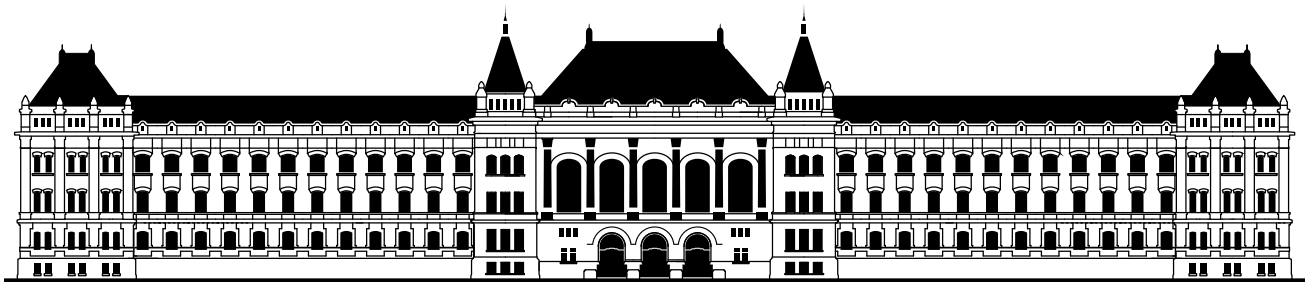
# PROCEEDINGS OF THE 26TH MINISYMPOSIUM

OF THE

## DEPARTMENT OF MEASUREMENT AND INFORMATION SYSTEMS
## BUDAPEST UNIVERSITY OF TECHNOLOGY AND ECONOMICS

## (MINISY@DMIS 2019)

**JANUARY 24, 2019**
**BUDAPEST UNIVERSITY OF TECHNOLOGY AND ECONOMICS**
**BUILDING I**



BUDAPEST UNIVERSITY OF TECHNOLOGY AND ECONOMICS
DEPARTMENT OF MEASUREMENT AND INFORMATION SYSTEMS

Head of the Department: Tamás Dabóczi

Conference chairman:
Béla Pataki

Organizers:
Rebeka Farkas
Ákos Hajdu
Attila Klenik

Homepage of the Conference:
`http://minisy.mit.bme.hu/`

Sponsored by:

Schnell László Foundation

# FOREWORD

This proceedings is a collection of the papers of the 26th Minisymposium taking place at the Department of Measurement and Information Systems of the Budapest University of Technology and Economics. At the beginning the main purpose of these symposia was to give an opportunity to the PhD students of our department to present a summary of their research results in the preceding year. As an interesting additional benefit, the students also got experience in organizing scientific events. Beyond the original goal, it turned out that the proceedings of our symposia give an interesting overview of the research in our department. A few years ago the scope of the Minisymposium had been widened; some of the best BSc/MSc students involved in research also participate.

The lectures reflect an important part of the scientific fields and work of our department. Traditionally the symposium was focused on measurement and instrumentation. The area has slowly changed and widened during the last few years. New areas mainly connected to embedded information systems, dependability and security, artificial intelligence, and cyber-physical systems are now in our scope of interest as well. The papers cover both theoretical and practical aspects..

The proceedings will not be published in printed form, it has turned out that nowadays the web publication of symposium lectures is enough. This new form has some advantages, but it has some challenges as well. We hope that the advantages will dominate.

During this twenty-six-year period there have been shorter or longer cooperation projects, research visits and internships between our department and some universities, research institutes, organizations and firms. Some of our research benefited substantially from these connections/collaborations. In the last year the cooperation was especially fruitful with the Software Competence Center Hagenberg (SCCH), Austria; University of Zagreb, Croatia; SRI International, New York, USA.

We hope that similarly to the previous years, this Minisymposium will also be useful for the lecturers, for the audience and for all who read the proceedings.

Budapest, January, 2019

Béla Pataki
Chairman of the
Minisymposium

## PAPERS OF THE MINISYMPOSIUM



The research reports indicated by * are not published in this volume. To gain access, please contact the organizers or the authors.

# Software Model Checking with a Combination of Explicit Values and Predicates

Viktória Dorina Bajkai[1], Ákos Hajdu[1,2]
[1]Budapest University of Technology and Economics, Department of Measurement and Information Systems
[2]MTA-BME Lendület Cyber-Physical Systems Research Group
Email: hajdua@mit.bme.hu

***Abstract*—Formal verification techniques can both reveal bugs or prove their absence in programs with a sound mathematical basis. However, their high computational complexity often prevents their application on real-world software. Counterexample-guided abstraction refinement (CEGAR) aims to improve efficiency by automatically constructing and refining abstractions for the program. There are several existing abstract domains, such as explicit-values and predicates, but different abstract domains are suitable for different kinds of software. Therefore, product domains have also emerged, which combine different kinds of abstractions in a single algorithm. In this paper, we present a new variant of the CEGAR algorithm, which is a combination of explicit-value analysis and predicate abstraction. We perform an experiment with a wide range of software systems and we compare the results to the existing methods. Measurements show that our new algorithm can efficiently combine the advantages of the different domains.**

## I. Introduction

Formal verification techniques (e.g. model checking) provide a sound mathematical basis to prove the correct operation of a program by exhaustively analyzing all possible states and transitions. This is also the downside of such methods, making them too expensive computationally. Counterexample-guided abstraction refinement (CEGAR) [1] is a widely used approach to overcome this limitation. CEGAR applies abstraction to hide certain details and to over-approximate the set of possible behaviors of the program. However, this does not only yield a smaller state space, but can also lead to spurious counterexamples. In such cases, the algorithm automatically refines the abstraction and repeats this process until a sufficient precision is found. CEGAR can work with different abstract domains, such as explicit values and predicates. The former works by tracking only a subset of the program variables, while the latter stores certain relationships and facts about them.

Product abstractions [3], [4], which combine multiple abstract domains also emerged, since different abstract domains turned out to be suitable for different kinds of software. In this paper we present a product abstraction algorithm, which combines explicit values [8] and predicate abstraction [7], exploiting the advantages of each domain. The key idea of our approach is that we always start with explicit values to avoid handling formulas, but switch to predicates if there are too many values for a variable.

Partially supported by Nemzeti Tehetség Program, Nemzet Fiatal Tehetségeiért Ösztöndíj 2018 (NTP-NFTÖ-18).

We implemented this algorithm in Theta [2], an open source verification framework, which already includes a generic CEGAR loop and some basic abstract domains. We evaluate our new approach and the existing methods on several programs from different problem domains, including PLC models form CERN [11] and C programs from the Competition on Software Verification (SV-Comp) [12]. Our results show that the new product abstraction algorithm successfully combines the advantages of two existing domains.

*Related work:* The dynamic precision adjustment approach [3] combines predicates and explicit values. The main difference is that it switches to predicates based on the whole state space, whereas we only consider the successors of a single state. Moreover, we allow successors to be enumerated if an expression cannot be evaluated, as opposed to the unknown values in dynamic precision adjustment.

Refinement selection [4] chooses between the explicit and predicate domains based on various metrics for the refinement quality. Our approach always tries the explicit domain first, but switches to predicates if there are too many different values.

## II. Background

### A. Control Flow Automata

In our work, we use *control flow automata* (CFA) [5] to model programs. A CFA is a tuple $(V, L, l_0, E)$ where

- $V = \{v_1, v_2, \ldots, v_n\}$ is a set of program *variables* with domains $D_1, D_2, \ldots, D_n$,
- $L = \{l_1, l_2, \ldots, l_k\}$ is a set of program *locations* representing the program counter,
- $l_0 \in L$ is the *initial location*, i.e., the entry point of the program,
- $E \subseteq L \times Ops \times L$ is a set of *directed edges* between locations, representing the operations that get executed when going from the source location to the target.

A *concrete state* $c = (l, d_1, \ldots, d_n)$ of the CFA consists of a location $l \in L$ and a value $d_i \in D_i$ for each variable $v_i$ from its domain. A *transition* $c \xrightarrow{op} c'$ exists between two states, if there is an edge $(l, op, l') \in E$ between their locations and the semantics of $op$ matches the variables. Operations $op \in Ops$ can be *assumptions*, *assignments* or *havocs*. Assumptions are first order logic (FOL) [6] predicates denoted by $[\varphi]$, which must hold at the source state. Assignments are in the form of $v_i := \psi$, where $\psi$ is a FOL expression with domain $D_i$ that

updates the variable $v_i$ in the target state. Havocs have the form *havoc* $v_i$, where $v_i$ is assigned a non-deterministic value in the target state. A *concrete path* $c_1 \xrightarrow{op_1} c_2 \xrightarrow{op_2} ... \xrightarrow{op_{n-1}} c_n$ is a sequence of concrete states and operations, where $c_1$ has the initial location $l_0$.

A *verification task* consists of a *CFA* and a dedicated *error location* $l_e \in L$. A CFA is *safe* if no concrete path exists to a state which contains the error location $l_e$.

As an example, consider the CFA in Figure 1a. It represents a program, which first examines whether its single variable $x$ is not 1 and then if it is 1, leading to the error location $l_e$. Otherwise, the program ends in the final location $l_f$. It is clear that no concrete path can reach $l_e$, thus the CFA is safe.

### B. Counterexample-Guided Abstraction Refinement

In this section, we define the abstract CEGAR framework, which will be instantiated in the next paragraphs. CEGAR can work with different abstract *domains* [5]. An abstract domain is a structure $(S, \sqsubseteq, \Pi, T)$ where

- $S = \{s_1, s_2, \ldots, s_n\}$ is a set of *abstract states*,
- $s \sqsubseteq s'$ is a *coverage relation*, which holds for two abstract states, if $s'$ represents all the states that $s$ does,
- $\Pi$ is a set of *precisions*, which controls granularity of the abstraction,
- $T$ is the *transfer function*, defining the successor relation between abstract states.

In this work we combine two different domains: explicit-value analysis [8] and predicate abstraction [7].

The first step of CEGAR is to build an *abstract reachability graph* (ARG) from the original model, with an initial, usually coarse *precision* $\pi \in \Pi$. ARG generation maintains a queue $Q$ for the unprocessed states, starting with the initial abstract state $s_0 \in S$, corresponding to $l_0$. As long as the queue is not empty, it picks a state $s \in Q$ and checks if it can be covered with some already explored state $s'$, i.e., $s \sqsubseteq s'$. If not, the successors of $s$ are added to the queue for each operation $op$ on the outgoing edges using the transfer function $T(s, op, \pi)$. Abstraction stops if the queue $Q$ is empty or a state with the error location $l_e$ is reached. Since the abstraction we use over-approximates the program, if we cannot reach $l_e$, the original program is also safe. Otherwise, an abstract counterexample exists, which is a path in the ARG leading to a state with $l_e$.

The second step is to examine the abstract counterexample. This is done by the *refiner* $R$, which checks whether the counterexample is feasible in the original program. If it is, the original program is unsafe. Otherwise, the counterexample is spurious and the abstraction is refined. The refiner $R$ typically returns a new precision $\pi'$ that should be joined to the previous ($\pi \cup \pi'$), so that the spurious counterexample is eliminated from the next iteration. The process repeats until there are no counterexamples or a feasible one is found.

### C. Explicit-Value Analysis

Explicit-value analysis [8] works by tracking only a subset of the program variables. A precision $\pi_e \in \Pi_e$ defines the subset of the variables $\pi_e \subseteq V$, which are currently tracked. If a variable is not tracked, its value is represented by a special *top element* $\top$, which means that it can take any value from its domain. Abstract states $s_e = (l, d_1, \ldots, d_n)$ in explicit-value analysis therefore, consist of a location $l$ and values $d_i \in D_i \cup \{\top\}$ for each variable $v_i$. The coverage relation $s \sqsubseteq s'$ holds between two states, if their locations are equal and each value in $s'$ is either $\top$ or the same as in $s$. The transfer function $T_e$ works in the following way for a given state $s_e$, operation $op$ and precision $\pi_e$. If $op$ is an assumption and evaluates to true or cannot be evaluated (due to $\top$ values), a successor state is created where the value of the tracked variables will not change. If $op$ is an assignment, the value of the assigned variable in the successor will be the result of evaluating the expression, or $\top$ if it cannot be evaluated or the assigned variable is not tracked. If $op$ is a havoc, the value of the havocked variable becomes $\top$. Abstraction usually starts with an empty precision $\pi_e = \emptyset$ and refinement $R_e$ is performed by iteratively extending the precision $\pi_e$ with additional variables. The new variables to be tracked can be inferred by different *interpolation* techniques [8], [9].

As an example, consider the ARG in Figure 1b (for the CFA in Figure 1a) created with explicit-value analysis. The value of $x$ is not known initially and also remains $\top$ after $[x \neq 1]$. However, after $[x = 1]$ we know that $x$ is 1.

### D. Predicate Abstraction

In predicate abstraction [7], the concrete values of variables are not tracked explicitly. Instead, certain facts and relationships are tracked through a set of FOL formulas over $V$, called the *predicates*. The precision $\pi_p \in \Pi_p$ is therefore, a set of predicates. Abstract states $s_p = (l, p_1, \ldots, p_k)$ contain the location and the ponated or negated version of the predicates $p_i$ in $\pi_p$. It is also possible that a predicate does not occur in an abstract state, if it can both hold or not. The coverage relation $s \sqsubseteq s'$ holds between two states if their locations are equal and the predicates of $s$ imply the predicates of $s'$. The transfer function $T_p$ works in the following way for a given state $s_p$, operation $op$ and precision $\pi_p$. If $op$ is an assumption, we check whether the conjunction of the predicates of the source state and the assumption is satisfiable. If yes, a successor state is created, including the predicates from $\pi_p$ (or their negated form) that are implied by the source state and the assumption. If $op$ is an assignment, we create a successor that includes the predicates or their negated form that are implied by the source state and the assignment. If the operation is a havoc, a successor state is created, where predicates including the havocked variable are excluded. Abstraction usually starts with an empty precision $\pi_p = \emptyset$ and refinement $R_p$ is performed by iteratively extending the precision $\pi_p$ with additional predicates. Similarly to explicit values, the new predicates can be inferred by interpolation [9].

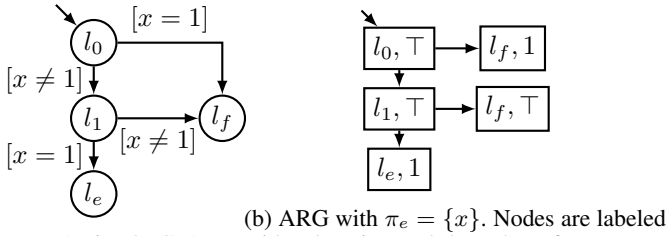
As an example, consider the ARG in Figure 2b (for the CFA in Figure 1a) with tracking the predicate $x = 1$. The assumption $[x \neq 1]$ ensures that $l_1$ is labeled with the negation of the predicate, and thus the error location will not be reachable in the ARG.

### E. Product Abstraction

Product abstractions [3], [4] combine different abstract domains. In our work we combine explicit-value analysis and predicate abstraction by tracking both explicit values and predicates simultaneously. Therefore the precision is $\Pi = \Pi_e \times \Pi_p$ and an abstract state $s = (l, d_1, \ldots, d_n, p_1, \ldots, p_k)$ consists of a location, values and predicates. A state $s$ is covered by another state $s'$ if their locations are equal and both components are covered. The transfer function $T$ gets a product state $s = (l, s_e, s_s)$, an operation $op$, a precision $\pi = (\pi_e, \pi_p)$ and calculates $T_e((l, s_e), op, \pi_e) \times T_p((l, s_p), op, \pi_p)$, that is the Cartesian product of the successor explicit and predicate states. Abstraction usually starts with an empty precision $\pi = (\emptyset, \emptyset)$ for both components. In product abstraction, the main decision regarding the new precision should be made during refinement $R$. The component refiners $R_e$ and $R_p$ will return new variables $\pi'_e$ and predicates $\pi'_p$ to be tracked and the algorithm has to decide which of them to use. In the next section we propose a new strategy to choose between explicit values and predicates.

## III. Product Strategy with Limited Enumeration

The key idea of our approach is that we always extend the set of explicitly tracked values ($\pi_e$) first, since handling predicate formulas is more expensive computationally (e.g., checking implications). However, a downside of explicit-value analysis is that some problems are not decidable due to expressions with unknown values ($\top$) that cannot be evaluated.




