Abstract Interpretation for Block Diagrams - Two Case Studies

Christian Dernehl
Informatik 11 Embedded Software
RWTH Aachen University
Aachen, Germany
dernehl@embedded.rwth-aachen.de

Jan Kühn
Informatik 11 Embedded Software
RWTH Aachen University
Aachen, Germany
kuehn@embedded.rwth-aachen.de

Stefan Kowalewski
Informatik 11 Embedded Software
RWTH Aachen University
Aachen, Germany
kowalewski@embedded.rwth-aachen.de

***Abstract*—Model based development is increasingly used in embedded systems, which are often deployed in a safety critical environment. Verification techniques, supporting the development process can not only increase safety, but also help to speed up the process. In many cases models are designed with block diagrams, assisting rapid prototyping. Instead of focusing on code, we apply abstract interpretation to models consisting of block diagrams. In addition to a value range analysis, we propose the computation of bounds for the rate of change for each signal, which can be used to check rate requirements automatically, which arise from physical constraints of the environment. We evaluate our work in two case studies from ongoing research projects. The first case study is from the medical domain, while the second example is a drone. Both systems are also verified with a commercial verification tool, highlighting benefits of the tool and our implementation.**

I. Introduction

In recent years, digitalization of control has emerged and software plays an increasingly role for designing embedded systems. More flexibility than hard wired layouts and rapid prototyping are a few reasons for preferring software solutions. However, questions about safety must also be considered. Various norms, such as IEC-61511, 61513, 62279, 62061, ISO-26262, DO-178C, to name a few, specify how safety critical software systems must be developed. In many cases testing is preferred to validate against requirements, however, formal methods become increasingly popular. Automatic verification techniques, which are applicable to large scale systems, pose a challenge due to large computational overhead. Nonetheless, static methods, such as abstract interpretation have been applied to large scale industry models [1], [2]. Abstract interpretation aids the developer to identify potential errors and, what is even more important, can prove absence of defects. Nevertheless, to our knowledge, there is currently no solution, which focuses on the rates of change for signals. In practice, rates of change play an important role in embedded environments, where sensors, controllers and actuators have rate limitations.

We propose in this work a way to extend our previously introduced static value range analysis [3] by including abstractions for rates of change, so that requirements about the rate of change can be verified automatically. Since our method also calculates bounds on the rate of change for system internals, limitations for individual parts can be analyzed, for instance a nonlinear controller, which may only operate in certain frequency ranges. Finally, providing information about rates gives the developer a better understanding of the system, which is vital in large scale projects. Our static analysis technique is part of the *artshop* framework [4], which contains a model repository, tool adapters and analysis plugins.

a) Abstract Interpretation: Static methods may perform style checks, analyze code structure, identify desired and undesired patterns, but can also be used to compute bounds on variables and calculate the call graph. With this data, certain non functional requirements can be fulfilled by ruling out errors, such as division by zero, under- and overflows, unreachable code or invalid function calls, such as `sqrt(-1)`. One way to compute the necessary data is *abstract interpretation* [5], which abstracts a program and proves properties in the abstract domain, which can be transfered to properties of the original program. In detail, states are abstracted by a domain, such as intervals, yielding potentially an over approximation, but also providing guarantees on the bounds of variables for a given program location. Termination of the algorithm is proven by widening, i.e. over approximating program states, so that a fix point within the abstract domain can be found. Although intervals can be used as abstract domain, others can be integrated such as relational domains [6], congruences [7], digital filters [8] and more.

b) Block Diagrams: With commercial tools available[1] for programming languages such as C/C++, abstract interpretation is used in industry. However, with growing complexity and integration of different disciplines in embedded systems, bridging knowledge is crucial in development processes, which may be supported by model based design methods [9]. With visual programming languages, complex software models tend to be easier understood by staff, who is not firm with classical programming languages. Model based development is supported by various commercial tools, among them Simulink[2] and SCADE[3]. Both provide a user interface to design block diagrams by connecting blocks with arrows, so that thebehav-

[1]Such as Astree, Frama-C, Goanna, Lint, PolySpace and more.
[2]See http://www.mathworks.com/products/simulink/ .
[3]See http://www.esterel-technologies.com/products/scade-suite/ .

ior is described by a data flow from sources to sinks through the model. While blocks model operations, users can construct systems by choosing from a given palette of basic blocks, so that complex designs are constructed in a bottom up fashion. Subsystems provide ways to construct hierarchical models, making comprehension easier.

A block may either define memoryless time invariant functions, or have internal states to model dynamics, for example, integrators or transfer function blocks in Simulink. Contrary to SCADE, which is depending on discrete time, Simulink provides also ways to specify continuous system, making the design of hybrid systems possible. Though this is an advantage for modeling purposes, continuous time models depend on a solver, which can either be variable size or fixed step size. These solvers are in many cases a Runge-Kutta variant to solve the resulting differential equations. Nevertheless, code generation for continuous time models is limited in many cases to fixed step solvers.

Generated code can be verified with the presented methods from abstract interpretation, however, if errors are found, these must also be corrected in the block diagram, since a regeneration of code reproduces the error. To ease this process, we have proposed in previous work to apply abstract interpretation with interval sets on block diagrams, providing the developer with valuable notifications about potential errors [3].

c) Applications: Bounds on rates of change may be used for a variety of applications, as they bound systems by a linear increase. With a given specification on rates for signals, our analysis can automatically checks if these are fulfilled. Static or dynamic rate limiter blocks[4] from the standard Simulink block set allow the user to specify bounds on the output rate. Our technique can automatically verify, whether the placement of such a block is necessary and provide the quantity of the limitation.

Rates of change can also help the user to identify potential mistakes in the model. Combined with rates, static analysis can prove not only whether a condition becomes true or a switch is eventually triggered, but also can potentially provide a lower bound on the time of activation. For instance, consider a relational operator, which checks whether a signal is below a given bound. This information can be used in the value range analysis to rule out false positives. Additionally, with a limited rate of change and a user specified upper bound on the execution time, value intervals can be tightened.

d) Contribution: In this paper we extend our static analysis to compute rate of change bounds on the signals and states of a block diagram. We show how a value range analysis can be combined to improve rate of change bounds and present our results in two case studies.

In the following Section, we highlight related research findings regarding software verification in general and applications to block diagrams in particular. Afterwards, in Section III we present how abstract interpretation can be used to compute bounds on rates. Eventually, we evaluate our solution in two larger case studies, where the first is a medical device and the second a drone project.

