set of benchmarks uses the discrete model G_1 of a cruise control system for a car, and accounts for rolling friction, aerodynamic drag, and the gravitational disturbance force [5]. The second set of benchmarks considers the discrete model G_2 of a simple spring-mass damper plant [36]. A third set of benchmarks uses the discrete model G_3 for satellite attitude dynamics [17], which require attitude control for orientation of antennas and sensors w.r.t. Earth. The fourth set of benchmarks presents an alternative discrete model G_4 of a cruise control system [36]. The fifth and sixth set of benchmarks describe the discrete model of a DC servo motor velocity dynamics [27, 35]. The seventh set of benchmarks contains a well-studied discrete non-minimal phase model G_7. Non-minimal phase models cause additional difficulties for the design of stable controllers [12]. The eighth set of benchmarks describes the discrete model G_8 for the *Helicopter Longitudinal Motion*, which provides the longitudinal motion dynamics of a helicopter [17]. The ninth set of benchmarks contains the discrete model G_9 for the known *Inverted Pendulum*, which describes a pendulum dynamics with its center of mass above its pivot point [17]. The tenth set of benchmarks contains the *Magnetic Suspension* discrete model G_{10}, which describes the dynamics of a mass that levitates with support only of a magnetic field [17]. The eleventh set of benchmarks contains the *Computer Tape Driver* discrete model G_{11}, which describes a system to read and write data on a storage device [17]. The last set of benchmarks considers a discrete model G_{12} that is typically used for evaluating stability margins and controller fragility [23, 24].

Additional benchmarks were created for the *Cruise Control System*, *Spring-mass damper*, and *Satellite* considering parametric additive in the nominal plant model (represented by $\Delta_p\vec{G}$ in Eq. (13)). The uncertainties are deviations bounded to a maximum magnitude of 0.5 in each coefficient. These uncertain models are respectively represented by G_{1b}, G_{2b}, G_{3b} and G_{3d}.

All experiments have been conducted on a 12-core 2.40 GHz Intel Xeon E5-2440 with 96 GB of RAM and Linux OS. All times given are wall clock times in seconds, as measured by the UNIX date command. For the two-stage verification engine in Figure 4 we have applied a timeout of 8 hours per benchmark, whereas 24 hours have been set for the approach using a one-stage engine.

4.2 Objectives

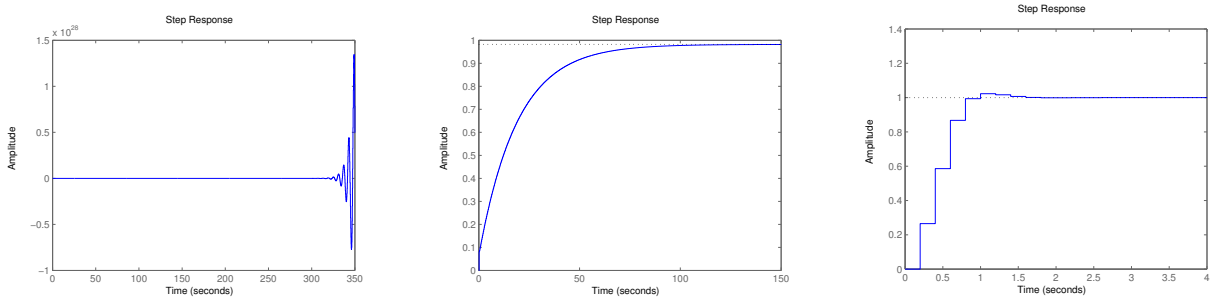
Using the closed-loop control system benchmarks given in Section 4.1, our experimental evaluation aims to answer two research questions:

RQ1 **(performance)** does the CEGIS approach generate a FWL digital controller in a reasonable amount of time?

RQ2 **(sanity check)** are the synthesized controllers sound and can their stability be confirmed outside of our model?

4.3 Results

We give the run-times required to synthesize a stable controller for each benchmark in Table 1. Here, *Plant* is the discrete or continuous plant model, *Benchmark* is the name of the employed benchmark, I and F represent the



(a) Original controller (b) First solution by DSSynth (c) Final solution by DSSynth

Figure 6: Step responses for original [36] closed-loop system with FWL effects and for each SYNTHESIZE iteration of DSSynth

number of integer and fractional bits of the stable controller, respectively, while the two right columns display the total time (in seconds) required to synthesize a stable controller for the given plant.