(a) A simple CFA.

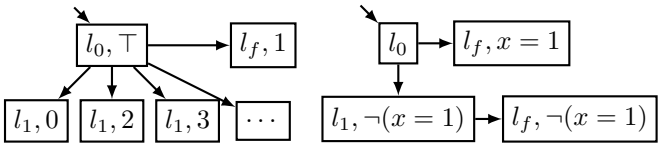
(b) ARG with $\pi_e = \{x\}$. Nodes are labeled with a location and the value of $x$.

Fig. 1: Example CFA and ARG with explicit-value analysis.

Recall the example CFA in Figure 1a, which first checks if $x \neq 1$ and then if $x = 1$ (which obviously cannot be possible). If no variables are tracked initially, the error location is trivially reachable and the refiner extends the precision with the only variable $x$. Figure 1b shows the corresponding ARG with $\pi_e = \{x\}$. Since $x$ is not initialized, the initial state is $(l_0, \top)$. Then we check the condition $x \neq 1$. Since $x$ is unknown, the condition can both hold and not. If it does not hold, the program terminates in the final location $(l_f, 1)$. However, if it holds we proceed to $l_1$, where $x$ is still $\top$, since we cannot represent the fact that $x \neq 1$ in explicit value analysis. Then we check the condition $[x = 1]$, which can again hold or not, due to $x$ being unknown. This way, the program can still reach the error location $(l_e, 1)$, which is a spurious counterexample. Since there are no more variables to be tracked, the program cannot be verified (with explicit-value analysis).

To address such limitations, we propose a modified version of the explicit transfer function $T_e$, where we start to list all possible values instead of using a $\top$ value if an expression cannot be evaluated. This can already solve the problem for some cases, e.g., an assumption $[0 < x < 5]$ would yield 4 successors with 4 values for $x$ (instead of a single successor where $x = \top$). However, this can also easily lead to state space explosion. As an example, consider the CFA in Figure 1a again, where $x$ was already added to the set of explicitly tracked variables as previously ($\pi_e = \{x\}$). The corresponding ARG for this precision can be seen in Figure 2a. The program starts at state $(l_0, \top)$, from where it can go in two different directions. Taking the assumption $[x = 1]$, it arrives at state $(l_f, 1)$ since $x = 1$ is the only possible value satisfying the formula. Otherwise, the program moves to $l_1$, where it starts to list the possible values for $[x \neq 1]$, which obviously leads to a state space explosion.

To overcome this issue, we also introduce a limit $k$. During enumeration in $T_e$, we count the different values for each explicitly tracked variable (in $\pi_e$). If the number of different values of a variable $v_i$ in the successor states exceeds $k$, we remove it from the explicit precision ($\pi_e := \pi_e \setminus \{v_i\}$) and also mark it with a flag, so that the refiner will not include it again. Since the precision changed, we restart the enumeration in $T_e$, but now with a new precision $\pi_e$. We repeat this process until there are no more successor states to be enumerated and no variable was excluded.




(a) ARG created with $\pi_e = \{x\}$ and enumeration.

(b) ARG created with $\pi_p = \{(x = 1)\}$.

Fig. 2: Examples for the new product abstraction algorithm.

During refinement $R$ we use both the explicit refiner $R_e$ and the predicate refiner $R_p$ to obtain new variables $\pi'_e$ and predicates $\pi'_p$ to be tracked. We loop through the new variables $v_i \in \pi'_e$ and check if they are marked with the flag. If they are, we do not include them, but rather extend the predicate precision $\pi_p$ with those predicates in $\pi'_p$ that contain $v_i$. Otherwise, we extend the explicit precision $\pi_e$ with $v_i$.

Recall the ARG in Figure 2a again. Our new transfer function $T_e$ stops enumerating values for $x$ after a finite $k$, removes $x$ from the set of explicitly tracked variables $\pi_e$ and restarts the enumeration. However, now $x$ is not tracked anymore, so we trivially reach the error location (similarly to Figure 1b but we reach $(l_e, \top)$, since $x$ is not tracked). The product refiner $R$ will not add $x$ again, since it is flagged. Instead, it uses $R_p$ to add some predicate, e.g., $x = 1$ to the precision $\pi_p$. Figure 2b shows the ARG created with the new precision. From $l_0$, the program can arrive to the final location $(l_f, x = 1)$ where the predicate is true, or move to $(l_1, \neg(x = 1))$ where the negation of the predicate holds.

At this point, the predicates keep track that $x \neq 1$, so the algorithm can only proceed to $(l_f, \neg(x = 1))$. Since there are no more states to explore and the algorithm did not reach the error location, the program is safe. For this example, product abstraction first used $x$, then discarded it and used a predicate instead. However, in general it is possible that some variables remain explicitly tracked, while others have predicates.

## IV. Evaluation

We implemented the algorithm in the open source Theta framework [2], which already includes the explicit and predicate domains, and the Z3 SMT solver [10]. We implemented a modified transfer function for the explicit domain and a refinement procedure for product abstraction. We ran measurements on 90 PLC (programmable logic controller) programs from CERN [11] and 340 C programs from the Competition on Software Verification (SV-Comp) [12], containing large event-driven systems (eca), small locking mechanisms (locks) and large server-client systems (ssh). We evaluated these programs with eight different configurations: explicit-value analysis (EXPL), predicate abstraction (PRED) and our new product strategy (PROD) with six different limits ($k = 1, 2, 4, 8, 16, 32$). We ran the measurements on a 64 bit Ubuntu 16.04 OS using the RunExec tool from the BenchExec suite [13], which ensures highly accurate results. We enforced a time limit of 180 seconds and a memory limit of 4 GB.

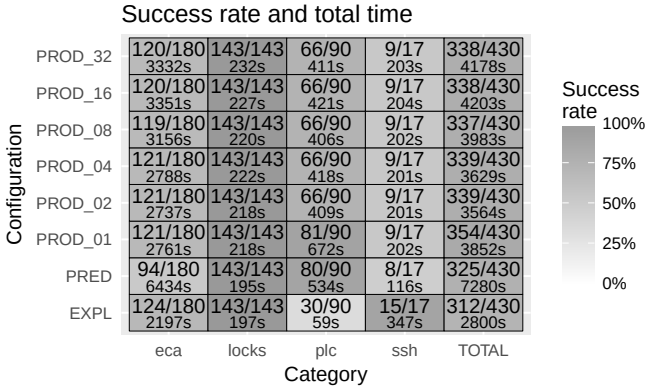



Fig. 3: Heatmap of the results for each configuration in each category. Cells include the number of verified models among the total and the required time for the successful executions.

The heatmap in Figure 3 shows the results of our evaluation. Rows correspond to the configurations, while columns represent categories. The last column is a summary of all categories. Each cell contains the number of successfully verified models and the total number in that category. We also included the execution time (in seconds) required for the successful runs. We can see, that the product abstraction strategies have better overall performance than explicit values and predicates. Furthermore, product abstraction with $k = 1$ has the best overall performance, verifying a total number of 354 models.

In category plc, PRED is successful but EXPL is not, and the eca category is the other way around. However, the PROD strategies (especially with $k = 1$) provide a good performance in both categories, combining the advantages of the two base domains. The locks category was easy for each configuration. The ssh category is interesting, because EXPL performs well, but the PROD strategies are closer to PRED with a rather poor result. This would require further investigation.

In general, the overall results confirm that our product abstraction strategy can successfully combine the strengths of explicit-value analysis and predicate abstraction.

## V. Conclusions

In our paper we investigated CEGAR-based software model checking and presented a new product abstraction strategy, which combines explicit values, enumeration and predicate abstraction. We implemented the new algorithm in the Theta verification framework, ran measurements on various input programs and compared it to existing domains. Our experiment shows that the new algorithm can successfully combine the advantages of explicit-value analysis and predicate abstraction, yielding a more efficient model checking strategy.

## References


[1] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement for symbolic model checking," *Journal of the ACM*, vol. 50, no. 5, pp. 752–794, 2003.

[2] T. Tóth, A. Hajdu, A. Vörös, Z. Micskei, and I. Majzik, "Theta: a framework for abstraction refinement-based model checking," in *Proc. 17th Conf. on Formal Methods in Computer-Aided Design*. FMCAD inc., 2017, pp. 176–179.

[3] D. Beyer, T. A. Henzinger, and G. Theoduloz, "Program analysis with dynamic precision adjustment," in *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 2008, pp. 29–38.

[4] D. Beyer, S. Löwe, and P. Wendler, "Refinement selection," in *Model Checking Software*, ser. LNCS. Springer, 2015, vol. 9232, pp. 20–38.

[5] D. Beyer, T. A. Henzinger, and G. Théoduloz, "Configurable software verification: Concretizing the convergence of model checking and program analysis," in *Computer Aided Verification*, ser. LNCS. Springer, 2007, vol. 4590, pp. 504–518.

[6] A. R. Bradley and Z. Manna, *The calculus of computation: Decision procedures with applications to verification*. Springer, 2007.

[7] S. Graf and H. Saidi, "Construction of abstract state graphs with PVS," in *Computer Aided Verification*, ser. LNCS. Springer, 1997, vol. 1254, pp. 72–83.

[8] D. Beyer and S. Löwe, "Explicit-state software model checking based on CEGAR and interpolation," in *Fundamental Approaches to Software Engineering*, ser. LNCS. Springer, 2013, vol. 7793, pp. 146–162.

[9] Á. Hajdu, T. Tóth, A. Vörös, and I. Majzik, "A configurable CEGAR framework with interpolation-based refinements," in *Formal Techniques for Distributed Objects, Components and Systems*, ser. LNCS. Springer, 2016, vol. 9688, pp. 158–174.

[10] L. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science. Springer, 2008, vol. 4963, pp. 337–340.

[11] B. Fernández Adiego, D. Darvas, E. Blanco Viñuela, J.-C. Tournier, S. Bliudze, J. O. Blech, and V. M. González Suárez, "Applying model checking to industrial-sized PLC programs," *IEEE Trans. on Industrial Informatics*, vol. 11, no. 6, pp. 1400–1410, 2015.

[12] D. Beyer, "Software verification with validation of results," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science. Springer, 2017, vol. 10206, pp. 331–349.

[13] D. Beyer, S. Löwe, and P. Wendler, "Reliable benchmarking: requirements and solutions," *International Journal on Software Tools for Technology Transfer*, 2017, online first.

# Data quality and usability requirement analysis with visualization techniques and domain-specific constraints

Dezső Burján, László Gönczy
Budapest University of Technology and Economics
Department of Measurement and Information Systems
Budapest, Hungary
Email: burjan.dezso.bme@gmail.com, gonczy@mit.bme.hu

***Abstract*—Nowadays numerous problems can be solved with data analysis which needs proper data collecting and storing mechanisms. Data has to be verified and validated according to requirements of quality and usability.**
**In our paper, we propose a method following the ISO 8000 standards and a supporting tool based on modern visualization techniques. The data quality process is greatly supported by proper visualization plots and vital questions can be answered with visualization methods and domain-specific constraints, such as the usability of the gathered data or the samples sufficiency.**

***Index Terms*—data quality, data usability, visualization, domain-specific constraints, data requirement analysis**

## I. Introduction

Data analysis is important, but to get the right results we need usable datasets which have good quality. In our study, we work on datasets which have data frame formats.
The structure of the document is as follows. Section II presents all the work which are related to our study. Section III shows the different characteristics of the requirements for data quality and usability. Section IV presents our method's meta-rules and visualization techniques, which can be used for verifying the requirements of a dataset. Section V shows an example of how to use our method on a Cloud-based distributed service. Section VI summarizes our work and also writes about the further development we plan to make.

## II. Related work

There are numerous studies related to our work; we highlight the two which we relied on in our work, both write about data quality and usability requirements. ISO 8000 standard [1], lays down the foundations. Reference [2], uses the first study and also gives a method called Data Quality Assessment Framework (DQAF) which can be used for checking that if the validity of the requirements for a dataset. Reference [8] provides and [9] defines a tool. Reference [7] gives a model-driven development based approach for resilient cyber-physical system and a method for verifying CPS components. Reference [6], shows how different visualization methods can be used to preserve the resilience of a system. There are studies [12], [13], [14], [15], [16], [17] about typical data cleaning techniques, but here the meta-rules we used in our work isn't presented.

## III. Requirement characteristics for data quality and usability

If we apply ISO 9000:2015 [3] definition of quality, data quality can be defined as the degree to which a set of characteristics of data fulfills requirements. Reference [2] descriptions of the characteristics can be found on the table below.

| Characteristics | Description and Validation |
|---|---|
| *Completeness* | Completeness implies having all the necessary or appropriate parts; being entire, finished, total.<br><br>**Validation:** Compare summarized data in amount field to summarized amount provided in a control record. |
| *Timeliness* | Timeliness is associated with data delivery, availability, and processing.<br><br>**Validation:** Compare actual time of data delivery to scheduled data delivery. |
| *Validity* | Validity is differentiated from both accuracy and correctness.<br><br>**Validation:** Compare values on incoming data to valid values in a defined domain (reference table, range or mathematical rule). |
| *Consistency* | Consistency is the degree to which data conform to an equivalent set of data, usually, a set produced under similar conditions or a set produced by the same process over time.<br><br>**Validation:** Compare record count distribution of values (column profile) to past instances of data populating the same field. |
| *Integrity* | Integrity refers to the state of being whole and undivided or the condition of being unified.<br><br>**Validation:** Confirm record level (parent/child) referential integrity between tables to identify parentless child records (i.e., "orphans records"). |

A dataset can be considered usable if it fulfills the requirements and it is accurate and reasonable.

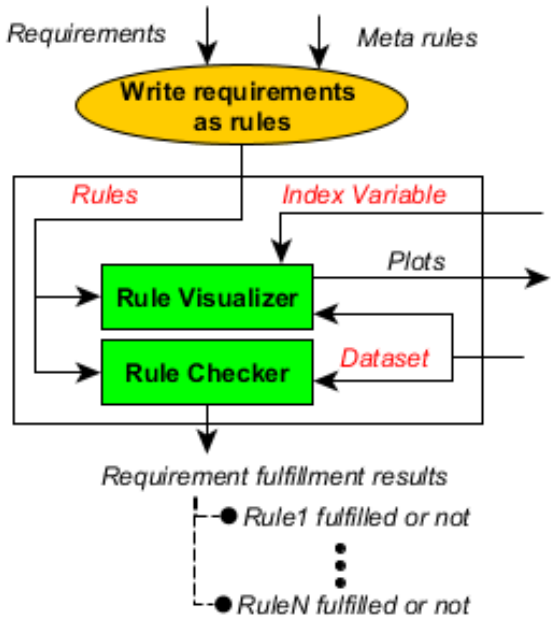



Fig. 1. The model of the requirement checking engine.

## IV. Requirement analysis

In this section, we will present our method, which can determine the validity of the requirements on a dataset.
The Requirement checker engine has three parameters:

1) Rules, which derived from the requirements using the meta-rules presented in this paper.
2) Dataset under validation.
3) Index Variable, a column from the dataset and this also the parameter for the plots.

### A. Meta-rules

Our method extends the DQAF method with meta-rules, which validity on a dataset can be checked using our *Rule checker* component. We represent the meta-rules with the grammar presented in the future. The operators in our meta-rules are the following:

- **SingleBound:** Numeric values of variable *X* must be below/above value *Y*.
- **MultipleBound:** Numeric values of variable *X* must be between/outside of values *Y* and *Z*.
- **SingleValue:** Values of variable *X* must (not) be *Y*.
- **Derivation:** Values of variable *X* can be calculated from $Y_1, ..., Y_N$.
- **Classification:** Qualitative values of variable *X* must be in the proper *Y* class.

The operands in our meta-rules are the following:

- **TypeBounding:** Type of variable *X* must be *Y*.
- **Environment:** Next or previous values can be used rather than current.
- **TimeRanges:** For datasets which contain timestamps, we can add time intervals.
- **Variables:** Instead of constants, column values can be used (e.g., values of *Y* variable).
- **Quantified:** For quantized variables, an order can be defined (e.g., GREEN $<$ YELLOW $<$ RED). Using this the category with the highest or lowest importance can be specified.
- **Aggregators:** Aggregator functions can be used.
- **LogicalOperands:** The above rules can be combined with *NOT/AND/OR* logical operators.