[4]See http://www.mathworks.com/help/simulink/slref/ ratelimiter.html .

II. Related Work

Verification of software system has been an important topic in recent decades. We have already introduced abstract interpretation in the introduction, so this section focuses on other verification techniques.

a) Software Verification: Model checking [10] is a technique to verify whether a model satisfies a given formal specification. Model checker implementations, such as SPIN [11] or UPPAAL [12] require the specifications to be in a (extended) temporal logic, such as linear temporal logic or computation tree logic. Though if a model does not fulfill the requirement, a counter example is shown to the user. Although model checking can be used to verify bounds on the rate of change of variables, our solution is based on abstract interpretation.

Another approach to verification is to use symbols to express and verify properties. Implementations such as NuSMV [13], [14] or the newer nuXmv [15] have been proven useful for many applications [16], [17]. As symbolic solvers play an important role, a variety of sat modulo theory (SMT) solvers [18]–[20] are also available. With SMT solvers, users can rewrite their model into a set of logic expressions, which can be solved, while the results can be used to prove or disprove properties of the model. Unlike symbolic representations, our concepts is based on intervals for rate of change.

b) Block Diagram Verification: Block diagrams can be verified either by translation into a formal format or by interpreting the model. Tripakis et al. [21] translate discrete Simulink models to Lustre, which can be used for model validation. With graph transformations, Agrawal el al. [22] transformed systems in Simulink to hybrid automata, so that verification techniques for hybrid automata can be applied.

A similar solution to abstract interpretation of block diagrams has been carried out by Chapoutot et al. [23], where the authors define formal semantics for the continuous solver. Furthermore Chapoutot et al. use abstractions o investigate the behavior of continuous solvers and resulting floating point errors [24], [25]. Reicherdt and Glesner applied the Boogie verification tool to verify Simulink models [26] with a SMT solver [18]. Unlike the presented work, our concept focuses on rates of change for discrete models.

In previous work [3], we have demonstrated how abstract interpretation can be adapted to block diagrams. We have used interval sets, based on interval arithmetic [27] to abstract signal values and internal states of the model.

III. Model Verification

With related findings presented, we focus on our proposal for model verification. In a first step, assumptions and limitations of our approach are shown. Afterwards, we demonstrate how block diagrams can be abstracted for verification and present ideas to calculate rates for each signal.

A. Model Assumptions

In our current implementation, we support over 60 different blocks from the standard Simulink block set[5]. We presume that blocks, which are not supported by out tool, operate locally, i.e. an unsupported block does not affect other blocks. The value range and intervals for the rate of change for the outputs of unsupported blocks is chosen to be a set, which over approximates all possible values, e.g. $\{[-\infty, \infty]\}$. Nevertheless, we presume, that unsupported block do not change the behavior of other unrelated parts in the model, i.e. that the block operates locally.

Blocks may either perform memoryless operations or linear computations with memory. In addition, the output of blocks at time step $k \in \mathbb{N}$ may either depend only on the internal on also requires the input. Those blocks, which are dependent on the input at the same time step are called *feed through* blocks, which includes arithmetic operations, for example.

With our given assumptions, loops can be constructed within the block diagram, which is used for instance to design Feedback control systems. An *algebraic loop* is a finite sequence of feed through blocks $b_1, b_2, \ldots, b_n$, so that b_i, $i \in \mathbb{N}$, $2 \leq i \leq n$, is a block for which the output can only be computed depending on the input of b_{i-1} and the input of b_1 depends on the output of b_n at the same time step. Mathematically, this poses a problem, since for a given time step, a signal can have two distinct values. Algebraic loops can be solved by introducing a block with an internal state, however, this changes also the behavior. In our work, we assume, that models do not contains algebraic loops. We believe this to be a valid assumption, since code generation is not supported for such models.

For the Simulink block diagram, we assume that a fixed step solver is used, i.e. we rule out variable step solvers. However, for our and other applications, this is no limitation, since variable size solvers are not recommended for code generation. Still, continuous blocks can be used with a fixed step solver. Hence, our algorithm abstracts the ordinary differential equation solver, which is configured for the model, similar to the work presented by Bouissou and Chapoutot [23]. Depending on the embedded target, minor steps might be executed on the desktop machine, but not on the embedded system, yielding inconsistent behavior for continuous systems. Since our technique assumes no minor steps, we issue a warning if the model contains continuous blocks. Although we have designed abstractions for many signal types, signals containing a complex, i.e. real and imaginary part, are not supported. Thus, a negative real input to a square root yields a *Not-a-Number*(NaN) float value.

B. Value Range Analysis

Much of the work of the value range analysis for block diagrams has been presented previously [3], therefore, we focus in this paper on rates of change and provide only a brief introduction and reference to our previous work. During execution, the state of a block diagram depends on the signals at each port and the internal states of blocks. Therefore, we define a map, which links ports and internal states to variables $\mathcal{V}$. Formally, a the set of all states σ of a block diagram is defined by the set Σ,

$$\Sigma := \{\sigma | \sigma : \mathcal{V} \to \mathbb{R}^\infty\} \tag{1}$$

where $\mathbb{R}^\infty := \mathbb{R} \cup \{\pm\infty\}$. Note, for this work, we use a real number abstraction for all scalars within a block diagram, since we need properties, such as associativity, which are not available when using IEEE-754 floating point numbers. We can use our previous analysis [3] to prove for a given model, that the NaN symbol does not occur and assume the absence of NaN for the rest of the paper.

a) Scalars: Simulink provides a variety of data types, such as integer types, `uint8`, `uint16`, `uint32`, `int8`, `int16`, `int32`, which are either signed or unsigned and can be 8,16, or 32 bits long. Additionally, a boolean type `bool` is provided. Regarding floating point types, there are two variants, either 32 or 64 bits, i.e. `float32`, `float64`. For each integer scalar, a value set with the corresponding type boundaries is used.

b) Custom Types: In addition to scalars, signals can also have composite types, such as matrices or buses[6], which are similar to structures in the C programming language. In matrices, all entries are of the same data type, while buses can be used may group signals of different types and dimensions together. For instance, a 2x2 matrix of type `uint8` and a `float64` scalar might be combined to a bus. Since buses are signals themselves, a bus may contain other buses, hence, hierarchical signal structures can be designed. For our algorithm, we implemented matrices and buses containing abstract signals and scalars.

c) Operations: We have defined abstract operations for arithmetic operations, such as +,-,*,/, trigonometric functions sin, cos, tan, arcsin, arccos, arctan, exponential and logarithmic, square root, power and modulo. In addition, rounding operations, logical and relational operators are abstracted. Furthermore, for each cast towards a primitive data type, abstract casting operations are implemented.