For the majority of the benchmarks, the conjecture explained in Section 3.3 holds and the two-stage verification engine is able to find a stable solution in less than one minute for half of the benchmarks. This is possible if the inductive solutions need to be refined with few counterexamples and increments of the fixed-point precision. However, the benchmark *SatelliteB2* with uncertainty (G_{3b}) has required too many counterexamples to refine its solution. For this particular case, the one-stage engine is able to complement the two-stage approach and synthesizes a solution. It is important to reiterate that the one-stage verification engine does not take advantage of the inductive conjecture inherent to CEGIS, but instead fully explores the counterexample space in a single SAT instance. As expected, this approach is significantly slower on average and is only useful for benchmarks where the CEGIS approach requires too many refinement iterations such that exploring all counterexamples in a single SAT instance performs better. Our results suggest an average performance difference of at least two orders of magnitude, leading to the one-stage engine timing out on the majority of our benchmarks. Table 1 lists the results for both engines, where in 16 out of 23 benchmarks, the two-stage engine is faster.

The presence of uncertainty in some particular benchmarks (2, 4, 6, and 8) leads to harder verification conditions to be checked by the VERIFY phase, which impacts the overall synthesis time. However, considering the faster engine for each benchmark (marked in bold in Table 1), the median run-time is 48 s, implying that DSSynth can synthesize half of the controllers in less than one minute. Overall, the average fastest synthesis time considering both engines is approximately 42 minutes. We consider these times short enough to be of practical use to control engineers, and thus affirm RQ1. We further observe that the two-stage verification engine is able to synthesize stable controllers for 19 out of the 23 benchmarks, and can be complemented using the one-stage engine, which is faster for two benchmarks where the inductive conjectures fail. Both verification engines together enable controller synthesis for 20 out of 23 benchmarks. For the remaining benchmarks our approach failed to synthesize a stable controller within the time limits. This can be addressed by either increasing either the time limit or the fixed-point word widths considered, or by using floating-point arithmetic instead. The synthesized controllers have been

#	Plant	Benchmark	I	F	2-stage	1-stage
1	G_{1a}	CruiseControl02	4	16	**12 s**	67 s
2	G_{1b}	CruiseControl02†	4	16	14600 s	**52 s**
3	G_{2a}	SpgMsDamper	15	16	**52 s**	318 s
4	G_{2b}	SpgMsDamper†	15	16	✗	✗
5	G_{3a}	SatelliteB2	3	7	**36 s**	✗
6	G_{3b}	SatelliteB2†	3	7	✗	**4111 s**
7	G_{3c}	SatelliteC2	3	5	**3 s**	205 s
8	G_{3d}	SatelliteC2†	3	5	**50 s**	1315 s
9	G_4	Cruise	3	7	1 s	1 s
10	G_5	DCMotor	3	7	**1 s**	10 s
11	G_6	DCServomotor	4	11	**46 s**	✗
12	G_7	Doyleetal	4	11	**8769 s**	✗
13	G_8	Helicopter	3	7	**44 s**	✗
14	G_9	Pendulum	3	7	**1 s**	14826 s
15	G_{10}	Suspension	3	7	**1 s**	5 s
16	G_{11}	Tapedriver	3	7	1 s	1 s
17	G_{12a}	a_ST1_IMPL1	16	4	**11748 s**	✗
18	G_{12a}	a_ST1_IMPL2	16	8	**351 s**	✗
19	G_{12a}	a_ST1_IMPL3	16	12	**8772 s**	✗
20	G_{12b}	a_ST2_IMPL1	16	4	**1128 s**	✗
21	G_{12b}	a_ST2_IMPL2	16	8	✗	✗
22	G_{12b}	a_ST2_IMPL3	16	12	**15183 s**	✗
23	G_{12c}	a_ST3_IMPL1	16	4	✗	✗

Table 1: DSSynth results (✗ = time-out, † = uncertainty)

confirmed to be stable outside of our model representation using MATLAB, positively answering RQ2. A link to the full experimental environment, including scripts to reproduce the results, all benchmarks and the DSSynth tool, is provided in the footnote.[1]

4.4 Threats to Validity

We have reported a favorable assessment of DSSynth over a diverse set of real-world benchmarks. Nevertheless, this set of benchmarks is limited within the scope of this paper and DSSynth's performance needs to be assessed on a larger benchmark set in future work.