With our rules above every data quality requirement can be described, which belongs to the mentioned characteristics. And with boolean logic, the validity of a rule can be checked on a dataset.

### B. Visualization techniques

We added plot types for representing the rules as a visualization problem, which is made by the *Rule Visualizer* component:

- **SingleBoundChecker**: Y-axis is variable *A*, X-axis is *Index variable*, the horizontal line is the *THRESHOLD* (maximum or minimum).
- **MultipleBoundChecker**: Y-axis is variable *A*, X-axis is *Index variable*, the horizontal line 1 is $THRESHOLD_1$ (lower) and horizontal line 2 is $THRESHOLD_2$ (upper).
- **SingleValueChecker**: Y-axis is variable *A*, X-axis is *Index variable*, the horizontal line is *VALUE* (equal or not equal).
- **NullValueChecker**: Y-axis is variable *A*, X-axis is *Index variable*, if there is a gap in the plot, then there is a NULL value in that column.
- **ComparingCategorials**: Mosaic plot for comparing two categorical variables.
- Multiple horizontal lines can be in a graph, based on the AND/OR logical statements.
- The horizontal lines can start from a specific X point and also can end in one for representing time intervals.
- If a column is categorical, then we can represent it as integer *1, 2, ..., N-1, N*, where *N* is the number of categories.

Using this visualization method makes it easier to understand the rules and the result of the validation process can also be checked manually.
Assuming that requirements are defined in the rule language proposed in the current paper, the tool offers automated evaluation support. The plots generated automatically by the scripts, which we made.
Using the R [5] programming language, we have created a script for which the user can specify requirements based on the meta-rules, a data set and a column that is responsible for indexing, the program then draws the appropriate graph using the ggplot2 [4] package in R, and lastly, it verifies that the data satisfies the specified rule.

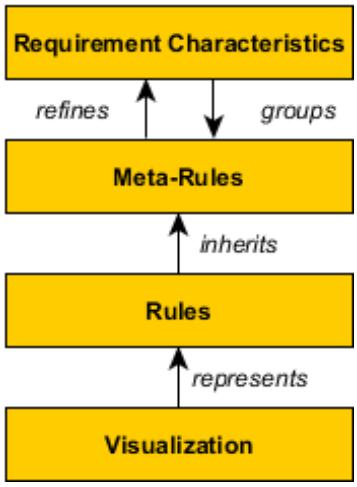



Fig. 2. The connections between the main aspects of our method.

## V. Case Study

The case study is based on measurements performed on a cloud-based distributed service [10] [11] which provides IP-based multimedia communication. The measurements were benchmarking the performance and performability of this service with multiple server-side configurations and workload characteristics. The aim of the measurements was to find early predictors for Service-level Agreement violations. The dataset was created during performance measurement and has 946 rows and 215 columns.

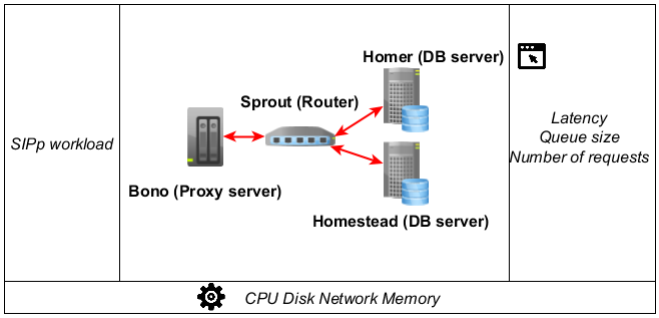



Fig. 3. The architecture of the distributed service.

In the following, we define sample requirements in the dataset containing performance measurement data.

| Characteristics | Requirement |
|---|---|
| *Completeness* | The *TIMESTAMP* column must not contain *NULL* value. |
| *Timeliness* | The *State* value must be calculated before the next granularity period. |
| *Validity* | The *CPUUtil_nice_Sprout* columns must contain only numeric values. |
| *Consistency* | The values of *State_Next* must be equal with the next value of the *State* variable. |
| *Integrity* | The *M_ID* values must be the same for a single measurement data. |

The following table summarizes rules corresponding to the above requirements.

| Characteristics | Rule |
|---|---|
| *Completeness* | Values of variable *TIMESTAMP* must not be *NULL*. |
| *Validity* | Type of variable *CPUUtil_nice_Sprout* must be double. |
| *Consistency* | Values of variable *State_Next* must be the next values of *State* variable. |
| *Integrity* | Values of variable *M_ID* must not be *2022*. |

Fig.4,5 and 6 shot plots created by *Rule Visualizer* component during data validation.

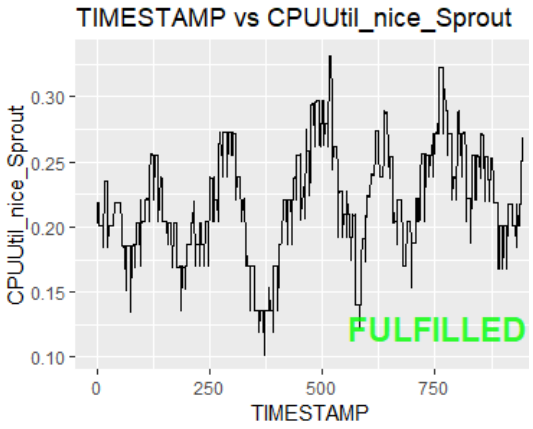



Fig. 4. The rule for Validity characteristic is fulfilled because the black line is continuous.

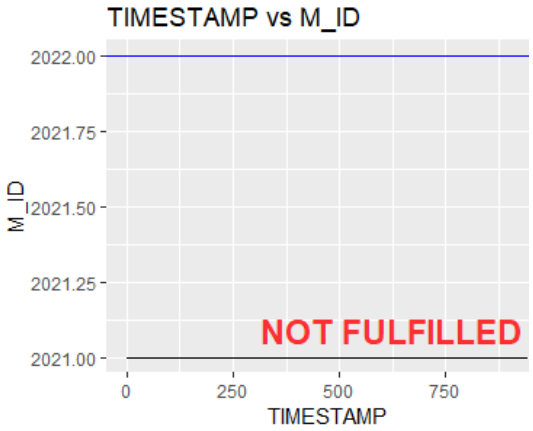



Fig. 5. The rule for Integrity characteristic not fulfilled, because the black line is in a different position from the blue one.

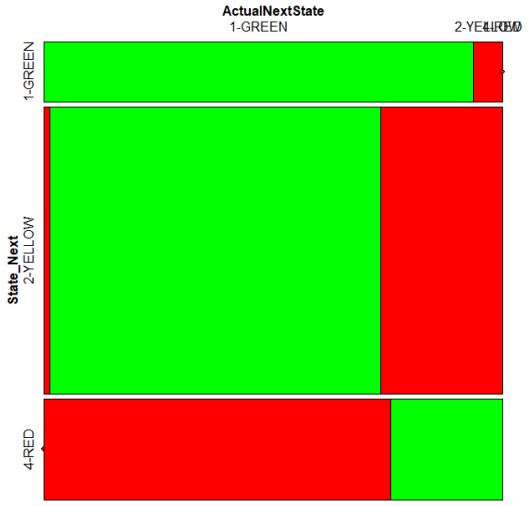



Fig. 6. The rule for Consistency characteristic not fulfilled, because there are red rectangles in the mosaic plot.

```
[1] "Rule: [Values of variable TIMESTAMP must not be NULL] fulfilled"
> checkRule(Rule="Type of variable CPUUtil_nice_Sprout must be double",
+           Data=CloudData,
+           IndexColumn = "TIMESTAMP")
[1] "Rule: [Type of variable CPUUtil_nice_Sprout must be double] fulfilled"
> checkRule(Rule="Values of variable M_ID must be 2022",
+           Data=CloudData,
+           IndexColumn="TIMESTAMP")
[1] "Rule: [Values of variable M_ID must be 2022] not fulfilled"
```

Fig. 7. The result of the Rule Checker component.

## VI. Conclusion and Future Work

During our work, we extended the DQAF method with different meta-rules, which allow formulating requirements with different characteristics for a data set. Also, we have created different types of plots that visually support the verifiability of the requirements.
In the future we will want to extend the script so more visualization methods can be used, which could be better for understanding the requirement analysis.

We consider using Context-aware Timed Propositional Linear Temporal Logic for representing the meta-rules as logical formulas. We plan to add syntax and grammar checking subsystems for our program.
Another important question is runtime correction, to extend the method not only to check the fulfillment of a rule, and also to fix the rule violations if possible.
Currently, our work only uses logical and arithmetic formulas, in the future, we plan to support more complex requirements, e.g., checking relations among multiple variables or distribution of a single variable.
We also plan to add report generation functionality to our tool, which can present the problematic rows and columns and the user can view it whenever he likes.

## VII. Acknowledgements

The project was funded by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013).

## References

[1] ISO/TS 8000 Standard, “Data Quality and Enterprise Master Data”. https://www.iso.org/committee/54158/x/catalogue/
[2] L. Sebastian-Coleman, Measuring Data Quality for Ongoing Improvement: A Data Quality Assessment Framework, 1st ed., 2013.
[3] ISO 9000 Standard, “Quality Management Principles”. https://www.iso.org/standard/45481.html
[4] W. Hadley: ggplot2: Elegant Graphics for Data Analysis. Springer (2010).
[5] R Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (2013). https://www.r-project.org/
[6] A. Pataricza, I. Kocsis, Á. Salánki, L. Gönczy (2013) Empirical Assessment of Resilience. In: Gorbenko A., Romanovsky A., Kharchenko V. (eds) Software Engineering for Resilient Systems. SERENE 2013. Lecture Notes in Computer Science, vol 8166. Springer, Berlin, Heidelberg
[7] L. Gönczy, I. Majzik, Sz. Bozóki, and A. Pataricza. MDD-based design, configuration, and monitoring of resilient cyber-physical systems. Trustworthy Cyber-Physical Systems Engineering, pages 395–420, 2016.
[8] P. Oliveira, F. Rodrigues and P. Henriques (2005). A Formal Definition of Data Quality Problems.. Proceedings of the 2005 International Conference on Information Quality, ICIQ 2005.
[9] P. Oliveira, F. Rodrigues and P. Henriques ”SmartClean: An Incremental Data Cleaning Tool,” 2009 Ninth International Conference on Quality Software, Jeju, 2009, pp. 452-457. doi: 10.1109/QSIC.2009.67
[10] I. Kocsis, Á. Salánki, A. Pataricza. ”Measurement-Based Identification of Infrastructures for Trustworthy Cyber-Physical Systems.” Trustworthy Cyber-Physical Systems Engineering (2016): 369.
[11] D. Burján (2018) ”Application of decision support methods in performance model synthesis” (BSc Thesis, BME VIK MIT)
[12] ISO/IEC 25012 x ISO/IEC 25012 Software-Engineering - Software product Quality Requirements and Evaluation (SQuaRE) - Data quality model. https://www.iso.org/standard/35736.html
[13] Rafique, I., Lew, P., & Abbasi, M.Q. (2012). Information Quality Evaluation Framework : Extending ISO 25012 Data Quality Model.
[14] E. Rahm & H. D. Hong (2000). Data Cleaning: Problems and Current Approaches. IEEE Data Eng. Bull.. 23. 3-13.
[15] Van den Broeck J, Argeseanu Cunningham S, Eeckels R, Herbst K (2005) Data Cleaning: Detecting, Diagnosing, and Editing Data Abnormalities. PLoS Med 2(10): e267. https://doi.org/10.1371/journal.pmed.0020267
[16] W. Hadley (2014). Tidy Data. Journal of Statistical Software, 59(10), 1 - 23. doi:http://dx.doi.org/10.18637/jss.v059.i10
[17] N. Szilvásy, B. Urbán (2015) ”Adatelemzési folyamatok diagnosztikája és adatminőségérzékenység-elemzése” (TDK-dolgozat, BME VIK MIT)

# Model checking and test generation: towards a combined approach to software verification

Mihály Dobos-Kovács*, András Vörös*†
*Budapest University of Technology and Economics,
Department of Measurement and Information Systems,
†MTA-BME Lendület Research Group on Cyber-Physical Systems

***Abstract*—Ensuring the correctness of safety-critical systems is a key aspect of the development process. Various approaches exist to find software bugs: (1) model checking examines the mathematical model of the software and proves the logical correctness, while (2) testing is an efficient and practical technique to find bugs. Model checking is a computationally expensive task, as it explores all the possible states of the software, and despite the technological advances of the last decade, software that cannot be formally verified still exists. On the other hand, testing is computationally cheaper than model checking, and widely used by the industry, but providing an efficient test suite for a given program is still under heavy research.**

**The goal of my work is to combine model checking and testing to exploit the advantages of both approaches. I introduce a new algorithm that uses an abstraction-based model checking technique to explore the behavior of the software. In case the model checking algorithm proves the properties of the software, the procedure terminates. If the algorithm can not reach a conclusion, test generation is applied, exploiting the information gathered during state space traversal of model checking.**

## I. Introduction

Ensuring the correct behaviour of safety-critical systems is an important task during system development. Various approaches exist to find software bugs with their own advantages and disadvantages.

Model checking is one of such approaches that examines the mathematical model of the software and proves the absence of bugs, or provides a counterexample to correctness. Model checking is a computationally expensive task, as it usually explores all the possible states of the program. Despite the technological advances of the last decade, we are still unable to formally verify industrial software systems. There are two barriers: on the one hand, software model checking is an algorithmically undecidable task. On the other hand, when the analysis is restricted to programs with finite memory and finite data structures, state space explosion still prevents successful verification.

Testing [8] is an effective technique to find the flaws in the source code of software systems. Testing is a standard step in every development process, and it is also prescribed by certain standards. The challenge of testing is usually finding a proper test suite, that covers the behavior of the program while still having a feasible size. In the literature, several approaches were published that increased the efficiency of test suits [9] [10] [11] [12].

The goal of this work is to combine model checking with testing to exploit the advantages of both approaches. Model checking explores the state space to find errors or prove the correctness of the software. However, when the verification algorithm reaches a resource limitation, our new approach tries to use up the information gathered during the verification and generate a test set targeting the unexplored part of the program. The approach can focus testing to the critical parts of the program, and we hope that fewer test cases will lead to the more rigorous testing of software systems.

## II. Background

Model checking is a common name of algorithms based on the rigor of mathematics. Model checking takes a formal model and a formal requirement, and verifies if the requirement holds on the model. To utilize model checking on computer programs, they need to be formalized. One of these formalizations is *Control Flow Automata* (CFA) [2], that consists of control locations and operations represented as edges.

One of the model checking algorithms is the so-called *CounterExample-Guided Abstraction Refinement* (or CEGAR for short) [2] [3] [4], that is used in our framework for exploring the behaviour of the software under analysis. The input of the CEGAR algorithm is a formal model (CFA in the case) and a formal requirement. The algorithm either proves that the given requirement holds on the given model, or proves otherwise, by giving a counterexample. Certain locations of the CFA are marked as error locations, and the given formal requirement is that a given error location is unreachable. Such error locations can be created from assertions in the program.

The state space of even a simple program can be huge if not infinite. To tackle this problem, CEGAR uses abstraction, such as explicit value abstraction [2] or predicate abstraction [5] [6]. In our framework, we chose predicate abstraction that follows a set of predicates (Boolean formulas over the set of program variables) instead of the concrete values of the variables. Henceforth an abstract state of a program is a set of (concrete) states that share the same control locations, and a set of predicates describing them.

The core of the algorithm is the so-called CEGAR-loop that consists of two distinct phases: (1) the abstraction and (2) the refinement phase. The task of the abstraction is to build the state space in the form of an abstract reachability

tree with the given set of predicates. If an erroneous state is encountered during the building phase, it is the task of the refinement to determine whether that state is reachable in the concrete state space as well. If an error location is reachable, then the program is unsafe, if it is not then more predicates need to be used [7], and the abstraction continues to build the state space. If the abstract state space contains no error locations, then the concrete state space does not either, as the abstract state space is an over-approximation of the possible state space of the program, so the program is safe.