Expressions can be constructed from constants, variables and the listed operations. For example $\sin(3 * 2 * x + 3 * y) - x * x$ is an expression. The valuation function $\mathrm{val}_\sigma : \mathit{Expr} \times \Sigma \to \mathbb{R}^\infty$ maps an expression and state to $\mathbb{R}^\infty$ and describes the semantics of the constructed expressions. For constants c, variables $v \in \mathcal{V}$, unary ($\blacklozenge$) and binary operations ($\lozenge$) the semantics are the following.

$$\mathrm{val}_\sigma(c) = c \qquad \mathrm{val}_\sigma(\blacklozenge e) = \blacklozenge \, \mathrm{val}_\sigma(e)$$
$$\mathrm{val}_\sigma(v) = \sigma(v) \qquad \mathrm{val}_\sigma(e_1 \lozenge e_2) = \mathrm{val}_\sigma(e_1) \lozenge \mathrm{val}_\sigma(e_2)$$

For example, $\mathrm{val}_\sigma\,(v_1 + v_2 + c_1)$ evaluates to $\sigma(v_1) + \sigma(v_2) + c_1$, assuming variables $v_1, v_2 \in \mathcal{V}$ and constant c_1. Note, that unary operations include minus, trigonometric functions, exponential, logarithm, and type casting, while binary operations

[5]See http://mathworks.com/help/simulink/blocklist.html .

[6]See http://mathworks.com/help/simulink/slref/buscreator.html .

include arithmetic $(+,-,*,/)$, logical $(\neg,\wedge,\vee)$, relational $(>,<,>=,<=,==,\sim=)$, power and modulo.

With matrices in mind, operations are applied element wise[7] and can be mapped to a set of scalars semantics.

$$\mathrm{val}_\sigma\left(\begin{bmatrix} e_{11} & e_{12} & \dots e_{1n} \\ e_{21} & e_{22} & \dots e_{2n} \\ \vdots & \vdots & \vdots \\ e_{m1} & e_{m2} & \dots e_{mn} \end{bmatrix}\right) := \begin{bmatrix} \mathrm{val}_\sigma(e_{11}) & \mathrm{val}_\sigma(e_{12}) & \dots \mathrm{val}_\sigma(e_{1n}) \\ \mathrm{val}_\sigma(e_{21}) & \mathrm{val}_\sigma(e_{22}) & \dots \mathrm{val}_\sigma(e_{2n}) \\ \vdots & \vdots & \vdots \\ \mathrm{val}_\sigma(e_{m1}) & \mathrm{val}_\sigma(e_{m2}) & \dots \mathrm{val}_\sigma(e_{mn}) \end{bmatrix} \tag{2}$$

For the abstract domain for scalars, we have chosen interval sets

$$\mathbb{IS}_{\mathbb{R}^\infty} : \mathcal{P}(\mathbb{R}^\infty \times \mathbb{R}^\infty)$$

over $\mathbb{R}^\infty$, which consist of multiple intervals. Unary $\blacklozenge_I$ and binary $\lozenge_I$ operations for intervals have already been explained in previous work, for example by Hickey [27]. Binary operations of two interval sets $IS_1 \in \mathbb{IS}_{\mathbb{R}^\infty}$ and $IS_2 \in \mathbb{IS}_{\mathbb{R}^\infty}$ are applied for each combination of intervals within IS_1 and IS_2.

$$IS_1 \lozenge^{\#} IS_2 := \bigcup_{\substack{I_1 \in IS_1 \\ I_2 \in IS_2}} I_1 \lozenge_I I_2 \tag{3}$$

Corresponding to the concrete valuation function, the abstract valuation function $Expr \times \{\rho | \rho : \mathcal{V} \to \mathbb{IS}_{\mathbb{R}}\} \to \mathbb{IS}_{\mathbb{R}}$ maps expressions to abstract values.

$$\mathrm{val}^{\#}_\rho(c) = \{[c,c]\} \qquad \mathrm{val}^{\#}_\rho(\blacklozenge e) = \blacklozenge^{\#}\,\mathrm{val}^{\#}_\rho(e)$$
$$\mathrm{val}^{\#}_\rho(v) = \rho(v) \qquad \mathrm{val}^{\#}_\rho(e_1 \lozenge e_2) = \mathrm{val}^{\#}_\rho(e_1) \lozenge^{\#} \mathrm{val}^{\#}_\rho(e_2)$$

For example, $\{[-7,-2],[6,9]\} +^{\#} \{[-3,-1],[1,3]\}$ yields the interval set $\{[-10,1],[3,12]\}$. The abstract valuation functions for value sets and rates of change of matrices are also applied element-wise, similar to Equation(2).

d) Blocks and Diagrams: With the given expressions, all standard Simulink blocks can be modeled. Using directed graphs, in which vertices and edges represent ports and lines, block diagrams can be represented, while a set of ports is assigned to a block. The *execution order* can be extracted from Simulink, which yields a set of block schedules. This schedule consists of execution contexts, forming a hierarchical graph, where elements on the same level can be computed independently. Leafs in the tree describe an ordered set, in which blocks must be executed.

During execution, a block has different phases. Some blocks compute the output depending on the current state and therefore do not rely on input immediately. Hence, in each step a block updates its internal state and compute an output, where the order depends on the block implementation. With the given operations, the state updates and output computations can be modeled. For example, an discrete time integrator block has an update equation, in which the state is changed by the input. This can be expressed by multiplication and addition operators.

[7]Except for special cases, such as matrix multiplication and inversion.

C. Abstract Interpretation with Rates

In this work, we extend our abstract interpretation with rates as abstract domain. Consider two consecutive states σ_{k+1} and σ_k with a step size of $\Delta t_k \in \mathbb{R}_{>0}$, then the rate of change for expression e is given by

$$\mathrm{val}_{(\sigma_{k+1},\sigma_k),R}(e) := \frac{\mathrm{val}_{\sigma_{k+1}}(e) - \mathrm{val}_{\sigma_k}(e)}{\Delta t_k}$$

With a sound abstraction for the value ranges, an over approximation of the rate abstraction can be made by incorporating the minimum and maximum value of an expression e.

$$\mathrm{val}^{\#}_{(\rho_{k+1},\rho_k),R}(e) \sqsubseteq_{\#} \left\{\left[-\frac{1}{\Delta t_k}, \frac{1}{\Delta t_k}\right]\right\} *^{\#} \left(\max_{\rho' \in \{\rho_k,\rho_{k+1}\}} \mathrm{val}^{\#}_{\rho'}(|e|) - \min_{\rho' \in \{\rho_k,\rho_{k+1}\}} \mathrm{val}^{\#}_{\rho'}(|e|)\right) \tag{4}$$

For example, if the value range for expression e is $[-1,1]$ in ρ_k and $[-5,3]$ in ρ_{k+1}, the rate can be over approximated safely by $[-9,9]$ if Δt_k is one.