Furthermore, our approach to select suitable FWL word widths to model plant behavior employs a heuristic based on user-provided controller word-width specifications. Given the encouraging results of our benchmarks, this heuristic appears to be strong enough for the current benchmark set, but this may not generalize. Further experiments towards determining suitable plant FWL configurations may thus be necessary in future work.

Finally, the experimental results obtained using DSSynth for stability properties may not generalize to other properties.

[1] http://www.cprover.org/DSSynth/experiment.tar.gz
CBMC (SHA-1 hash) version:
7a6cec1dd0eb8843559591105235f1f2c4678801

The inductive nature of the two-stage back-end of DSSynth increases performance significantly compared to the one-stage back-end, but this performance benefit introduced by CEGIS inductive generalizations may not be observed for other controller properties. Additional experiments are necessary to confirm that the performance of our inductive synthesis approach can be leveraged in those scenarios.

5. RELATED WORK

Robust Synthesis of Linear Systems.

The problem of parametric control synthesis based on stability measures for continuous Linear Time Invariant (LTI) Single Input-Single Output (SISO) systems has been researched for several decades. On a theoretical level it is a solved problem [37], for which researchers continuously seek better results for a number of aspects in addition to stability. A vast range of pole placement techniques such as Moore's algorithm for eigenstructure assignment [25] or the more recent Linear Quadratic Regulator (LQR) [6] have been used with increasing degrees of success. The latter approach highlights the importance of conserving energy during the control process, which results in lower running costs. Since real systems are subject to tolerance and noise as well as the need for economy, more recent studies focus on the problem of achieving robust stability with minimum gain [33, 26]. However, when applied with the aim of synthesizing a digital controller, many of these techniques lack the ability to produce sound or stable results because they disregard the effects of quantization and rounding. Recent papers on implementations/synthesis of LTI digital controllers [10, 18] focus on time discretization, failing to account for these error-inducing effects and can result in digital systems that are unstable even though they have been proven to be robustly stable in a continuous space.

Formal Verification of Linear Digital Controllers.

Various effects of discretizing dynamics, including delayed response [13] and Finite Word Length (FWL) semantics [3] have been studied, with the goal to either verify [7] or to optimize [29] given implementations.

There are two different problems that arise from FWL semantics. The first is the error in the dynamics caused by the inability to represent the exact state of the physical system while the second relates to rounding errors during computation. In [16], a stability measure based on the error of the digital dynamics ensures that the deviation introduced by FWL does not make the digital system unstable. A recent approach [38] uses the μ-calculus to directly model the digital controller so that the selected parameters are stable by design. Most work in verification focuses on finding a correct variant of a known controller, looking for optimal parameter representations using FWL, but ignore the effects of rounding errors due to issues of mathematical tractability. The analyses in [32, 36] rely on an invariant computation on the discrete system dynamics using Semi-Definite Programming (SDP). While the former uses BIBO properties to determine stability, the latter uses Lyapunov-based quadratic invariants. In both cases, the SDP solver uses floating-point arithmetic and soundness is checked by bounding the error. An alternative approach is taken by [30], where the verification of existing code is performed against a known model by extracting an LTI model of the code through symbolic execution. In order to account for rounding errors, an upper bound is introduced in the verification phase. If the error of the implementation is lower than this tolerance level, then the verification is successful.

Robust Synthesis of FWL Digital Controllers.

There is no technique in the existing literature for automatic synthesis of fixed-point digital controllers that considers FWL effects.

Other tools such as [14] are aimed at robust stability problems, but they fail to take the FWL effects into account. In order to provide a correct-by-design digital controller, [2] requires a user-defined finite-state abstraction to synthesize a digital controller based on high-level specifications. While this approach overcomes the challenges presented by the FWL problem, it still requires error-prone user intervention. A different solution that uses FWL as the starting point is an approach that synthesizes word lengths for known control problems [22]; however, this provides neither an optimal result nor a comprehensive solution for the problem.