## III. Overview of approach

In our work, we aim to combine model checking and testing to analyze the safety of software (illustrated on Fig. 1). As a result of limited resources (time, memory, etc.), *formal verification* cannot always succeed. Should that happen, the verification task needs to be terminated, and *test generation* is applied. The test generation method can use the output of the model checking procedure: the *Abstract Reachability Tree*. The role of applying model checking is to decide the correctness of the software (safe depicted as a tick, unsafe depicted as a cross). However, when verification fails to reach a conclusion, then *test running* can still find bugs. If *testing* finds no errors either, then the safety of the program is undecided with the given resources (depicted as a question mark).

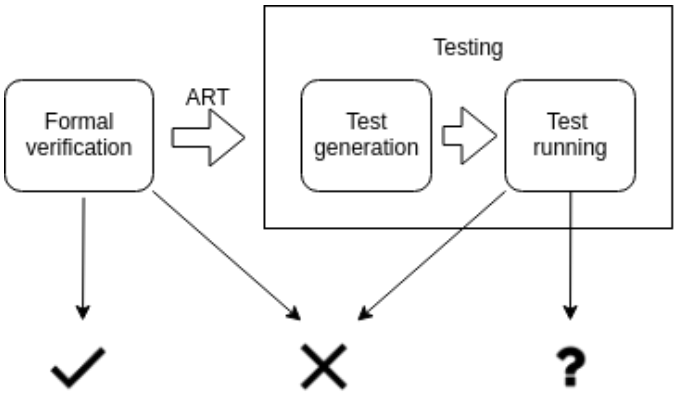



Fig. 1. Combining model checking and testing

Once the CEGAR algorithm terminates, the information gained during the traversal needs to be extracted. The algorithm stores this information in the so-called *Abstract Reachibility Tree*. Each node in the tree corresponds to an abstract state, while the children of the node denote the abstract states reachable via an operation from the parent node. Each node of the tree has one of the following four types:

- *Unreachable*: Nodes whose abstract state is part of the state space, but no input exists that drives the program to these states. These nodes can be removed from the tree.
- *Covered*: If a node $A$ in the tree shares the same control location as a node $B$, and the predicates of $A$ imply the predicates of $B$ , then $B$ is covered (by $A$).
- *Expanded*: A node is expanded if the tree contains the nodes that are reachable from that node via an operation.
- *Incomplete*: All nodes that are not unreachable, covered or expanded are incomplete.

Incomplete nodes represent the ”doorway” to the untraversed part of the state space, as all the possibly reachable states are reachable through them. (All the error states that are reachable from a covered node are reachable from the node that covers it, while all the error states that are reachable from an expanded node are reachable from one of its children.) The goal of test generation is to guide the program through these doorways, which can be achieved by creating an SMT problem [1] from the operations and guards on the path from the root to the incomplete node, and solving the problem for the input variable (note, that this method does not provide any coverage guarantee). This procedure will be detailed in the followings.

*For example, a part of an Abstract Reachability Tree is depicted on Fig. 2. The node with $l_3$ is unavailable as no such $x$ exists that satisfies $3 \leq x < 3$. The node with $l_1$ in the bottom left corner is covered by the node labelled $l_1$ in the center. The node with $l_1$ in the center and the node with $l_0$ are expanded, while the node labelled $l_2$ is incomplete.*

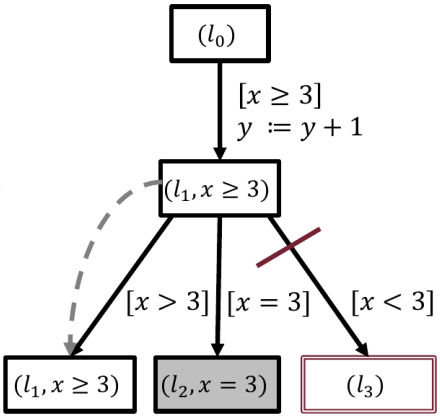



Fig. 2. (Part of an) Abstract Reachibility Tree.

## IV. Generating test cases

Using the information extracted from CEGAR, test cases can be generated utilizing more approaches.

### A. Boundary value analysis of input variables

Boundary value analysis is a black box testing technique, that assumes, that errors happen more frequently at the extreme/boundary values of variables. It is similar to testing based on equivalence partitioning, however, it focuses rather on the corner cases (and does not build equivalence classes explicitly). In our setting, we do local boundary analysis, that is motivated by traditional boundary analysis techniques, but focuses the boundary values by the unexplored part of the state space.

To do boundary value analysis, for each input variable the possible maximum and minimum values should be found. As mentioned earlier, an SMT problem can be constructed out of the operations on the path to an incomplete node.

Solving this problem gives one possible combination of many for the input variables. By giving the solver an optimization constraint, such as the value of one of the variables should be minimal/maximal, such a solution can be found, where the given variable is on one of its boundary values. Some solvers can solve the optimization problem [13].

The solution of the optimization problem is a combination of input values that guides the execution to the given incomplete node, while one of the input values is minimal/maximal. This minimal/maximal value can differ for the same variable if an other incomplete node is reached. These minimal or maximal values are local: the program can accept lower/higher input values than these boundary values, but on the given path, and on the state space that is reachable from the given incomplete node, these are the local minimal/maximal values. The computed values focus onto the unexplored part of the software.

To apply boundary value analysis systematically, the SMT problem should be solved twice for each input variable: for the first time the optimization constraint should be to minimize the current variable, for the second time to maximize it. Out of each solution of the SMT problem, a test case can be generated that tests the software for the minimal/maximal value of one input variable.

### B. Robustness testing

The philosophy behind robustness testing is similar to boundary value analysis. The difference is that by robustness testing the errors are assumed to happen on the extremes of arithmetic conditional expressions.

The process of finding the necessary input values is similar to the method described in the previous subsection. The SMT problem with an optimization constraint needs to be solved, and using the solution, a test case can be generated. However the optimization constraint is not to minimize/maximize one of the input variables rather to minimize/maximize one of the variables that happen to be a result of an arithmetic expression.

*For example, let us assume that $(z \leq 5)$ arithmetic conditional expression is given in a guard, where $z$ is a positive integer. According to robustness testing the errors are more frequent on the extremes, so the possible minimal and maximal value of variable $z$ should be determined, and used in the test cases. As the value of $z$ might depend on the value of other variables, an SMT problem needs to be constructed and solved, as described earlier.*

To apply robustness testing systematically, again the SMT problem should be solved twice for each variable in an arithmetic expression: for the first time the optimization constraint should be the minimization of the variable, for the second time the maximization. These test cases test the software for errors that happen in arithmetic conditions, for an input or computed variables on boundary values.

### C. Finding number representation errors

In a computer program, every variable is stored on a finite number of bits. As a result, the range of every variable is a finite set (all the integer have a minimal and a maximal value, the floating-point variables are stored using the exponent and mantissa, etc.). A number representation error occurs when such a value is reached during the running of the software, that cannot be represented using the type of the variable.

*For example. Let $x$, $y$, $z$ be 4 bit unsigned integers (meaning, that the finite range of these variables is $\{0, 1, ..., 15\}$. Let $x = 8$ and $y = 8$ hold. If $z = x + y$, then $z$ should be 16, which is not part of the domain of the variable, so it cannot be represented. This kind of error is called overflow/underflow, and a common problem in embedded systems.*

Formal methods should take into consideration these characteristics of real-life program variables. Model checking does logical analysis (in our specific use-case; other model checkers can do bit-precise verification as well), so the range of every integer variable is the set of integers ($\mathbb{Z}$), while the range of every floating point variables is the set of real numbers ($\mathbb{R}$). As a result, formal methods may miss some software bugs that are related to the representation of numbers.

Finding these kinds of errors is different from the earlier methods: as it does not aim the untraversed part of the state space, rather the traversed one. The aim is to find errors, that model checking might have missed. To identify these errors, those variables should be found first, whose value might be unrepresentable. These variables are those that store the result of an arithmetic operation (such as adding, multiplying, etc. variables). This information can be extracted from the source code.

```
1. unsigned x; scanf("%u", &x);
2. unsigned y; scanf("%u", &y);
3. unsinged z = x + y;
4. if(z <= 5) {
5.     ...
6. } else {
7.     ...
8. }
```

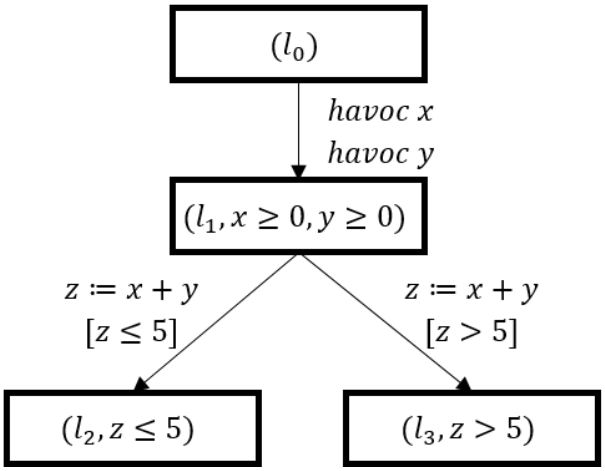



Fig. 3. A C code, and the corresponding ART (fraction)

Numerous SMT problems can be constructed from the operations and guard expressions on the path to the leafs of the ART. However, new constrains must be added, that state the possible range of each input variable. The optimization constraint should be the minimization/maximization of the variables under analysis (whose value might be unrepresentable). If in the solution of the SMT problem the value of the variable is outside the representable range, then an overflow/underflow occurred. Again, using the solution a test case can be generated that reproduces the error.

There are other kinds of number representation errors as well, but they are not discussed here.

## V. Case study

On Fig. 3 a simple C program is depicted, that receives two input numbers and behaves differently based on their sum. A part of the source code is depicted below, with a fraction of the abstract reachability tree corresponding to the code. The root of the fraction is $l_0$ while the incomplete nodes are $l_2$ and $l_3$. The three methods described earlier will be presented using the paths from $l_0$ to $l_2$ and $l_3$. Furthermore let us assume that all variables are unsigned integers.

To apply boundary value analysis on the left hand side (from $l_0$ to $l_2$), the input variables should be found first. These variables are $x$ and $y$. Therefore the optimization constraints and the solutions of the SMT problem will be the following:

- $max(x)$: $\{(x = 5), (y = 0)\}$
- $min(x)$: $\{(x = 0), (y = 0)\}^*$
- $max(y)$: $\{(x = 0), (y = 5)\}$
- $min(y)$: $\{(x = 0), (y = 0)\}^*$

To apply robustness testing on the left hand side (from $l_0$ to $l_2$), the variables in arithmetic conditions should be found first. The only variable is $z$, because of the $[z \leq 5]$ condition. Therefore the optimization constraints and the solutions of the SMT problem will be the following:

- $max(z)$: $\{(x = 5), (y = 0)\}^*$
- $min(z)$: $\{(x = 0), (y = 0)\}$

To find errors of number representations on the right hand side (from $l_0$ to $l_3$), the variables that can overflow/underflow need to be identified first. The only variable is $z$, because the it is the only variable that stores the result of an arithmetic operation ($x + y$). Let us assume, that the range of $x, y, z$ is the integers between 0 and 15 (4 bit unsigned integer). The optimization constraints and the solutions of the SMT problem will be the following:

- $max(z)$: $\{(x = 15), (y = 15)\}^*$
- $min(z)$: $\{(x = 3), (y = 3)\}^*$

The first case, when both $x$ and $y$ are 15, the value of $z$ is 30, so an overflow occurred.

## VI. Conclusion

Ensuring correctness is a key aspect of the development process in the safety-critical domain. However, it is not a trivial task. Existing approaches, such as model checking and testing both have their advantages and disadvantages: model checking can prove the correctness for the price of heavy computations, while testing can efficiently find bugs in software.

By combining them, it is possible to exploit the advantages of both worlds. If the verification has enough resources to complete the task, then the correctness can be decided. If it is aborted as the resources are not sufficient, the information gathered during the state space traversal can be used to generate test cases focusing on the unverified part of the state space.

The novelty of the presented approach is that by using the information provided by the verification algorithm, the targeted test suite can be generated that results in fewer test cases.

### A. Future Work

There is much work left. In the following, we introduce some important directions:

- In the future, further test generating methods (eg. based on equivalence classes) need to be developed to cover a greater part of the state space.
- More CEGAR abstraction methods are needed to be analyzed to extend our method.
- A common pattern in software is input inside a cycle, which often breaks abstraction. Methods need to be devised to provide values for such inputs during test case generation. That is the main weakness of our approach.

## References


[1] L. De Moura and N. Bjørner, "Satisfiability modulo theories: introduction and applications," Communications of the ACM, vol. 54, no. 9, pp. 69-77, 2011.

[2] D. Beyer and S. Löwe, Explicit-State Software Model Checking Based on CEGAR and Interpolation, Lecture Notes in Computer Science, vol. 7793, pp. 146-162, 2013.

[3] E. Clarke, O. Grumberg, S. Jha, Y. Lu and H. Veith, Counterexample-guided Abstraction Refinement for Symbolic Model Checking, J. ACM, vol. 50, no. 5 pp. 752-794, 2003.

[4] Á. Hajdu, T. Tóth, A. Vörös and I. Majzik, A configurable CEGAR framework with interpolation-based refinements, Lecture Notes in Computer Science, vol. 9688, pp. 158-174, 2016.

[5] S. Graf s H. Saidi, Construction of abstract state graphs with PVS, Lecture Notes in Computer Science, vol. 1254, pp. 72-83, 1997.

[6] D. Beyer and M. Dangl, SMT-based Software Model Checking: An Experimental Comparison of Four Algorithms, Lecture Notes in Computer Science, vol. 9971, 2016.

[7] K. L. McMillan, Applications of Craig interpolants in model checking., Lecture Notes in Computer Science, vol. 3440, pp. 1-12, 2005.

[8] I. S. T. Q. Board, Certified Tester Foundation Level Syllabus, 2018.

[9] N. Tillmann, J. de Halleux and T. Xie, Pex for Fun: Engineering an Automated Testing Tool for Serious Games in Computer Science, 2018.

[10] J. de Halleux and N. Tillmann, Moles: Tool-Assisted Environment Isolation with Closures, Lecture Notes in Computer Science, vol. 6141, pp. 253-270, 2010.

[11] C. Cadar, D. Dunbar and D. Engler, KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs, in Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation, 2008.

[12] G. LiIndradeep, G. Sreeranga and P. Rajan, KLOVER: A Symbolic Execution and Automatic Test Generation Tool for C++ Programs, Lecture Notes in Computer Science, vol. 6806, pp. 609-615, 2011.

[13] L. M. d. Moura s N. Bjørner, Z3: An Efficient SMT Solver, TACAS, 2008.


*One out of many possibilities

# Design for dependability of cyber-physical systems

András Földvári
*Budapest University of Technology and Economics*
*Department of Measurement and Information Systems*
fandras95@gmail.com

András Pataricza
*Budapest University of Technology and Economics*
*Department of Measurement and Information Systems*
pataric@mit.bme.hu

***Abstract*—Cyber-Physical Systems (CPS) are smart frameworks integrating the physical and the computational worlds. CPS plays an increasingly important rule in a variety of application domains.**

**Building and managing CPSs is a highly complex task. Integration of the required services and assurance of the dependability necessitate the fulfillment of a variety of functional and extra-functional requirements.**

**However, the assurance of the compliance of application-dependent requirements remains a primary task for all over the architecture design, component development, system integration process. Formal proof of correctness is increasingly used for V&V of critical systems.**

**The interaction between components of a complex CPS is the primary source of faults in integration-based system composition. Integration testing is mandatory, even if the components are typically pretested. Dependability assurance in a dynamic CPS necessitates the extension of the verification of the interoperability from design-time to runtime.**

**Our research objective was the elaboration of a method assuring the dependability aspects of the designated system both at design and runtime by supporting the system integration by reusing component tests for integration testing and runtime verification. Our method extends the Assume-Guarantee approach elaborated by NASA in the '90s for testing to a general-purpose runtime verification paradigm.**

***Index Terms*—assume-guarantee, cyber-physical systems, dependability**

## I. Introduction

### A. Objective

Cyber-Physical Systems (CPS) are smart frameworks integrating the physical and the computational worlds [1].

The conceptual design of CPS (Fig. 1) shows a continuous interaction between (sub)systems, components, physical environment and users [2]. A proper interaction is a prerequisite for supervisory and control functions carried out by a CPS. Even in the case of occurrence of minor faults, the physical environment can potentially amplify the impact of a them to a catastrophic failure.