We can apply this over approximation on all functions with a fixed bounded input, if the minimum and maximum is known in advance. In addition, time invariant memoryless nonlinear functions can be over approximated in that way, if a fixed point for the value set is found. Consider for example trigonometry, in particular $\sin$ or $\cos$, which are bounded by ± 1 for reals. Independent of the input, the output rate is limited by $[-2,2]$ in one time step. Blocks with bounded functions include saturation, modulo, remainder, min, max, logic operations and trigonometric functions, such as sin, arcsin, cos, arccos.

a) Linear Systems: Considering the standard Simulink block set, many blocks do not have an internal state and those which have a state describe a *linear system* performing a state transition from σ_k to σ_{k+1}

$$\mathrm{val}_{\sigma_{k+1}}(x) = \mathrm{val}_{\sigma_k}(Ax + Bu) \tag{5}$$
$$\mathrm{val}_{\sigma_k}(y) = \mathrm{val}_{\sigma_k}(Cx + Du) \tag{6}$$

with matrices A, B, C, D, input u, state x and output y vector for discrete time. As a result, rate of change computations must be defined for linear discrete time systems and nonlinear time invariant memoryless operations.

For a linear system, the output rate depends on the input value and rate ranges. Assume for simplicity $D = 0$, since the part where $D \neq 0$ can be extracted as a linear operation without a state, i.e. an offset bounded by the input value range. The solution of the recursive Equation (5) with respect to y from Equation (6) for any discrete time linear system with $D = 0$ and $\sigma_0(x) = 0$ can be expressed by

$$\mathrm{val}_{\sigma_N}(y) = \sum_{k=0}^{N} CA^kB \;\; \mathrm{val}_{\sigma_{N-k}}(u)$$

and

$$\mathrm{val}_{(\sigma_{k+1},\sigma_k),R}(y) = \sum_{k=0}^{N} CA^kB \;\; \mathrm{val}_{(\sigma_{N-k+1},\sigma_{N-k}),R}(u)$$

for rates of change respectively. Assuming $D \neq 0$ an additional gain must be added to the equations and if $\sigma_0(x) \neq 0$, the product $A^N \sigma_0(x)$ must be added. As a result, we can compute the output rate with the given formula for any linear block. Examples for the included block types are gain, sum, transfer function, state space. Note, that assuming $N \to \infty$, the sum exists only if all eigenvalues of A reside within the unit circle, i.e. $|\lambda_i(A)| < 1$ for each eigenvalue λ_i of A. In cases where $A = VDV^{-1}$ is diagonalizable, which is true for many control applications, a solution can be computed, which yields a closed form. Otherwise we draw no conclusions about the output rates. For special blocks, such as integrators, the output rate can be narrowed with the input value set if the input is constant or bounded. Although we have shown how linear operations can be treated, these results only hold if the system is truly linear, which is not the case if under- or overflows occur. Consequently, we add a check, based on information from the value range analysis, whether such nonlinearity can occur.

In case no under- or overflow occurs, suppose ρ' is a fix point for the rates on input variable u, so that

$$\mathrm{val}^{\#}_{(\rho_{N-k+1},\rho_{N-k}),R}(u) \sqsubseteq_{\#} \mathrm{val}^{\#}_{\rho',R}(u)$$

then an over approximation for all consecutive states can be given.

$$\mathrm{val}^{\#}_{(\rho_{k+1},\rho_k),R}(y) \sqsubseteq_{\#} ||C||_\infty \, ||B||_\infty \, ||V||_\infty \, ||V^{-1}||_\infty \sum_{k=0}^{\infty}{}^{\#} \, ||D||_\infty^k \; \mathrm{val}^{\#}_{(\rho'),R}(u)$$

Since we assume, that all eigenvalues of A reside within the unit circle, the infinite sum is a geometric sum and can be simplified to $1/(1 - \max_i \lambda_i(D))$, where $\lambda_i(D)$ is the i-th eigenvalue of the diagonal matrix D.

b) Casting Operations: Since our static analysis is type sensitive, rates are also calculated for type casts. In case all elements fit into the target data type the rates stay the same. This is always true, if the target data type is larger. If the cast is executed towards a smaller data type, the value interval set can be used to estimate, if an under- or overflow is possible. For under- and overflows, the rate interval is depended on the boundaries of the target type. Consider the `uint8` type, whose maximum value is 255. With Equation (4) the resulting rate interval is $[-255, 255]$.

c) Nonlinear Functions: For other functions, such as the square root, logarithm or exponential function, certain properties can be exploited by using the computed value range. Suppose a square root block has a valid input signal with values between $s \geq 0$ and e, then the input interval can be split at one. Due to the structure of the square root, for the first part, i.e. $[s, 1]$, the input rate is a lower bound, while for the second interval $[1, e]$ the input rate is an upper bound. Likewise bounds of the input rates can be used to improve the exponential, logarithm and power functions. A special cases arises for the clock signal, which represents the time duration. In this case the calculation simplifies to constant Δt_k.

d) Widening: Abstractions for rates can be used to compute abstract sets for each concrete state. However, termination, i.e. finding a fix point for rates, is not guaranteed. In our previous work, we have used widening of value ranges to guarantee termination of our algorithm. In case a fix point for the value range is found, but not for the rate of change, we use Equation (4) as a fix point, as this is a safe over approximation and we already have a fix point on the value range.

IV. Case Studies

After introducing a method to derive bounds on rates with abstract interpretation, we evaluate our solution with two models from ongoing research projects. Both systems have each more than 1000 blocks and in both cases, code is generated and compiled onto a target device, which is used for field applications. The first model is from the medical domain and is used as part of a controller in a networked intensive care setup. Although, timing constraints can be on a large scale compared to other fields of application, but can be very critical in such an application. In this example the model has to estimate the gas transfer in an extracorporeal blood circuit. Therefore it has to react to the incoming inputs in reasonable time to allow a safe and efficient control of the therapy to the actual state of the patient. The second model is from the aerospace domain and describes a one meter wingspan tilt-wing *unmanned aerial vehicle* (UAV). In this application, various tasks, such as take-off, cruise flight and landing are automated. However, the major complexity is drawn from the instability of the airframe concept, which is stabilized using a controller.