The CEGIS Architecture.

Program synthesis is the problem of computing correct-by-design programs from high-level specifications, and algorithms for this problem have made substantial progress in recent years. One such approach [21] inductively synthesizes invariants to generate the desired programs.

Program synthesizers are an ideal fit for synthesis of parametric controllers since the semantics of programs capture effects such as FWL precisely. In [31], the authors use CEGIS for the synthesis of switching controllers for stabilizing continuous-time plants with polynomial dynamics. The work extends to its application on affine systems, finding its major challenge in the hardness of solving linear arithmetic with the state-of-the-art SMT solvers. Since this approach uses switching states instead of linear dynamics in the digital controller, it entirely circumvents the FWL problem. It is also not suitable for the kind of control we seek to synthesize. We require a combination of a synthesis engine with a control verification tool that addresses the challenges presented here in the form of FWL effects and stability measures for LTI SISO controllers. We take the former from [11] and the latter from [7] while enhancing the procedure by evaluating the quantization effects of the Hardware interfaces (ADC/DAC) to obtain an accurate discrete-time FWL representation of the continuous dynamics.

6. CONCLUSIONS

We have presented a method for synthesizing stable controllers and an implementation in a tool called DSSynth. The novelty in our approach is that it is fully automated and algorithmically and numerically sound. In particular, DSSynth marks the first use of the CEGIS that handles plants with uncertain models and FWL effects over the digital controller. Implementing this architecture requires transforming the traditional CEGIS refinement loop into a two-stage engine: here, the first stage performs fast, but potentially unsound fixed-point operations, whereas the second stage restores soundness by validating the operations performed by the first stage using interval arithmetic. Our experimental results show that DSSynth is able to synthesize stable controllers for most benchmarks within a reasonable amount of time fully automatically. Future work will be the extension of this

CEGIS-based approach to further classes of systems, including those with state space. We will also consider performance requirements while synthesizing the digital controller.

7. REFERENCES

[1] R. Alur, D. Fisman, R. Singh, and A. Solar-Lezama. SyGuS-Comp 2016: Results and analysis. In *Workshop on Synthesis*, volume 229 of *EPTCS*, 2016.

[2] R. Alur, S. Moarref, and U. Topcu. Compositional synthesis with parametric reactive controllers. In *HSCC*. ACM, 2016.

[3] A. Anta, R. Majumdar, I. Saha, and P. Tabuada. Automatic verification of control system implementations. In *Embedded Software (EMSOFT)*, pages 9–18, 2010.

[4] K. Åström and B. Wittenmark. *Computer-controlled systems: theory and design*. 1997.

[5] K. J. Astrom and R. M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. 2008.

[6] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1), 2002.

[7] I. Bessa, H. Ismail, L. Cordeiro, and J. Filho. Verification of fixed-point digital controllers using direct and delta forms realizations. *Design Autom. for Emb. Sys.*, 20(2), 2016.

[8] I. Bessa, H. Ismail, R. Palhares, L. Cordeiro, and J. E. C. Filho. Formal non-fragile stability verification of digital control systems with uncertainty. *IEEE Transactions on Computers*, 66(3):545–552, 2017.

[9] E. M. Clarke, D. Kroening, and F. Lerda. A tool for checking ANSI-C programs. In *TACAS*, volume 2988, 2004.

[10] S. Das, I. Pan, K. Halder, S. Das, and A. Gupta. LQR based improved discrete PID controller design via optimum selection of weighting matrices using fractional order integral performance index. *Applied Mathematical Modelling*, 37(6), 2013.

[11] C. David, D. Kroening, and M. Lewis. Using program synthesis for program analysis. In *LPAR*, LNCS, 2015.

[12] J. C. Doyle, B. A. Francis, and A. R. Tannenbaum. *Feedback Control Theory*. 1991.

[13] P. S. Duggirala and M. Viswanathan. Analyzing real time linear control systems using software verification. In *IEEE Real-Time Systems Symposium*, Dec 2015.