Dependability of a service delivered by a critical CPS necessitates the fulfillment of a variety of functional and extra-functional requirements.

The project was funded by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013) and SCCH Hagenberg by providing a grant for a summer internship.

The built-in features of modern CPS runtime platforms like those implementing the OMG standard Data Distribution Service (DDS) [13] or similar frameworks [11] provide extensive support for a variety of core QoS attributes like security, scalability, maintainability, throughput, and timeliness in hard real-time applications, etc. Moreover, modern platforms support dynamic, operation time system reconfiguration for adaptive systems, on-demand CPS application synthesis and multi-purpose use of CPS infrastructures.

### B. System integration and dependability assurance

The architecture composition paradigm of complex CPSs is component integration supporting dynamic integration of physical and computational resources and software components in a service-oriented way, as well. Service orientation causes here a strong correlation between the topology of the system (components) and the flow of data (communication).

Our research objective was the elaboration of a formal proof of correctness-based method assuring primarily the integrity, safety, availability, and reliability of the designated system at design and runtime.

The integration of third-party components and sub-services is challenging as they are frequently known only at a black-box level by their interface definition and specification (input-output invariants).

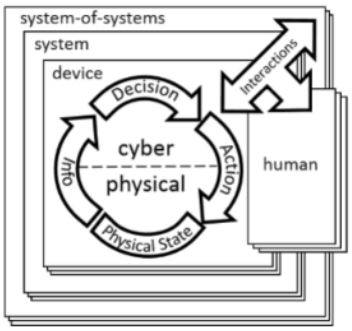



Fig. 1. Conceptual Design of Cyber-Physical Systems [2]

However, acceptance and compliance tests of the component are typically provided to or elaborated by the system integrator. This provides a potentially valuable asset for assuring

dependability of the system by supporting system integration and runtime verification.

The starting point of our method is a well-founded approach for reusing component tests in the system integration test phase. Test oracles implement the assumptions on the use of the component (input invariants) and check compliance of their output to the specification. NASA developed a methodology called assume-guarantee (A-G) [3] [4]for reusing the component tests together with their oracles for testing the system integration. As a component and its environment have to fulfill the same requirements after integration as during component testing.

A-G upholds a ”divide-and-conquer” approach [4]. It checks proper embedding of a component into the system by checking *a*) the compliance of its inputs potentially generated by other components to the input specification and *b*) checking the individual component outputs against their respective specifications *c*) checking the composite outputs of multiple components with respect to system specification. This technique has long held promise for modular verification of the system integration [4] by successively applying it in a component by component integration process.

Furthermore, our extension exploits the fact, that the oracles can be reused for runtime verification by embedding them into the target system as monitoring components. Continuous execution of the same checks during runtime as during design time promises the following benefits: *a*) even a perfectly tested component may fail during runtime if temporal faults occur in its runtime environment; *b*) errors originating in the physical environment can be detected, and their propagation can be blocked prior to causing failures; *c*) runtime verification may compensate for the incomplete fault coverage of the design time tests by detecting hidden design faults and preventing catastrophic consequences. Finally in dynamic architectures system integration becomes to a runtime functionality necessitating a proper check.

Here the design of test oracles and monitoring components relies on a formal specification of the component input and output invariants by means of temporal logic expressions formalized as state machines. Automated code generation implements the oracles and monitoring components from formal models.

### C. Structure of the Paper

Section II presents a methodology for adapting the general purpose model-driven design approach for dataflow oriented architecture design. Section III introduces the A-G approach and Section IV presents the novel runtime verification process based on the A-G artifacts.

## II. Adaptation of Model-Driven Engineering

Component-based integration of complex systems necessitates a clean architectural paradigm for component interconnection, especially in dynamic system architectures.

Modern CPS platforms support a direct mapping of the functional topology of component interactions to the implementation configuration of interconnection. This dataflow oriented integration offers simple and general purpose interfaces to the individual components, while a descriptor of the functional interaction defines their interconnection. Implementation and management of the interconnection fabric is an exclusive task of the runtime middleware. For instance the publish-subscribe principle (Fig. 2) provides a pure dataflow view for the application integration. QoS is managed by the runtime platform. Structural changes originating in dynamic application composition can be managed similarly at the platform level.

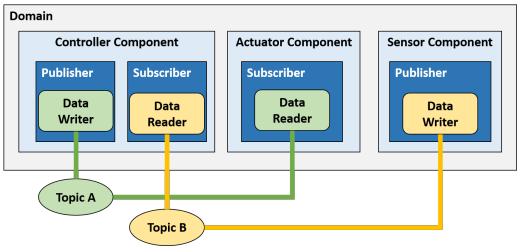



Fig. 2. Data Distribution Service

An additional benefit of using standardized interfaces for interconnection is that any component can be extended by checkers without changing the functional interfaces. Cascading input checkers - functional component - output checkers (Fig. 3) results in a ”self-checking component”. This way runtime verification can be seamlessly integrated into the functional topology and implementation architecture.

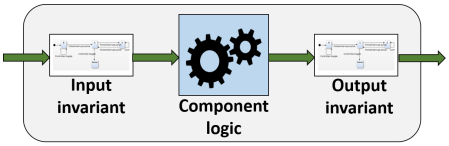



Fig. 3. Input checker - Functional component - Output checker

Model-driven engineering is a software and system design paradigm for the development of component-based systems. It structures the specification and the specification based requirements in the form of models.

As dataflow oriented system implementation maps the platform independent model (PIM) of the target application in an unchanged form directly to the implementation, thus a simplified adaption of the classic model-driven development (MDD) [5] approach was used in this paper (Fig. 4).

The design workflow is confined to the high-level functional architecture (e.g., dataflow diagram) complemented with the specification of the extra-functional requirements. For instance, SysML based design (the dominating MDD design paradigm for CPS) can be carried out by using only the diagrams defining the requirements and the functional architecture of the target system by a dataflow diagram.

Usually, the functional decomposition of a system defines both static and dynamic requirements. However, the majority of the extra-functional requirements is static, like those

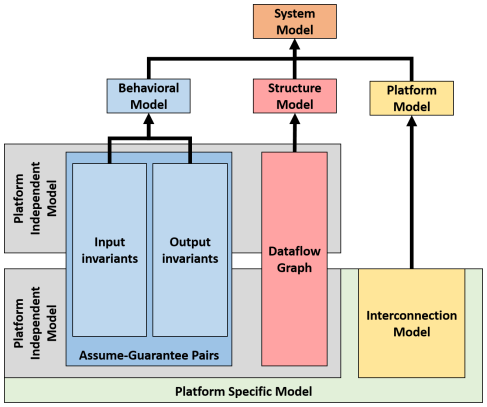



Fig. 4. Model-Driven Engineering

related to structure and platform are responsible for representing the structure (architecture, composition, components) and platform (deployment, integration, throughput). Runtime verification of the soundness should be integrated into the reconfiguration process verifying the compliance of the designated new configuration with the requirements in a similar way as configuration checking is executed during design time.

Dynamic requirements (e.g., resilience aspect, temporal) will be checked by introducing input-output monitors around the components in order to assure their self-checking.

In both cases, management of alerts after error detection should happen at the system level.

## III. Assume-Guarantee Approach

Originally, A-G reasoning targeted the stepwise development of concurrent processes. Over time, A-G reasoning was extended to cover the entire V-model of the development process from the design-level verification [3] to testing [4]. The design-level A-G artifacts guide the implementation phase and provide efficient reasoning at the implementation level.

Assumptions and properties originate in dynamic requirements. The A-G artifacts (components, assumptions, properties) are represented as labeled transition systems in this section.

Usually, developers have behavioral information about the interfaces. The environment (other components in the system) of the component is capable of invoking operation sequences at the component's interface. Assumptions define and restrict the behavior of the environment. Guarantee properties are the acceptable and required operation of the component. If $A$ is an assumption and $P$ is a guarantee property, then $A \rightarrow P$ indicates that $P$ works correctly in an environment restricted by $A$.

A-G artifacts are interpreted in terms of testing as follows: assumptions are functions used on test sequences (pre-invariants) to determine if the operational conditions of the component under test are correct. Properties (post-invariants) are criteria which the component should comply with during operation. If the test sequences and the test results were both correct (valid), the component guarantees its property and integrates well into the system.

Finding proper assumptions is difficult. The assumption can be more abstract (less permissive) than the actual implementation of component under evaluation, in order to avoid escaping integration faults due to the use of overspecified components.

Assumption generation provides automated support for A-G reasoning. Assumption $A_w$ holds for the weakest assumption [7] and characterizes all possible environments $E$ under which the assumption holds. Techniques were developed to generate the weakest environment assumptions [6] that enable the property to hold.

## IV. Runtime Verification

### A. Architectural Change Management

Numerous CPS architectures require dynamic reconfiguration for the sake of fault-tolerance or demand-driven adaptation of the functionality. By principle, verification of the integration is impossible during the design time. Moreover, the usual limitations on the execution of the reconfiguration prohibit the exhaustive testing of the new setup.

Changing a component (reconfiguration, extension) necessitates an update of the related artifacts (assumptions, properties) to perform the verification.

Dynamic CPS middlewares (e.g., DDS, OPC UA[1]) facilitate reconfiguration and automatic discovery of components by standard services. Several gateways (e.g., DDS-OPC UA gateway [9]) are available to interconnect the different middlewares.

### B. Assume-Guarantee Runtime Verification

The A-G approach is reusable for runtime-verification. This part presents the method and the architecture (Fig. 5) of the approach developed by us for implementing this idea.

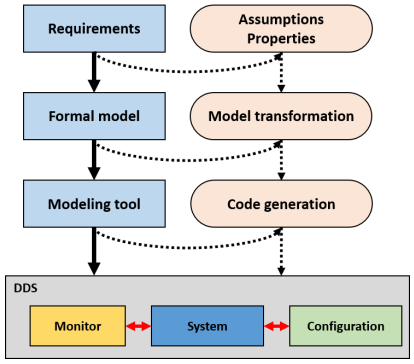



Fig. 5. Assume-Guarantee Runtime Verification

The main idea is the creation of self-checking components (as illustrated before in Fig. 3) by adding input and output monitors derived directly out of the requirements.

[1]OPC Unified Architecture (OPC UA) is a machine to machine communication protocol for industrial automation.

*1) Verification Method:* Assumptions and properties are derived in the early phase of the design process from the requirements and specifications. In traditional methodologies, the oracles implemented for testing purposes [3] [4] remain further unexploited byproducts after testing. The monitors used for runtime verification are developed in a separate component development phase.

Test oracles and monitors run in different setups (e.g., over a tester and the operating system), their implementations cannot be directly reused. However, as checks for testing and runtime verification share the identical set of assumptions and properties, MDD provides an easy way to reuse the implementation efforts. If the A-G pairs are formulated as a platform independent model (PIM) sharing a single formal model, automated code generation can produce the actual code for testing and runtime verification respectively. In our case, linear temporal logic (LTL) serves as a basis of the PIM A-G artifacts.

The method uses statecharts to represent both the component behaviors and the A-G pairs. The first step is to design and generate statecharts models from high-level requirements. The safety (assumptions and properties) LTL expressions also translate into statecharts via automated model transformation [10].

In our approach, the Gamma Statechart Composition Framework [12] (a YAKINDU extension) serves to model both the functional components and the runtime monitors. The DDS extension for Gamma provides the interconnection between the participants using DDS.

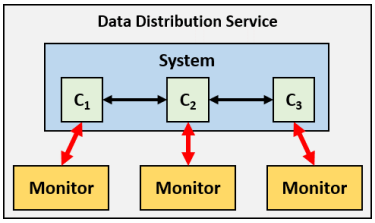



Fig. 6. System and monitors

The generated system code and the monitors are standard DDS components (Fig. 6). As DDS supports distributed environment, it is possible to deploy the code and the monitor over independent resources thus avoiding the malicious impact of correlated fault originating in shared resources.

*2) Error handling:* It is possible to identify malfunctions (error detection) during runtime. After the identification of a fault, the monitors can raise alerts (interrupts) to the affected components. Error handling (e.g., fault-tolerant patterns) is the responsibility of the system components.

*3) Implementation details:*

1) **System models and LTL expressions**: System models were created with the MagicDraw modeling tool. The A-G LTL expressions were created manually, based on the behavioral models.
2) **Statecharts**: Yakindu Statechart Tools helped to model both the system behavior models and the A-G monitors. Gamma provided the interconnection of the individual statechart components.
3) **Code generation**: Gamma offered a built-in code generator that generated DDS specific code from the statecharts and interface specifications.

## V. Summary

The solution presented reuses pre-tested formal verification engines as a byproduct of the original A-G based testing for runtime verification. MDD is used for the efficient generation of the test oracles and the runtime monitors. The method extends the widely used A-G method to runtime verification by combining the principle with modern implementation technologies and exploiting the architectural paradigms in modern CPS middleware platforms.

### A. Further work

The current implementation of the tooling supports static architectures. The verification of static architecture-related requirements is an ongoing effort by adopting the constraint satisfaction programming based approach used in design time for checking intended dynamic architectural changes [8].

## VI. Acknowledgment

The authors acknowledge the professional guidance and tutoring of Dr. Josef Pichler (SCCH Hagenberg, Austria).

## References

[1] A. Karmarkar and M. Buchheit, “The Industrial Internet of Things Volume G8: Vocabulary,” tech. rep., Industrial Internet Consortium, 2017.
[2] E. Griffor, D. Wollman, and C. Greer, “Framework for cyber-physical systems,” tech. rep., National Institute of Standards and Technology - Cyber Physical Systems Public Working Group, 2016.
[3] D. Giannakopoulou, C. S. Pasareanu, and J. M. Cobleigh, “Assume-guarantee verification of source code with design-level assumptions,” in *Proceedings of the 26th International Conference on Software Engineering*, pp. 211–220, IEEE Computer Society, 2004.
[4] C. Blundell, D. Giannakopoulou, and C. S. Păsăreanu, “Assume-guarantee testing,” in *ACM SIGSOFT Software Engineering Notes*, vol. 31, p. 1, ACM, 2005.
[5] E. Ovaska, A. Balogh, S. Campos, A. Noguero, A. Pataricza, K. Tiensyrjä, and J. Vicedo, “Model and quality driven embedded systems engineering,” *Technical Research Centre of Finland*, 2009.
[6] D. Giannakopoulou, C. S. Pasareanu, and H. Barringer, “Assumption generation for software component verification,” in *Automated Software Engineering, 2002. Proceedings. ASE 2002. 17th IEEE Int. Conf. on*, pp. 3–12, IEEE, 2002.
[7] H. Mehrpouyan, D. Giannakopoulou, G. Brat, I. Y. Tumer, and C. Hoyle, “Complex engineered systems design verification based on assume-guarantee reasoning,” *Systems Engineering*, vol. 19, no. 6, pp. 461–476, 2016.
[8] A. Földvári, ”Design for dependability of cyber-physical systems”, B.Sc. Thesis, Department of Measurement and Information Systems - Budapest University of Technology and Economics, 2018.
[9] “OPC UA/DDS Gateway.” https://www.rti.com/blog/announcing-the-opc-ua-dds-gateway-standard.
[10] G. Pintér and I. Majzik, “Runtime verification of statechart implementations,” in *Architecting Dependable Systems III*, pp. 148–172, Springer, 2005.
[11] “OPC Unified Architecture.” https://opcfoundation.org/.
[12] “Gamma statechart compostition framework.” https://inf.mit.bme.hu/en/gamma.
[13] “Industrial Internet Reference Architecture.” https://www.iiconsortium.org/IIRA.htm.