Table I lists all blocks, which have been used in both models. A major difference between both models is that the medical application is discrete, while the tilt-wing model uses continuous blocks. Due to the computation of attitude, trigonometric functions are inevitable for the tilt-wing model.

A. Medical Software Case Study

One of our case studies is a Simulink model consisting of more than 2000 Simulink Blocks which is part of ongoing research in a complex medical intensive care system. The system is an *extracorporeal lung assist system* (ECLA) applied to treat severe cases of the *acute respiratory distress syndrome* (ARDS) [28]. It consists of several connected medical devices acting as sensors or actuators and a dedicated control device. The given Simulink model is part of the therapy automation software on this control device (Figure 1). Its purpose can be summarized as model based estimation of a value difficult to be measured directly. The method in use is based on a Kalman filter, which is a common estimator in many other fields of application like navigation software. Basis is a grey box model of the real system which is described in detail in [28].

a) Model Description: As mentioned before some input values necessary for an exact calculation are difficult to measure and rely on expensive equipment with a high sensitivity regarding operating problems like sensor misplacement.

Block Type	ECLA	UAV	Block Type	ECLA	UAV	Block Type	ECLA	UAV
Abs	3	4	FromWorkspace	20	0	Relay	0	1
Bias	1	0	Gain	134	73	Rounding	3	0
BusAssignment	2	0	Goto	34	1	S-Function	4	4
BusCreator	32	16	Inport	641	245	Saturate	25	42
BusSelector	63	28	Integrator	0	5	Scope	4	17
Clock	2	1	Logic	0	2	Selector	5	0
Constant	375	155	Lookup_n-D	0	25	Signum	0	1
DataTypeConversion	0	85	Math	90	3	Sqrt	0	2
DataTypeDuplicate	0	4	Memory	0	2	SubSystem	220	153
Delay	14	0	MinMax	1	5	Sum	328	132
Demux	8	7	MultiPortSwitch	18	0	Switch	13	29
DiscreteIntegrator	44	0	Mux	11	2	Terminator	12	7
DiscretePulseGenerator	0	1	Outport	335	161	TransferFcn	0	4
DiscreteTransferFcn	1	0	Product	228	45	Trigonometry	0	4
EnablePort	5	0	Quantizer	6	0	UnitDelay	8	0
Fcn	0	2	RateTransition	26	0	VariableTransportDelay	19	0
From	38	4	RelationalOperator	8	14	ZeroOrderHold	6	0

TABLE I
SYSTEM CONTENTS

The Kalman filter is used to estimate the internal model states and the disturbances. For this purpose, the nonlinear model is linearized with respect to each operation point for the prediction step of the filter.

Target of the static analysis is the whole validation model, which contains the filter and the nonlinear system model. Further components regard the data import from workspace, handling of the test signals and evaluation. This allows comparison of the Kalman filter with values measured directly or generated by simulation of the nonlinear system model.

Significant characteristics of the model are a high number of math functions. Most of the blocks are used for the calculation of physiological values with exponential or logarithmic functions. Saturation blocks and limitations for integrators are used to prevent overflow. A separate clock and rate transition blocks are used at the inputs and outputs of the filter, to allow execution in an interval different from the surrounding test system.

The desired filter behavior for tests was defined by upper and lower bounds regarding the quality measures of a step answer. Therefore the rate of change is an important outcome of the static analysis to check the implementation for the violation of timing constraints.

b) Integration on Hardware Platform: The Simulink model is used on a real-time capable hardware platform as central control device in the network. The platform in use is a MicroAutoBox II (dSpace GmbH, Paderborn, Germany) with vendor tool chain. Its widely used for development and research in automotive and automation.

The tool chain provides a vendor extension of the Simulink code generator, which generates the platform specific binary file. Furthermore it provides an environment to create graphical user interfaces. These allow supervision of the hardware platform by a connected computer. A crucial aspect here is, that otherwise static model elements like Constant blocks can be changed by the user during execution. Obviously, this has to be taken into account for a reliable static analysis of such a model.

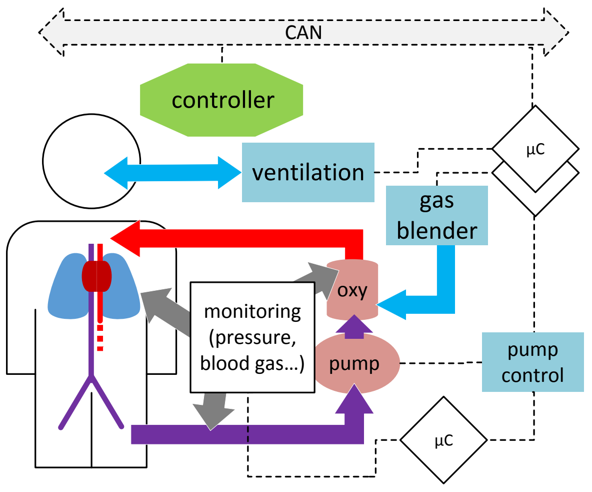

Fig. 1. Automated extracorporeal lung assist in intensive care

B. UAV

In the recent years, the development of *unmanned air vehicles* (UAVs) has rapidly increased and nowadays, small UAVs can be bought off the shelf for end users. Due to their limited size, the resource constraints are stronger, making energy efficient concepts more interesting. The second presented application is a one meter wingspan vertical take-off and landing plane, as shown in Figure 2 allowing to tilt the wing [29], [30], to transit between hover and forward fixed wing flight. Tilt-wings are applied as light weight drones for diverse use cases.

a) System Setup: In the last years, several instances of the UAV have been constructed. Thrust is generated by a DC engine at each wing, while an additional rotor at the tail increases stability during hovering. With an almost constant turn rate, the blade pitch of the tail rotor can be manipulated to control lift. An aileron at each wing allows to change yaw