[14] C. Economakos, G. Economakos, M. Skarpetis, and M. Tzamtzi. Automated synthesis of an FPGA-based controller for vehicle lateral control. In *MATEC Web of Conferences*, volume 41, 2016.

[15] S. Fadali and A. Visioli. *Digital Control Engineering: Analysis and Design*, volume 303 of *Electronics & Electrical*. 2009.

[16] I. J. Fialho and T. T. Georgiou. On stability and performance of sampled-data systems subject to wordlength constraint. *IEEE Trans. on Automatic Control*, 39(12), 1994.

[17] G. Franklin, D. Powell, and A. Emami-Naeini. *Feedback Control of Dynamic Systems*. 7th edition, 2015.

[18] S. Ghosh, R. K. Barai, S. Bhattarcharya, P. Bhattacharyya, S. Rudra, A. Dutta, and R. Pyne. An FPGA based implementation of a flexible digital PID controller for a motion control system. In *Computer Communication and Informatics (ICCCI)*. IEEE, 2013.

[19] H. Ismail, I. Bessa, L. C. Cordeiro, E. B. de Lima Filho, and J. E. C. Filho. DSVerifier: A bounded model checking tool for digital systems. In *SPIN*, volume 9232, 2015.

[20] R. Istepanian and J. F. Whidborne. *Digital controller implementation and fragility: A modern perspective*. 2012.

[21] S. Itzhaky, S. Gulwani, N. Immerman, and M. Sagiv. A simple inductive synthesis methodology and its applications. In *ACM Sigplan Notices*, volume 45. ACM, 2010.

[22] S. Jha and S. A. Seshia. SWATI: Synthesizing wordlengths automatically using testing and induction. *arXiv preprint arXiv:1302.1920*, 2013.

[23] L. Keel and S. Bhattacharyya. Robust, fragile, or optimal? *IEEE Trans. on Automatic Control*, 42(8), 1997.

[24] L. Keel and S. Bhattacharyya. Stability margins and digital implementation of controllers. In *Proc. American Control Conference*, volume 5, 1998.

[25] G. Klein and B. Moore. Eigenvalue-generalized eigenvector assignment with state feedback. *IEEE Trans. on Automatic Control*, 22(1), 1977.

[26] U. Konigorski. Pole placement by parametric output feedback. *Systems & Control Letters*, 61(2), 2012.

[27] Y. Li, K. Ang, G. Chong, W. Feng, K. Tan, and H. Kashiwagi. CAutoCSD–evolutionary search and optimisation enabled computer automated control system design. *Int J Automat Comput*, 1(1), 2004.

[28] R. E. Moore. *Interval analysis*, volume 4. 1966.

[29] A. K. Oudjida, N. Chaillet, A. Liacha, M. L. Berrandjia, and M. Hamerlain. Design of high-speed and low-power finite-word-length PID controllers. *Control Theory and Technology*, 12(1), 2014.

[30] J. Park, M. Pajic, I. Lee, and O. Sokolsky. Scalable verification of linear controller software. In *TACAS*. Springer, 2016.

[31] H. Ravanbakhsh and S. Sankaranarayanan. Counter-example guided synthesis of control Lyapunov functions for switched systems. In *Conference on Decision and Control, CDC*, 2015.

[32] P. Roux, R. Jobredeaux, and P. Garoche. Closed loop analysis of control command software. In *HSCC*, 2015.

[33] R. Schmid, L. Ntogramatzidis, T. Nguyen, and A. Pandey. A unified method for optimal arbitrary pole placement. *Automatica*, 50(8), 2014.

[34] A. Solar-Lezama. Program sketching. *STTT*, 15(5-6), 2013.

[35] K. Tan and Y. Li. Performance-based control system design automation via evolutionary computing. *Engineering Applications of Artificial Intelligence*, 14(4), 2001.

[36] T. E. Wang, P. Garoche, P. Roux, R. Jobredeaux, and E. Feron. Formal analysis of robustness at model and code level. In *HSCC*, 2016.

[37] W. Wonham. On pole assignment in multi-input controllable linear systems. *IEEE Trans. on Automatic Control*, 12(6), 1967.

[38] J. Wu, G. Li, S. Chen, and J. Chu. Robust finite word length controller design. *Automatica*, 45(12), 2009.