# Increasing the accuracy of measuring heart rate variability and pulse wave transit time

Péter Nagy, Ákos Jobbágy
Budapest University of Technology and Economics,
Department of Measurement and Information Systems,
Budapest, Hungary
Email: {nagy, jobbagy}@mit.bme.hu

***Abstract*— Blood pressure is strongly influenced by the stress level of the individual. High level of emotional or physical stress may lead to blood pressure values not correctly reflecting the state of the cardiovascular system. Using ECG and photoplethysmographic (PPG) sensors, heart rate variability and pulse wave transit time can be measured and used to assess the stress level of the individual prior to and in parallel with blood pressure measurement. This paper investigates the effect of noise on the accuracy of the calculated heart rate variability (HRV) and pulse wave transit time (PWTT) values using simulated noisy ECG and PPG signals. The effect of physical stress on PWTT and HRV is also investigated. Results are illustrated by simulations and real data from a measurement including physical stress for the tested person.**

***Keywords—heart rate variability; pulse wave transit time; stress level assessement; R-peak detection***

## I. Introduction

Blood pressure (BP) is one of the most important vital signals providing information about the state of the cardiovascular system. Accurate BP measurement is essential for optimal diagnosis and treatment of cardiovascular diseases. Autonomic, humoral, mechanical and myogenic factors as well as environmental stimuli influence the momentary value of BP [1]. Stress level of the examined person can have a large impact on the measurement results and may induce incorrect medical conclusions if high stress level remains undetected. Commercially available automated blood pressure monitors display the average heart rate as a measure of stress level. Our goal is to provide a meaningful parameter better characterizing the stress level of the tested person. We use ECG and PPG sensors for the determination of heart rate variability (HRV) and pulse wave transit time (PWTT). These parameters are appropriate to determine whether the person – whose BP is about to be measured – is at rest.

HRV is a commonly used measure to determine momentary stress level. It characterizes the variation of the beat-to-beat time intervals [2].

PWTT is the time while the pressure wave generated by the heart propagates from the aortic valve to a peripheral part of the body. PWTT is influenced by heart rate, blood pressure and arterial stiffness therefore it is a potential indicator of the complex cardiovascular response to physical and psychological stress. Hey et al. [3] applied PWTT successfully to identify stressful moments during the Trier Social Stress Test. However, in relaxed sitting position, the effect of mental stress on PWTT may be not significant [4] [5], therefore the analysis of heart rate and HRV is advantageous even if PWTT is registered.

## II. Materials and Methods

### A. Characterizing HRV

Calculation of HRV using oscillometric pulses or characteristic points in the PPG signal may be inaccurate, because of the beat-to-beat variation of the propagation time of the pulse wave from the heart to the cuff and to the PPG sensor. Investigation of the differences in calculated beat-to-beat time intervals based on PPG and ECG signals is described in [6]. Accurate determination of HRV is possible using the ECG signal if R-peaks are accurately detected.

HRV can be characterized by various measures both in time domain and in frequency domain. For short time recordings, time domain analysis is required to characterize HRV [6]. Here, usually the lengths of heart periods (tRR, NN (the time interval between normal R-peaks)) are examined. We investigated the distribution of NN intervals and the differences in successive NN intervals.

### B. Characterizing PWTT

PWTT can be calculated using ECG and PPG signals. We measured PPG signals at the left index fingertip. PWTT was defined as the time difference between the R-peak in the ECG signal and the corresponding local minimum in the PPG signal. The bigger the distance between the PPG measurement site and the heart is, the less impact uncertainty of local minimum designation has on the calculated PWTT values (the same absolute error leads to a smaller relative error) [3].

### C. Simulating Noisy ECG Signal

Accurate detection of R-peaks in the ECG signal is essential for both HRV and PWTT determination. In order to investigate the effect of noise on the accuracy of the R-peak detection, a noisy ECG signal was simulated based on [7]. One period of the noise-free ECG was obtained by averaging

periods of an ECG record with high signal-to-noise ratio. The single period of the ECG was copied and appended to itself repetitively in order to form 100 periods of the ECG with approximately 78 s total duration. Beat-to-beat time interval variation was added by extending each period to a randomly selected length. Maximal increase of the beat-to-beat time interval was 50 ms. Five types of noise artifacts were added to the noise-free signal: power line interference; motion artifacts; muscle contraction (electromyographic interference); baseline drift and amplitude modulation with respiration.

### *D. Comparing Different Pre-Processing Steps of R-peak Detection*

Most QRS detection algorithms use a filter stage prior to the actual detection in order to attenuate other signal components and artifacts [8]. Elgendi et al. [9] investigated the optimum bandpass frequency range for the detection of the QRS complexes and recommended a bandpass frequency range of 8-20 Hz for the best signal-to-noise ratio. For the accurate calculation of PWTT and HRV, not only the QRS complexes but also the R-peaks must be precisely detected. Therefore, we investigated the effect of filtering on the signal shape in the time domain. For this purpose, we used the simulated noisy ECG signal where the noise-free signal, locations of the R-peaks and the superimposed noise parameters are exactly known. The Pan-Tompkins algorithm [10] was used to detect QRS complexes in the simulated ECG signal. After detection of QRS complexes, the unfiltered signal was re-filtered, independently of the filtering in the Pan-Tompkins algorithm. Notch filtering at 50 Hz and 100 Hz, and lowpass filtering at 50 Hz were used. The effect of median filtering was also investigated. We also assessed the performance of using only lowpass filtering at 25 Hz. R-peaks were detected in the re-filtered signal as local maxima within the +/- 80 ms neighboring intervals of the R peaks designated by the Pan-Tompkins algorithm.

### *E. Examining the Effect of Filtering on Detection of Local Minima in the PPG Signal*

Filtering of the PPG signal is necessary for PWTT calculation. Baseline wandering (mainly caused by breathing) and motion artifacts have to be removed for the accurate detection of local minima in the PPG signal. Elgendi et al. [11] examined the spectrum of the PPG signal and suggested a second-order Butterworth bandpass filter with cutoff frequencies at 0.5 Hz and 10 Hz for PPG-based HRV analysis. However, the distortion caused by filtering has a large effect on the locations of local minima in the PPG signal. In order to investigate this effect, we used a simulated noisy PPG signal which was generated similarly to the simulated noisy ECG signal. Power line interference, baseline drift and amplitude modulation with respiration was added to the noise-free signal. The simulated signal was filtered with different cutoff-frequencies and the locations of detected local minima were compared to the reference values.

### *F. Experimental Data*

For confirmation of the simulated results, real data were also analyzed from a measurement where short-term physical stress was induced for the tested persons by running 3 floors downstairs then 3 floors upstairs. Three persons (3 males), one healthy senior adult and two healthy young adults participated in the measurement. More than 20 measurements were recorded for each tested person. Data were recorded before (referred to as resting state) and after the physical stress. Recording length was 120 seconds. Experimental data were recorded by a custom-developed home health monitoring device. The device measures ECG in Einthoven I lead and PPG signal at the fingertip using a transmission-type PPG sensor. The sampling frequency was 1 ksample/s.

## III. RESULTS

### *A. R-peak Detection in the Simulated Noisy ECG Signal*

In order to compare the effect of different filtering techniques on the accuracy of R-peak detection, the mean absolute time difference between the known R-peaks and the R-peaks detected in the re-filtered signals was calculated as described in the chapter *Comparing Different Pre-Processing Steps of R-peak Detection*. Results are summarized in Table I. For the lowpass and notch filters, the order 3 was chosen. For the median filter, the order 10 yielded the best results. Clifford et al. [12] concluded that an error above 1 ms in the location of detected R-peaks can be considered to be significant for HRV analysis. The best two results in Table I are below this limit.

TABLE I. COMPARISON OF DIFFERENT PRE-PROCESSING STEPS OF R-PEAK DETECTION

| **Filter type** | **Mean absolute error (ms)** |
|---|---|
| Two notch filters, center frequencies at 50 and 100 Hz and lowpass filter, cutoff-frequency at 50 Hz | 0.67 |
| Two notch filters, center frequencies at 50 and 100 Hz | 1.94 |
| Lowpass filter, cutoff-frequency at 25 Hz | 0.76 |
| Median filter | 1.60 |
| No filter | 5.09 |

### *B. The Effect of Physical Stress on PWTT and HRV*

Our goal was to compare the effect of physical stress on average heart rate to the effect of physical stress on PWTT and HRV. For this comparison, average heart rate and momentary and average values of PWTT and HRV were calculated for the measurements where short-term physical stress was induced for the tested persons. Data were recorded in resting state and after physical stress. Average values were calculated in a sliding window containing 30 heart periods and the window was stepped by 1 heart period. HRV was characterized by the absolute value of differences in successive NN intervals (dtRR), the ratio of differences exceeding 50 ms to the total number of differences (pNN50), and the ratios of differences between 0 and 20 ms as well as 20 and 50 ms to the total number of differences (pNN0_20 and pNN20_50) [6]. For the change in PWTT following physical stress, we found that even the trend of change was not the same for the three tested persons.

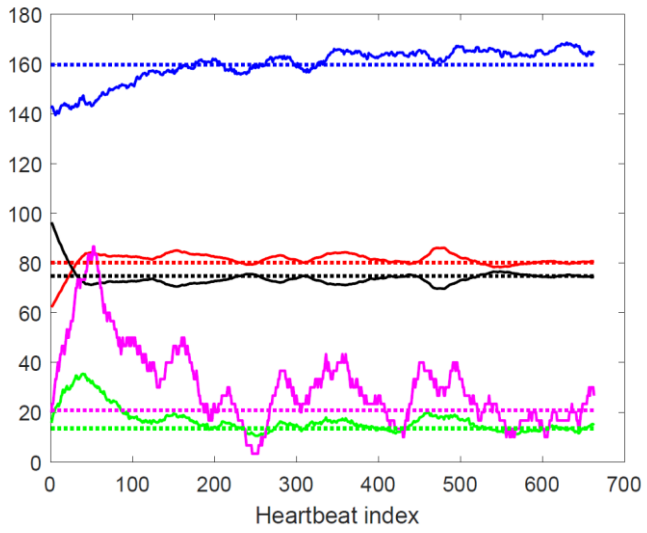



Fig. 1. Time functions of parameter values averaged in a sliding window containing 30 heart periods: Blue: PWTT (ms); Red: tRR (10 ms); Black: heart rate (1/min); Magenta: pNN20_50 (%); Green: absolute dtRR (ms). Dotted lines represent the average value of parameters measured in resting state.

In order to investigate the time course of the calculated parameters in a longer period, we recorded measurements following the same physical stress, but with a recording length of 10 minutes. Figure 1 shows the time course of the parameter values averaged in the sliding window containing 30 heart periods, for a 10-minute-long measurement of the healthy senior adult. Dotted lines represent the mean value of parameters measured in resting state, prior to the physical stress. Figure 1 demonstrates, that the parameters tRR and heart rate reach their resting state value within less than 50 heartbeats (less than 1 minute) while PWTT and HRV (absolute dtRR, and pNN20_50 (for healthy young adults, pNN0_20 may be more representative, than pNN20_50)) parameters reach their resting state value in between 200-300 heartbeats (approximately 3-5 minutes). This means that the PWTT and HRV parameters contain different information about the stress level of the tested person than the average heart rate alone.

### *C. The Effect of Inaccurate R-peak Detection on PWTT- and HRV-Based Stress Level Estimation*

As shown in Table I, inappropriate preprocessing in R-peak detection can lead to a mean absolute error of approximately 0.5-5 ms, depending on the filter type used. This error can also distort the time plots and averaged values of PWTT and HRV parameters, but the effect is more severe if the distribution of parameters is characterized. Figure 2 shows the histogram (upper part) and the time function (lower part) of absolute differences between the dtRR values calculated for the same measurement with two different pre-processing methods. The first method used two notch filters (center frequencies at 50 and 100 Hz) and a lowpass filter (cutoff-frequency at 50 Hz). The second method used only a notch filter (center frequency at 50 Hz). dtRR values were calculated for the last 300 heart periods of the measurement illustrated in Figure 1. For other HRV and PWTT parameters, filtering has similar effect.

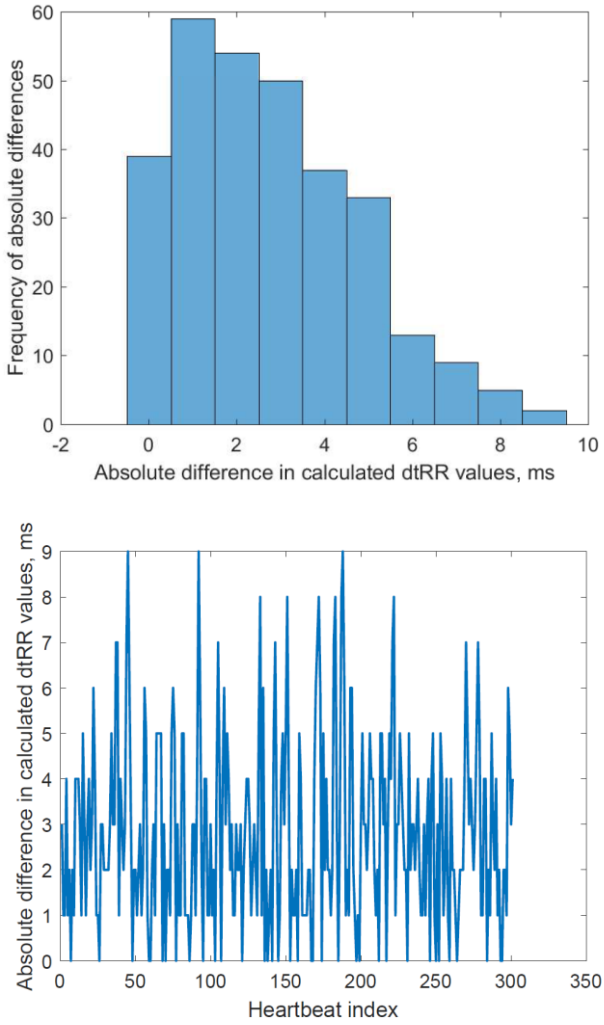



Fig. 2. Histogram (upper part) and time function (lower part) of absolute differences in the dtRR values for 300 heart periods with two different pre-processing methods. First method: two notch filters (center frequencies at 50 and 100 Hz) and a lowpass filter (cutoff-frequency at 50 Hz) were applied. Second method: a notch filter (center frequency at 50 Hz) was applied.

### *D. The Effect of Filtering of PPG on PWTT*

The effect of filtering on detection of local minima in the PPG signal was examined using the simulated noisy PPG signal and real data. Results for the simulated signal showed that decreasing the upper cutoff-frequency of the used bandpass filter below 14 Hz introduces a shift of local minima in negative direction (local minima appear earlier in time than the reference locations). Increase of the upper cutoff-frequency is limited by the level of noise. For the simulated PPG signal, upper cutoff-frequencies between 14-16 Hz yielded the best results. Manipulation of the lower cutoff-frequency caused no shift of local minima. Figure 3 shows the time function of differences in the calculated PWTT values after using two different filters on the PPG signal for a real measurement. PPG was filtered with two different bandpass filters (cutoff-frequencies at 0.5 Hz and 16 Hz as well as at 0.5 Hz and 10 Hz). The figure demonstrates that the time difference of local minima locations detected after the two different filtering is not constant, and PWTT differences may reach large values.

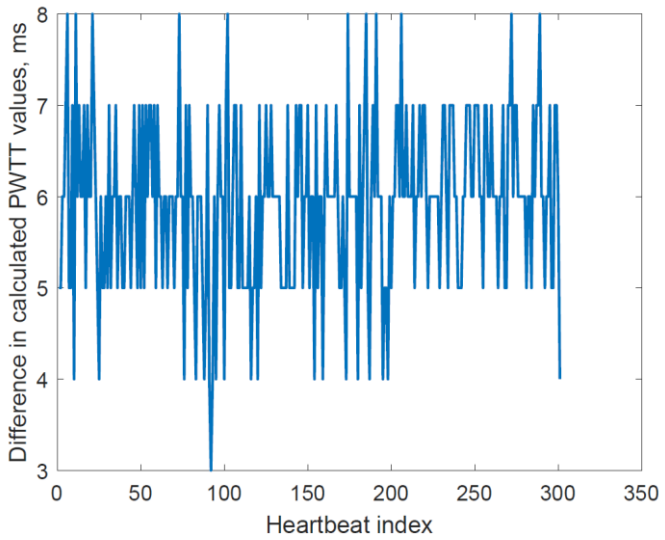



Fig. 3. Time function of differences in the calculated PWTT values after using two different filters on the PPG signal. PPG was filtered with two different bandpass filters (cutoff-frequencies at 0.5 Hz and 16 Hz as well as at 0.5 Hz and 10 Hz).