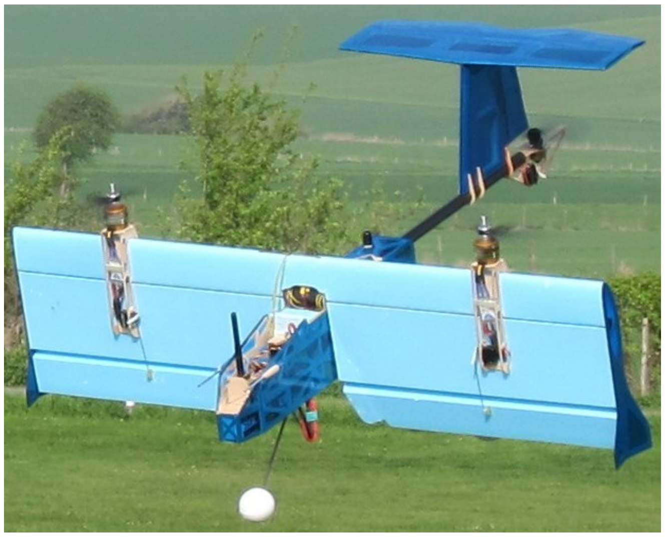
Fig. 2. Tilt-wing UAV after take-off in vertical flight. Two main engines at the wing yield thrust for take-off, while the tail rotor stabilizes pitch

during hovering and the roll angle in fixed wing flight. Differential thrust at the main engines controls yaw during fixed wing flight and roll in hovering mode. This coupling effect has been investigated in wing tunnel experiments, yielding 25 large lookup tables for the transition, which is indicated in Table I.

Since autonomous features require information about the environment, several sensors are attached to the air frame. An *inertial measurement unit* (IMU) with a GPS receiver compute a precise attitude and position, while a pressure sensor with an connected pitot tube estimates the wind speed. Finally, an ultra sonic sound sensor and barometer improve the altitude calculation. In case the autopilot fails, a pilot can take over with an remote control. Still, due to induced complexity from roll yaw coupling, only roll, pitch and yaw commands are transmitted. These commands are translated to actuator settings by a PID controller combined with lookup tables. Eventually, a mission controller manages flight tasks automatically, such as take-off, cruise flight between waypoints and landing.

b) Model Description: For the software development process, model based design methods were used by modeling the flight automation in Simulink with Stateflow. On the top level of the model, inputs from sensors are scaled and grouped into buses, which themselves are merged into a bus. Inputs include data from remote control, IMU, GPS, pressure, battery status and environment data. The data is feed into a Stateflow chart, in which different flight stages are modeled. Results from the chart include reference position, which is feed to an altitude, longitude and latitude controller. Afterwards, an attitude controller calculated output for the control surfaces. Due to the strong non-linearity of the model, corresponding lookup tables map the reference attitude combined with wind data to the control surface output. Finally, the outputs are scaled again to the corresponding value ranges of the actuators.

preciser	ECLA		UAV	
ranges	SLDV	artshop	SLDV	artshop
with p.i.	60	282	29	26
without p.i.	595	9	324	4

TABLE II
THE NUMBER OF PORT RANGES CALCULATED MORE PRECISELY COMPARED TO THE OTHER TOOL, INCLUDING THE INFLUENCE OF ARTSHOP PARAMETER INTERPRETATION (P.I.)

c) Integration on Hardware Platform: While the engineer designs a specific Simulink controller model, a second block diagram for code generation is constructed automatically. Although the inputs, i.e. sensors, and outputs are fixed during the development process and may not be changed, custom blocks allow the engineer to specify parameters and scopes for signals with additional external inputs. This is due to the fact, that during flight testing additional parameters may be changed and further states have to be watched. Hence, from the custom blocks, additional global inputs and outputs are included, yielding a model from which C code can be generated. In a final step, the C code is automatically integrated into the existing framework.

C. Evaluation

Both case studies were tested with the abstract interpretaion implemented as part of the tool artshop. It was run on a desktop computer with an Intel i5 2.67 GHz CPU, eight GB memory and a 64 bit Microsoft Windows 7 operating system.

To our knowledge, there is currently no comparable solution to compute rates of signals, however, we have used the *Simulink Design Verifier* (SLDV) R2015b by Mathworks, to provide an estimate on how our algorithm for value range analysis performs compared to a commercial tool.

In contrast to our abstract interpretation, the SLDV in the given version does not support continuous models as it is the case for the UAV model. One workaround to test a continuous model with the tool is to test the discretized version, as it was used here.

Another notable difference between the tools is, that block parameters are usually left uninterpreted by the SLDV. This results in a high number of overapproximated ranges. For instance, a constant block is assumed to allow any possible output value. However, it is not a disadvantage in general, as proven by the case study models. In both applications platform dependent code generation allows variables representing constant blocks to be overwritten by the target platform during execution. Since this is commonly used for user interface purposes and testing, it has to be taken into account for our analysis. In doing so, we added an option, which allows the user to decide whether block parameters shall be interpreted or not.

a) Precision of the value range analysis: Table II shows that, neglecting parameter interpretation, SLDV calculates for several ports more precise results than artshop. If parameters are fix and cannot be changed externally, our solution improves the outcome significantly. Examples, in which the value ranges are tighter include constant, PID controller and lookup table

		SLDV	artshop	
			with p.i.	without p.i.
ECLA	time [s]	1840	181	63
	warnings	227	28	180
UAV	time [s]	>7200	13	7
	warnings	71	35	73

TABLE III
THE COMPUTATION TIME OF THE ANALYSIS AND THE NUMBER OF ISSUED WARNINGS OF EACH TOOL INCLUDING THE INFLUENCE OF ARTSHOP PARAMETER INTERPRETATION (P.I.)

		SLDV	artshop	
			p.i.	no p.i.
ECLA	Division by zero	187	23	175
	Overflow	39	10	10
	Array Bounds	1	0	0
	Total	227	33	185
UAV	Division by zero	36	3	13
	Overflow	35	48	95
	Array Bounds	0	0	0
	Total	71	51	108

TABLE IV
ISSUED WARNING TYPES FOR BOTH MODELS WITH ALL EVALUATED TECHNIQUES.

blocks, where initial values or internal limitations can be set. Especially in case of the ECLA model, artshop produces tighter intervals, due to existing abstractions for several blocks, e.g. lookup tables, within the ECLA model, which are not supported by the SLDV and therefore overapproximated. Leaving the feature of parameter interpretation aside, the our approach is less sophisticated than the SLDV, which uses a logic solver to calculate tighter bounds for many cases. By integrating an logic solver, the computational overhead increases, which can be seen in the computation time. Table III lists the number of ports, for which each tool produced tighter value ranges. The abstract interpretation with the value range analysis in artshop took between several seconds and a few minutes. SLDV used more computation time due to the fact that a logic solver is applied to improve the outcome. A time limit of two hours was applied for the SLDV analysis, which was only reached in case of the UAV model. Smaller time limits might reduce the precision, but were not considered here.

b) Warnings: Table IV lists in detail, which warning types are issued by each technique for both models. In total, the SLDV supplies three warning categories for our models, division by zero, overflow and invalid array bound access. Additional warnings given by artshop, but not mentioned in the table, include warnings on unsupported blocks, possible underflows, division by infinite values, implicit rate transitions and constant resets. Based on the increased precision by interpretation of block parameters, the number of warnings in Table III is also significantly decreased in case of the proposed algorithm. We focus in the following on our technique without interpretation of parameters, since this is the relevant approach for both use cases. Considering divisions by zero and invalid array access, our technique yields less warnings regarding the ECLA model. The first is mostly influenced by the abstraction of artshop which allows a more precise handling of some block

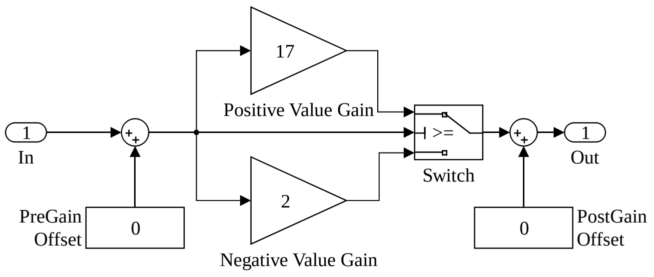

Fig. 3. UAV model parts where SLDV performs better due to constraint propagation

types. An example are lookup table blocks, where the SLDV cannot handle the interpolation in certain cases which results in a division by zero. Similar problems occur for the saturation blocks.

The invalid array access warning arises from the filter block implemented in Matlab code in the ECLA model. Here, a filter input signal is assigned in this manner: `internalvar = initial([1:12])`. However, `initial` is a signal provided by a constant block containing sufficient elements, which seems to result in a false positive given by the SLDV.

Nonetheless, on overflow warnings the SLDV performs better in the UAV model, which is due to constraint propagation. Figure 3 shows an example, in which cases artshop issues a false positive. The positive part is multiplied with a different gain, compared to the negative part. Since our algorithm does not use constraint propagation from the switch blocks, the gain of 17 is applied also to the negative part, while 2 is also multiplied with positive input.

Further noticeable are the warnings, which can only be given by artshop with parameter interpretation, which shows the relevant cases for these additional information about the block behavior. For example, it allows to return warnings for constant reset conditions, which are part of the block parameters. This might be a false positive, if the constant reset is overwritten externally during testing. In such cases, an automatic detection of external variation points can help the analysis to understand, which signals are manipulated externally.

Finally, performance of artshop is comparable to the SLDV regarding warnings and false positives. Both provide different features, which help to reduce the number of false positives.

Compared to the necessary test case development and the guaranteed safety of testing by simulation of the model the verification can be less time consuming. In contrast to testing, where we have to provide a sufficient range and combination of input signals, the verification can also be done for all possible inputs to test safety specifications like output limitations.

c) Rate of Change: For both models, rates of change have been analyzed as well and provide, in case these have not to be overapproximated by the value range analysis, additional information about the system properties. Since both case study models were thoroughly tested to be successfully applied on the target platforms, the rate of change analysis did not reveal an overall significant unexpected behavior. One positive outcome regarding the first case study revealed an error which was also found by testing with the appropriate test case.

We verified for the ECLA case study, that the rate of change of the relevant outputs behaves as specified, despite one output describing the artificial gas exchange realized via the extracorporeal circuit. Its calculated rate of change is unexpected high which corresponds to an undesired dynamic compared to the expected range. This causes an unnecessary sensitivity to noise. By adjusting the filter parameters, the noise sensitivity can be reduced.

As a result of the UAV model verification, a high rate of change of the servo speed control signal before scaling to pulse width modulation could be shown. This yields to dangerously high changes of the pulse width per time step in the final output for each actuator, which might lead to broken servos. Further testing is necessary to guarantee an acceptable rate for all actuator outputs.

In both models, which were already tested in simulations and as prototypes, the rate of change highlighted possible flaws in corner cases. With the rates of changes, potential models errors can be identified. Furthermore specifications usually include timing constraints which can be very important, especially in safety critical applications and automatically proven with this technique.

d) Drawbacks: After having highlighted the benefits of our proposal, we discuss drawbacks of our technique. Abstract interpretation requires, due to its static nature, necessary overapproximations for states, yielding potential false positives. Considering rates of change, we have shown that this method was helpful to find design flaws of the case studies. Nevertheless, in general this might not be the case, since the maximal rate of change might often be approximated with the maximum difference allowed by the value range. In this case saturation blocks might be helpful to narrow the rates afterwards. Thereby, limiting the signal between system parts does not only directly facilitate a safer model design, but also increases the efficiency of verification for parts which are not supported. This is especially true, if the model contains blocks, for which no abstractions have been defined.

Another aspect is the usefulness of data about rate of change. While we believe value ranges are an important aspect, since potential arithmetical errors can be identified, there is no such benefit without further requirements. Thus, results from the rate of change computations might often only be helpful to experts, who have a deep knowledge about the system behavior. Though bounds on the rates of change might be provided, no further time frame is specified, in which this bound holds. In particular for cases, in which the maximum rate of change can only occur in the very first time steps, the bounds might prove to be unhelpful.

Considering our current implementation, only value and rate ranges with warnings are yielded. Although with model slicing techniques, the potential sources can be identified, no concrete counter example is given. Furthermore, without relational domains or symbolic support, interval sets prove to be very overapproximative.

V. Conclusion

In this paper we have shown that abstract interpretation for block diagrams is feasible. Therefore, the proposed approach was successfully applied to two models already implemented and tested in research prototypes in the domain of intensive care and unmanned air vehicles. Furthermore we presented how dynamics properties, such as rate of change can be included in a static analysis. It has to be tested to what extend this might improve the precision of a value range analysis. In conclusion, our evaluation highlights challenges of real world examples, as well as the advantages and drawbacks of our method, which trades precision for speed.

VI. Acknowledgments

The authors gratefully acknowledge help and support from the Institute for Flight System Dynamics, headed by Professor Moormann and the Philips Chair for Medical Information Technology headed by Professor Leonhardt, both at RWTH Aachen University, who provided the case study models.