## IV. Discussion

Both simulation and real data showed that the accuracy of R-peak detection strongly depends on the filter type used in the ECG pre-processing. In our investigation, notch filtering with center frequencies at 50 and 100 Hz together with lowpass filtering with cutoff-frequency at 50 Hz provided the best results. Appropriate ECG pre-processing is especially important if the distribution of PWTT or HRV parameters is characterized (e.g. skewness or kurtosis), in those cases inaccurate R-peak detection can strongly distort the calculated values. The effect of filtering as part of the PPG signal pre-processing on the accuracy of local minima detection in the PPG signal was demonstrated, but not fully analyzed in the present study. For accurate PWTT calculation, correct detection of local minima in the PPG signal is also essential. Therefore, we aim to further investigate different steps of PPG signal pre-processing as well.

Measurement of ECG and PPG after short physical stress revealed that tRR and heart rate parameters reach their resting state value 4-6 times faster than PWTT and HRV. This means that using ECG and PPG signals, extra information can be provided about the stress level of the tested person compared to the average heart rate. However, the parameter combination characterizing low and high levels of stress is person specific. Therefore prior to stress level estimation, a set of measurements has to be recorded for each person, both in resting state and with induced stress. The results of these measurements can be used to define parameter ranges, where the stress level of the tested person is low enough so that a blood pressure measurement is suggested to be taken.

## V. Conclusion

Stress level of the examined person can have a large impact on the accuracy of blood pressure measurement. ECG and PPG sensors provide a possibility to non-invasively measure PWTT and HRV prior to blood pressure measurement. If signal processing is appropriate and ECG R-peak detection as well as PPG local minimum detection is accurate, this information can improve stress level estimation.

The present study revealed an interesting difference in the physiological regulation of heart rate, heart rate variability and arterial stiffness. However, the measurement series, needed to evaluate how this difference can be used for personalized stress level estimation has not been completed yet. For the explanation and physiological interpretation of the observed phenomena and exact implementation of the proposed parameters, we aim to organize a new measurement series with more participants.

## Acknowledgment

The research work was supported by the Croatian-Hungarian project Home Health Monitoring (TÉT_16-1-2018-0038) and by the European Regional Development Fund of the European Union (EFOP-3.6.2-16-2017-00013).

## References

[1] J. S. Floran “Blood pressure variability: a novel and important risk factor”, Canadian Journal of Cardiology, vol. 29, no. 5, 2013, pp. 557-563.

[2] A. J. Camm, et al. ”Heart rate variability: standards of measurement, physiological interpretation, and clinical use. Task Force of the European Society of Cardiology and the North American Society of Pacing and Electrophysiology”, Circulation, vol. 93, no. 5, 1996, pp. 1043–1065.

[3] S. Hey, A. Gharbi, B. von Haaren, B. Walter, K. Walter, N. König, L. Löffler “Continuous noninvasive pulse transit time measurement for psycho-physiological stress monitoring“, International Conference on eHealth, Telemedicine, and Social Medicine, 2009. eTELEMED'09. IEEE, 2009, pp. 113-116

[4] A. Szabo “The combined effects of orthostatic and mental stress on heart rate, T-wave amplitude, and pulse transit time”, European journal of applied physiology and occupational physiology, vol. 67, no. 6, 1993, pp. 540-544.

[5] J. J. Furedy, A. Szabo, F. Peronnet “Effects of psychological and physiological challenges on heart rate, T-wave amplitude, and pulse-transit time”, International journal of psychophysiology, vol. 22, no. 3, 1996, pp. 173-183.

[6] Á. Jobbágy, M. Majnár, L. K. Tóth, P. Nagy “HRV-based stress level assessment using very short recordings”, Periodica Polytechnica EECS, vol. 61, no. 3, 2017, pp. 238-245.

[7] G. M. Friesen, T. C. Jannett, M. A. Jadallah, S. L. Yates, S. R. Quint, H. T. Nagle ”A comparison of the noise sensitivity of nine QRS detection algorithms“, IEEE Transactions on Biomedical Engineering, vol. 37, no. 1, 1990, pp. 85-98.

[8] B-U Kohler, C. Hennig, R. Orglmeister “The principles of software QRS detection“, IEEE Engineering in Medicine and Biology Magazine, vol. 21, no. 1, 2002, pp. 42-57.

[9] M. Elgendi, M. Jonkman, F. DeBoer “Frequency bands effects on QRS detection“, Pan, vol. 5, 2010, pp. 15Hz.

[10] J. Pan, W. J. Tompkins “A real-time QRS detection algorithm”, IEEE Transactions on Biomedical Engineering, vol. 32, no. 3, 1985, pp. 230-236.

[11] M. Elgendi, M. Jonkman, F. DeBoer “Heart rate variability and the acceleration plethysmogram signals measured at rest“, In: International Joint Conference on Biomedical Engineering Systems and Technologies, Springer, Berlin, Heidelberg, 2010, pp. 266-277.

[12] G. D. Clifford, P. E. McSharry, “Method to filter ecgs and evaluate clinical parameter distortion using realistic ECG model parameter fitting“, Computers in Cardiology, 2005, pp. 715-718.

# Measuring quality of datasets using prediction explanation

Tamás Szántó*, Zoltán Micskei†
Budapest University of Technologies and Economics, Department of Measurement and Information Systems, Hungary
Email: *tmas.szanto@gmail.com, †zoltan.micskei@mit.bme.hu

***Abstract*—Machine Learning (ML) techniques, especially Deep Neural Networks are achieving compelling results in many fields. However, the integration of ML solutions into critical systems where trustworthiness, reliability, and safety are crucial is an unsolved problem. The datasets have great importance for Machine Learning algorithms since the achievable precision during the training, and the accuracy of the final evaluation rely both on the quality and quantity of the data. Our goal was to improve the measurement of the quality of datasets. We recommend to 1) use the requirements gathered during systems design, and 2) generate metrics, which can provide information about the representativeness of a dataset to determine which are the problems that we can safely solve by using the data. This paper describes a general method for dataset evaluation using multiple measurement techniques, including prediction explanations. The method is demonstrated by applying it to a case study of categorising road signs.**

## I. Introduction

*Machine Learning (ML) techniques* are achieving great results in many fields, usually outperforms the traditional manually created algorithms. In general, the problem with these methods is that the increase of the accuracy most of the times means a decrease in the observability of the mechanism. The ML techniques as becoming more independent from hard-coded behaviours can reach better results. These methods are extracting their knowledge from datasets during a training period and later can use this knowledge to predict the outputs of previously unexplored inputs. In one perspective this way these techniques can learn more complex associations in the data than it would be possible in a manual way. However, they can handle only situations that relate somehow to the training datasets. We can improve the achieved results by using different ML techniques, but in the end, the dataset is a hard limit of the possible precision.

*The motivation of the research* is the significant role of the training and testing datasets in ML techniques. The training data determines the overall extractable knowledge, and the testing dataset is responsible for valid precision measurement. ML methods require a trust towards the quality of the dataset, especially if they are used in critical systems (e.g., automotive, transportation). When designing and verifying such systems, assurance of their compliance with the functional and safety requirements is mandatory, that needs to be thoroughly demonstrated. The ML method is used typically in one of the system components (e.g., processing sensor data), therefore it has to comply to the same requirements too. Determining the representativeness of the dataset for the entire task is a complex problem. However, most of the datasets provide only essential diversity metrics like the number of elements or the precision of the labelling. These parameters are relevant but insufficiently detailed to establish a trust for in critical systems.

*The goal of our research* is to improve the measurements of the quality of the datasets. The primary purpose of a dataset is to represent a problem, a domain. The quality of the data means how accurate is this representation. Our idea is to provide a more detailed evaluation process based on 1) the given requirements of the problem, and 2) metrics obtained from various sources. We want to determine the use cases where it is safe to use the dataset and to find the missing, under and over represented scenarios.

*We propose a method* for improving the measurements of the quality of datasets. Our evaluation is based on the requirements representing the problem domain captured during traditional systems design. The evaluation method has three main parts: a general diversity, a static and dynamic analysis. The method contains metrics like standard numeric data about the structure of the dataset, summarised diversity metrics based on generic processing methods and using prediction explanations for more detailed analysis of the elements. The general method does not contain any domain-specific assumptions, but the metrics have to be mapped to the actual domain.

*Gaining trust in ML techniques* is hard but necessary to use them in real-world applications, especially in safety-critical systems [1]. The first step is to have metrics about the quality of the training and validation datasets. This paper summarises multiple existing quality measurement approaches (Sect. II) and proposes a general method for a more detailed dataset evaluation (Sect. III). We also demonstrate our method on a case study, highlighting how it can uncover problems in the quality of datasets (Sect. IV).

## II. Related work

There are simple, essential numeric metrics about a dataset like the number of its elements, classes and size. However, richer ways were proposed to describe the collected data.

The ImageNet paper [2] describes an averaging method for evaluating the diversity. The idea is to stack all of the images from a selected class on top of each other and to generate a new image by calculating the mean value for every pixel. In theory, the created images should be completely grey, since ideally each of the pixels contained all of the colours. [2] We can identify some of the over-represented parts, and also can

compare the same classes from different datasets. Exporting the generated images as JPGs, we can associate the grey areas with the size of the files. The numeric representation of the metric is vital during evaluating large datasets.

It is difficult to determine the representativeness of a dataset alone accurately, but comparing multiple ones from the same domain can reveal some of the defects. Torralba and Efros [3] observed the biases in several datasets for general image classification. The main troubling idea is if we play a guessing game, where we randomly select an image from a dataset and trying to pick its source we can achieve too good results. The images are representing the same domain (the selected class) and collected similarly from the internet. However, the datasets have to have strong biases that allow us to identify them. Part of the cases we can trace back the source of these to the content of the images, like the age and environment of the cars. The authors also used a cross-evaluation for identifying the impact of these biases on the performance. The process using the subject dataset of the evaluation for the training and then running the tests on images from different datasets too, almost every time the results are the drop of the accuracy of the model on the other dataset compared to the original tests.

Prediction explanation [4] tools provide a glance inside the working mechanism of a black box model. They can calculate the relevance of the different parts of the input with one specific prediction. We can use the generated explanations also to examine the data, determine which features are well-, over- or under-represented. One of the most general approaches for prediction explanation is LIME [5]. It can interpret a prediction of a wide variety of ML techniques. It builds a linear model to approximate the examined method locally. The generated evaluations are detailed and precise, but generating them for a prediction is a computation-heavy, time-consuming task. Another method is Grad-CAM [6], which can interpret any CNN deep neural network. It has a more limited applicable field than LIME, but it is still quite extensive. It based on the sensitivity analysis of the layer, that represents the extracted features the most (for example the last convolutional layer of a convolutional network [6]). It can calculate de relevance of the input parameters based on the gradient values of the neurons. Its output explanation also less detailed than LIME's, but its main feature is that it does not have significant computational requirements, and gives very little overhead to the entire prediction process.

## III. Recommended method

Fig. 1 depicts how our approach combines systems engineering and ML engineering to improve the evaluation of the dataset's quality. The *Domain* is the collection of the scenarios that are part of the solvable problem. The possible inputs that can appear in an actual use-case are part of it. Even with a not too complex problem, we can get an enormous domain quite easily. It is not possible to use or collect the entire domain. Instead, a *Dataset* can represent the possible elements. Finding the right dataset for the actual problem is a difficult task, we can choose an existing one (or several ones) or can build it from the ground. Regardless of which approach we use, the selected dataset must represent the domain well. To be able to evaluate this fact we must measure several parameters of the dataset. Usually, most of the dataset comes (or we can calculate it easily) with some *General diversity metrics*.

To improve the evaluation process, our first step starts from the *Requirements* representing the domain. In the *Static evaluation* step, these requirements are compared to the dataset. This step only uses the textual or graphical requirements and the plain dataset with some helper analysis (like well-defined image processing techniques). The *Dynamic evaluation* uses knowledge about the actual content of the elements in the dataset. We can estimate this knowledge with using the dataset in a simple *Model* with a *Prediction explanation* tool.

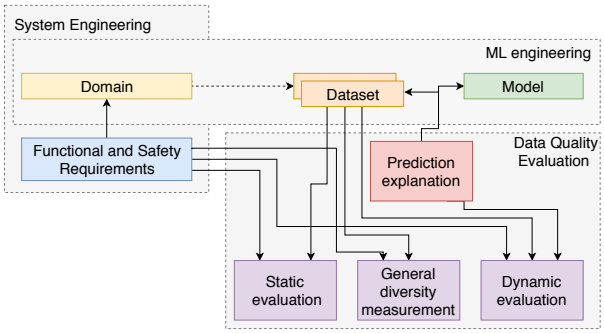



Fig. 1. Overview of the recommended method

*General diversity measurements:* As mentioned above these metrics usually come with the datasets, they are giving a minimal summary of the collected data. Typical examples are the number of elements, classes, sizes of the classes, distribution of the data points. Optionally they can reveal the source and method of the data collection, the used labelling technique. The purpose of these metrics is to provide a basic idea about the usability and quality of the dataset. In some cases, we can eliminate dataset candidates based on these numbers, but for final selection, we need more detailed evaluations [7].

*Static evaluation:* The functional and safety requirements provide a proper description of the scope of the problem. These requirements can have multiple formats and levels. Typical representations include textual (pattern-based), model-based (UML or SysML) and formal language based solutions [8]. The requirements can identify the relevant scenarios and also can determine if an element is part of or not part of the task. A dataset is a representation of the domain but giving an accuracy metric for the representation is a challenging problem. As a metric for this, we can evaluate the defined requirements on the dataset to verify its representativeness. The analysis method depends on the nature of the requirements and data. In general, in this step, we should find algorithmic methods that can generate features for the elements and the dataset which are usable for the evaluation of the requirements. For example, for analysing image data, we usually can use aggregated metrics based on the pixel values or some edge detection algorithms.

*Dynamic evaluation:* In this step, we run an example model to extract further parameters of the dataset with the help of a prediction explanation tool [4]. This tool can identify the relevant and irrelevant parts of the input data, which can be used for further analysis. These can help the evaluation of the requirements by the categorisation of the input features by relevance. The generated importance maps also can define relevance patterns inside the dataset. For example, if an input parameter is never used for the prediction, it can indicate that it is not relevant for the problem or not represented well, i.e., there are not enough example for the possible scenarios.

The dynamic evaluation can contain cross-evaluation between multiple datasets. Since the datasets represent the same domain, we expect that we should get similar results with the tests, independently from the datasets that are used for the training. Thus we can train multiple models (but with the same architecture) on multiple datasets and after that, and we can evaluate the trained models on each other's test data [3].

## IV. Case study

We created a case study to demonstrate the working mechanism of the method and to show an example solution that can work with almost any image classification problem.

*Solvable problem:* The task is to categorise the road signs in Belgium[1] and Germany[2]. Although there are several minor differences in the legislation of the countries, in this case study we assume they are the same domain (we can only use the road signs which are the same in the two countries from the datasets for the evaluation methods that requires comparisons). The following requirements can define the domain and the tasks of a hypothetical system that can use ML techniques (the requirements of such existing systems are not public).

- REQ1 The system shall identify the road sign in Belgium or Germany. The classification shall handle all of the possible sign types. However, finding the image parts which are containing a sign is a task for another system in the pipeline, this system only responsible for the categorisation of the signs.
- REQ2 The system shall handle images with different lighting conditions, such as dark, half-light, daylight, harsh sunlight, shadows.
- REQ3 The system shall perform the classification independently from the environment of the sign, such as buildings, countryside, trees, sky, street.
- REQ4 The system shall identify the traffic signs from different angles, and image quality too.
- REQ5 The system shall identify the traffic signs even when they are partly covered.

Note that our goal is to demonstrate the recommended method, but a real scenario would require more detailed refinements of these high-level requirements.

*General diversity measurements:* The German dataset is almost ten times larger than the Belgian and has a more reasonable size. Some of the classes in the Belgian dataset only contains around twenty images which are not enough data in most of the cases.

| Dataset | Classes | Training | Tests |
|---|---|---|---|
| Belgian | 62 | 4575 | 2520 |
| German | 43 | 39209 | 12569 |

TABLE I
General diversity of the datasets

As a general diversity metric, we can evaluate the averaging image technique [2]. In the following figures, the leftmost image is generated by stacking and averaging all of the images in the given class. The stacked image should have a greyish background if the class has enough diversity since this colour is the result of mixing all the others.

Fig. 2. Stop and Cattle crossing sign stacked and example images (Belgian)

For example, the stacked image of the cattle crossing sign has a green background (Fig. 2), which indicates that most of the images were taken in a green environment. The problem is that these signs not necessary has a green background (winter or blue sky). However, with these images, it is a possibility that the ML technique learns the background too.