Furthermore the authors gratefully acknowledge the contribution of the German Research Foundation DFG (Grant PAK 138/2; LE 817/15-1; KO 1430/14-1; RO 2000/18-1).

References

[1] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, and X. Rival, "Why does ASTRÉE scale up?" *Formal Methods in System Design*, vol. 35, no. 3, pp. 229–264, 2009.

[2] S. Stattelmann, S. Biallas, B. Schlich, and S. Kowalewski, " Applying Static Code Analysis on Industrial Controller Code ," in *19th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA).* IEEE, 2014, work in Progress Best Paper Award.

[3] C. Dernehl, N. Hansen, T. Gerlitz, and S. Kowalewski, "Static value range analysis for matlab/simulink-models," in *INFORMATIK 2015*. Douglas W. Cunningham, Petra Hofstedt, Klaus Meer, Ingo Schmitt, 2015, pp. 1649–1660.

[4] T. Gerlitz, N. Hansen, C. Dernehl, and S. Kowalewski, " artshop: A Continuous Integration and Quality Assessment Framework for Model-Based Software Artifacts ," in *12. Dagstuhl-Workshop Modelbasierte Entwicklung eingebetteter Systeme (MBEES).* fortiss Technischer Bericht, 2016, pp. 13–22.

[5] P. Cousot and R. Cousot, "Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints," in *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, ser. POPL '77. New York, NY, USA: ACM, 1977, pp. 238–252. [Online]. Available: http://doi.acm.org/10.1145/512950.512973

[6] A. Miné, "Relational abstract domains for the detection of floating-point run-time errors," in *Programming Languages and Systems.* Springer, 2004, pp. 3–17.

[7] P. Granger, "Static analysis of arithmetical congruences," *International Journal of Computer Mathematics*, vol. 30, no. 3-4, pp. 165–190, 1989.

[8] J. Feret, "Numerical abstract domains for digital filters," in *International workshop on Numerical and Symbolic Abstract Domains (NSAD)*, 2005.

[9] M. Broy, S. Kirstan, H. Krcmar, B. Schätz, and J. Zimmermann, "What is the benefit of a model-based design of embedded software systems in the car industry?" *Software Design and Development: Concepts, Methodologies, Tools, and Applications*, p. 310, 2013.

[10] E. M. Clarke, O. Grumberg, and D. E. Long, "Model checking and abstraction," *ACM transactions on Programming Languages and Systems (TOPLAS)*, vol. 16, no. 5, pp. 1512–1542, 1994.

[11] G. J. Holzmann, "The model checker SPIN," *IEEE Transactions on software engineering*, vol. 23, no. 5, p. 279, 1997.

[12] K. G. Larsen, P. Pettersson, and W. Yi, "UPPAAL in a nutshell," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 1, no. 1, pp. 134–152, 1997.

[13] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, "NuSMV: A new symbolic model verifier," in *Computer Aided Verification*. Springer, 1999, pp. 495–499.

[14] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "Nusmv 2: An opensource tool for symbolic model checking," in *Computer Aided Verification*. Springer, 2002, pp. 359–364.

[15] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta, "The nuXmv symbolic model checker," in *Computer Aided Verification*. Springer, 2014, pp. 334–342.

[16] A. Lomuscio, C. Pecheur, F. Raimondi *et al.*, "Automatic Verification of Knowledge and Time with NuSMV." in *IJCAI*, 2007, pp. 1384–1389.

[17] M. Panti, L. Spalazzi, and S. Tacconi, "Using the NuSMV model checker to verify the kerberos protocol," 2002.

[18] L. De Moura and N. Bjørner, "Z3: An Efficient SMT Solver," in *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, ser. TACAS'08/ETAPS'08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 337–340. [Online]. Available: http://dl.acm.org/citation.cfm?id=1792734.1792766

[19] S. Gao, S. Kong, and E. M. Clarke, "dReal: An SMT solver for nonlinear theories over the reals," in *Automated Deduction–CADE-24*. Springer, 2013, pp. 208–214.

[20] B. Dutertre and L. De Moura, "The yices smt solver," *Tool paper at http://yices. csl. sri. com/tool-paper. pdf*, vol. 2, no. 2, 2006.

[21] S. Tripakis, C. Sofronis, P. Caspi, and A. Curic, "Translating discrete-time simulink to lustre," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 4, no. 4, pp. 779–818, 2005.

[22] A. Agrawal, G. Simon, and G. Karsai, "Semantic translation of Simulink/Stateflow models to hybrid automata using graph transformations," *Electronic Notes in Theoretical Computer Science*, vol. 109, pp. 43–56, 2004.

[23] O. Bouissou and A. Chapoutot, "An operational semantics for simulink's simulation engine," *ACM SIGPLAN Notices*, vol. 47, no. 5, pp. 129–138, 2012.

[24] A. Chapoutot and M. Martel, "Abstract Simulation: A Static Analysis of Simulink Models," in *Embedded Software and Systems, 2009. ICESS '09. International Conference on*, May 2009, pp. 83–92.

[25] ——, "Static analysis of simulink programs," *Model-driven High-level Programming of Embedded Systems (SLA++ P08), ENTCS*, 2008.

[26] R. Reicherdt and S. Glesner, "Formal Verification of Discrete-Time MATLAB/Simulink Models Using Boogie," *Software Engineering and Formal Methods*, pp. 190–204, 2014.

[27] T. Hickey, Q. Ju, and M. H. Van Emden, "Interval arithmetic: From principles to implementation," *Journal of the ACM (JACM)*, vol. 48, no. 5, pp. 1038–1068, 2001.

[28] C. Brendle, K.-F. Hackmack, J. Kühn, M. N. Wardeh, R. Kopp, R. Rossaint, A. Stollenwerk, S. Kowalewski, B. Misgeld, S. Leonhardt, and M. Walter, "In silico evaluation of gas transfer estimation during extracorporeal membrane oxygenation," in *9th IFAC Symposium on Biological and Medical Systems*, 2015, to appear.

[29] M. Schütt, P. Hartmann, and D. Moormann, "Fullscale windtunnel investigation of actuator effectiveness during stationary flight within the entire flight envelope of a tiltwing MAV," in *IMAV 2014: International Micro Air Vehicle Conference and Competition 2014, Delft, The Netherlands, August 12-15, 2014*. Delft University of Technology, 2014.

[30] P. Hartmann, C. Meyer, and D. Moormann, "Unified approach for velocity control and flight state transition of unmanned tiltwing aircraft," in *AIAA Guidance, Navigation, and Control Conference*, 2016, p. 2101.