*Static evaluation:* REQ1: The datasets only contain images of road signs, they can be used to categorise signs, but not to detect them. The first problem with both of the datasets is that they do not contain all of the road sign categories. There are around 150 road sign types [9], but the Belgian dataset contains 63, the missing elements mostly highway, roadwork and train related signs. The German dataset is less complete, there are 43 classes, and 8 of them are just different speed limits; crosswalk or parking signs are entirely missing.

REQ2: We can estimate the lighting conditions from the overall brightness of the pictures which can be determined by calculating the mean pixel value on the images. The diversity of the lighting conditions is eligible in both of the datasets.

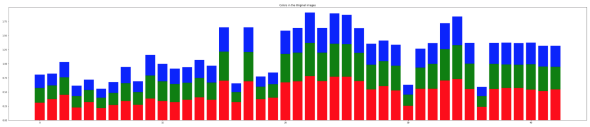


Fig. 3. RGB values in the STOP sign category in the Belgian dataset

REQ3: We can examine the diversity of the images with calculating mean RGB values for the images (Fig.3). Mostly the reds are dominating the pictures, but this is natural because of the red parts of the signs. The differences between the

[1]Belgium Traffic Sign Dataset: https://btsd.ethz.ch/shareddata/
[2]German Traffic Sign Recognition Benchmark: http://benchmark.ini.rub.de/

distribution of the RGB values can indicate the different environments on the images.

REQ4 and REQ5: The stacked images almost clearly show the signs in the centre of the images, which indicates that there aren't too many images that contain only parts of the signs or partly covered ones. Also, we want to see a grey colour in the background, but some of the classes have a coloured background (mainly those that contain fewer examples).

*Dynamic evaluation:* This evaluation requires an actual running model on the dataset. Our choice is a simple convolutional deep neural network [10]. It contains three convolutional layers for the feature extraction and another three fully connected layers for the classification. The input size of the network is 32 by 32 as the rescaled images in the datasets.

We trained two models with the same (above described) architecture on the training part of the two datasets. Both of the training was only 80 epochs, at this point we have managed to get decent enough results for the evaluation. First, we run the inference with the datasets' test data and then used the other's tests.

| Train / Test | Belgian | German | Bg. STOP | Gm. STOP |
|---|---|---|---|---|
| Belgian | 68.3% | 26.8% | 35.6% | 12.5% |
| German | 78.4% | 94.2% | 84.4% | 95.4% |

TABLE II
THE RESULTS OF THE CROSS-EVALUATIONS OF THE MODELS

In general, the model with the Belgian dataset has weaker performance results, and its accuracy dropped more with the tests from the other dataset than the one trained with the German data. However, the German model loses 6% in accuracy too, which is significant if we calculate that the training contained more than ten times more images than the Belgian dataset (for example there are 50 STOP signs in the Belgian and 800 in the German dataset). The first part of the table contains the results of the evaluation of the entire test sets of the datasets and second half is an example for the STOP sign category. With ideal conditions, the differences should be far less between a class and the entire results, but we used a simple model with a short training on unbalanced datasets.

Grad-CAM can provide prediction explanation for any convolutional network so that it can interpret our model's predictions too. It generates the relevance maps for the inputs, to help the further evaluation processes, there is an image with the raw greyscaled mask and another with the coloured heatmap on the top of the input image.

The prediction explanation can help the evaluation by highlighting the main parts of the images. These parts are relevant for the predictions we want them to be diverse enough.

The grey-scaled images are the raw relevance masks, which are more usable with algorithms to recognise patterns (for example selecting the images with less white areas from the entire dataset). The coloured heatmaps on top of the original images are more usable for manual evaluation. The whites and reds are the most relevant areas. As the first example shows too, most of the signs are clear and in the centre. The second example is more complicated (it shows a partly covered sign) than the more general first. These scenarios are harder to handle, it is essential to have enough data from these. In both of the datasets, the scenarios with the partly covered sign are under-represented. The generated masks also can help the static evaluation for selecting the subject on the images.


Fig. 4. Example for the dynamic evaluation (Belgian)

*Summary:* the evaluation process generated valuable, usable metrics, managed to cover all of the requirements at least by one measurement. However, we discovered some severe deficiencies with both of the datasets, which eliminates the possible use of them for the defined recognition task. Also, our method identified several possible improvements of the datasets to be complete and usable.

## V. Conclusion

We managed to generate a meaningful evaluation for the case study based on the proposed general method. The requirement analysis part of the process and selecting the relevant evaluation methods for checking the rules with almost every system needs some level of manual work. However, we can automate most of the evaluations independently from any domain. Overall our approach even if it requires further specifications of the evaluation methods can be used efficiently for gaining more trust towards ML techniques applicability in critical systems.

*Acknowledgement* The research reported in this paper was supported by the BME Artificial Intelligence FIKP grant of EMMI (BME FIKP-MI/SC)

## References

[1] Waymo LLC., "Waymo safety report – on the road to fully self-driving," 2017, https://storage.googleapis.com/sdc-prod/v1/safety-report/waymo-safety-report-2017.pdf.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition*. IEEE, 2009, pp. 248–255.

[3] A. Torralba and A. A. Efros, "Unbiased look at dataset bias," in *Computer Vision and Pattern Recognition*. IEEE, 2011, pp. 1521–1528.

[4] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems*, 2017, pp. 4765–4774.

[5] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should I trust you?: Explaining the predictions of any classifier," in *Int. Conf. on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 1135–1144.

[6] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual explanations from deep networks via gradient-based localization," in *ICCV*, 2017, pp. 618–626.

[7] M. A. Mazurowski, P. A. Habas, J. M. Zurada, J. Y. Lo, J. A. Baker, and G. D. Tourassi, "Training neural network classifiers for medical decision making: The effects of imbalanced datasets on classification performance," *Neural networks*, vol. 21, no. 2-3, pp. 427–436, 2008.

[8] K. Pohl, *Requirements Engineering: Fundamentals, Principles, and Techniques*, 1st ed. Springer Publishing Company, Incorporated, 2010.

[9] UNECE, "Vienna convention on road signs and signals," 2007, https://www.unece.org/fileadmin/DAM/trans/conventn/Conv_road_signs_2006v_EN.pdf.

[10] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

# Analysis of Distributed Multi-Channel Active Noise Cancelling Algorithms

Balázs Varga, György Orosz
Budapest University of Technology and Economics
Department of Measurement and Information Systems
Budapest, Hungary
Email: bvarga92@gmail.com, orosz@mit.bme.hu

***Abstract*—Multi-channel active noise cancellation is typically achieved by using computationally expensive signal processing algorithms. A centralized architecture, in which all computation is carried out by a single processing unit, is therefore often costly and poorly scalable. Noise cancelling systems designed in a distributed fashion can be more efficient. In this article, the authors propose and compare a number of different implementation schemes of distributed active noise cancellation, with emphasis on computational complexity and settling time.**

***Index Terms*—active noise cancellation, FxLMS, distributed signal processing**

## I. Introduction

Noise control refers to a means of reducing acoustic emissions in order to improve personal comfort, comply with legal requirements or to reduce environmental noise pollution. Noise control methods can be divided into two major categories: passive and active. Conventional *passive noise control* measures use physical barriers and isolating materials such as soundproofing insulation and sound-absorbing wall panels. However, these methods lack flexibility and they generally do not work well at low frequencies, where the acoustic wavelengths become large compared to the thickness of a typical acoustic absorber [1].

In *active noise control* (also commonly referred to as active noise cancellation, ANC), noise suppression is achieved by using a speaker to generate an anti-noise, which causes destructive interference in the desired protection zone – as illustrated in Fig. 1. As opposed to passive methods, active noise cancelling offers higher flexibility and portability, as well as better low-frequency performance. However, they are not without drawbacks either. Active methods require the constant availability of a power source, and the zone of noise suppression is smaller – especially at higher frequencies, as it is inherently comparable in size to the wavelength of the sound. Furthermore, an active noise cancellation system may create areas outside the protection zone where the noise is not cancelled but amplified. Important practical applications of ANC include enhancing vehicle comfort (e.g. suppression of engine, propeller or rotor noise in car interiors and aircraft cabins), noise-cancelling headphones, electronic stethoscopes and sleep aid devices [2].

Due to their relatively small size, usually multiple protection zones are necessary in practical applications (e.g. multiple seats of a car, two ears of a person). As we will see in Section II, this requirement can greatly increase the computational complexity of digital active noise cancellation algorithms. Therefore, ANC systems implemented in a centralized architecture – i.e. a single processing unit is used to carry out all of the necessary computation – often use expensive high-performance DSP (digital signal processor) or FPGA (field-programmable gate array) circuits. However, performance limits are still easily reached with the addition of more protection zones.

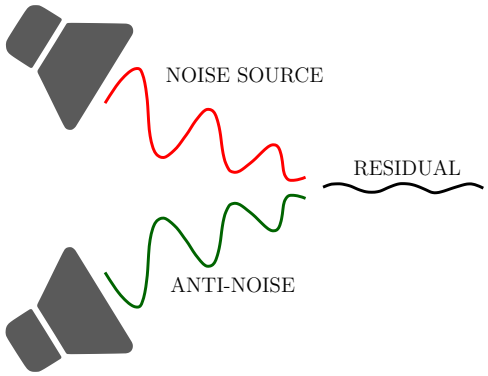



Fig. 1. Active noise cancellation.

Cost and scalability can be improved by distributing the computational load among multiple processing units which are interconnected via a common communication network. As an additional benefit, a distributed architecture creates the potential for improved fault tolerance. However, decomposing the algorithm into separately executable sections is not a trivial problem, and several different approaches exist. Therefore, when designing a distributed active noise cancelling system, factors such as the available computational performance, network bandwidth, and the achievable settling time must be taken into careful consideration.

This paper is structured as follows. Section II describes an algorithm widely used in digital active noise cancelling systems, as well as three implementation architectures with varying degrees of centralization. In Section III, results of numerical simulations are presented for each of these architectures, allowing for comparison. Finally, Section IV concludes the paper.

## II. Algorithms and Architectures

### A. The FxLMS Algorithm

One of the most frequently used algorithms in active noise cancelling is the Filtered-$x$ Least Mean Squares (FxLMS), proposed by Widrow et al. in 1981 [3]. The algorithm is illustrated in Fig. 2 for the case of single-channel noise cancellation.

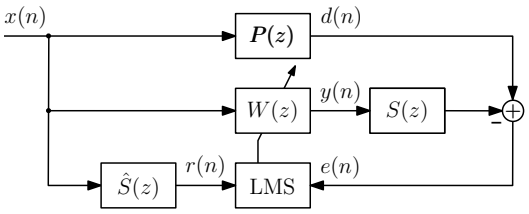



Fig. 2. Diagram of the FxLMS algorithm.

The *reference signal* $x(n)$ is measured at the noise source, and propagates to the point of suppression through the *primary acoustic path* modeled by the discrete-time transfer function $P(z)$. The resulting *disturbance signal* $d(n)$ is the noise we aim to suppress. The speaker outputs the *anti-noise* $y(n)$ which propagates through the *secondary acoustic path* $S(z)$. The two signals get added[1] together at the point of suppression, and the resulting *error signal* $e(n)$ is picked up by the microphone. The reference signal is supplied to the algorithm and is filtered with the transfer function $\hat{S}(z)$. Ideally, $\hat{S}(z) = S(z)$, however, the transfer function of the secondary path is usually not known analytically. Therefore, $\hat{S}(z)$ is the result of a system identification performed prior to starting the normal noise-cancelling operation. The transfer function $W(z)$ is a finite impulse response (FIR) filter of order $L-1$ that is initialized to zero and is updated in each step according to the LMS rule:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + 2\mu e(n)\mathbf{r}(n) \tag{1}$$

where $\mathbf{w}(n)$ is a vector of the filter coefficients, $\mathbf{r}(n)$ is a vector containing the previous $L$ samples of the *filtered reference signal* $r(n)$, and $\mu$ is the *step size* parameter which influences stability and settling time.

If the algorithm has achieved convergence, $W(z) \approx \frac{P(z)}{S(z)}$, resulting in $e(n) \approx 0$, i.e. the disturbance is being actively cancelled.

### B. Completely Centralized Architecture

For the following sections, we restrict our analysis to the special case of multiple channel noise cancellation, where the number of microphones (protection zones) is equal to the number of speakers – let this number be $N$. In this case, $P_m(z)$ is the primary path from the noise source to the $m$th microphone, $S_{s,m}(z)$ is the secondary path from the $s$th speaker to the $m$th microphone, and $r_{s,m}(n)$ is the reference filtered with $\hat{S}_{s,m}(z)$, where $s, m = 1 \ldots N$. Each microphone has its own error signal $e_m(n)$, and each speaker has its own filter $W_s(z)$ and output $y_s(n)$. The adaptation rule now becomes:

$$\mathbf{w}_s(n+1) = \mathbf{w}_s(n) + 2\mu \sum_{m=1}^{N} e_m(n)\mathbf{r}_{s,m}(n) \tag{2}$$

[1]For historical reasons, the error signal is written as the *difference* between the disturbance and the anti-noise. This is merely a sign convention; in a practical application the computed anti-noise is multiplied by $-1$.

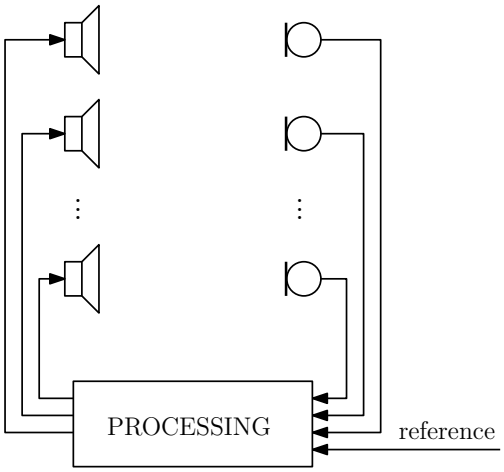



Fig. 3. Completely centralized architecture.

Fig. 3 shows an architecture in which a single processing unit is responsible for sampling the microphones, driving the speakers, as well as executing the FxLMS algorithm for all channels, which requires $N(N+1)$ FIR filtering and $N^2$ vector addition operations.

### C. Partially Distributed Architecture

An example of a partially distributed architecture is shown in Fig. 4. In this case the majority of the computation is still carried out by a high-performance central processing unit, however, the sampling of the error signals is done by individual sensor nodes (also known as *motes*), which are connected to the central unit via a communication network

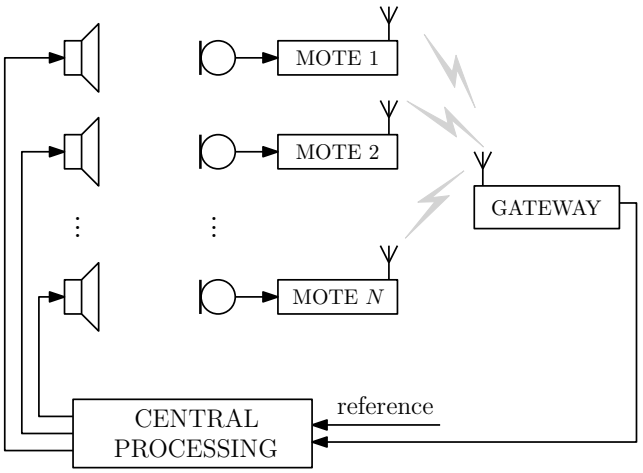



Fig. 4. Architecture with distributed data acquisition.

such as ZigBee or Ethernet. Since the motes are already equipped with a simple processing unit (typically a low-power microcontroller), some rudimentary preprocessing may be done by the motes themselves (e.g. data compression).

In the typical operation of such an architecture, the motes buffer their error signals and periodically send their buffer contents to the central unit. Mathematically, this can be modeled by introducing a delay in the secondary paths:

$$S'_{s,m}(z) = S_{s,m}(z)z^{-\Delta} \tag{3}$$

where $\Delta$ is the send period (expressed in the number of samples). Assuming simultaneous transmission, the required communication bandwidth is $Bf_sN$, where $B$ denotes the number of bits used to represent a signal sample, and $f_s$ is the sampling frequency.

### D. Completely Distributed Architecture

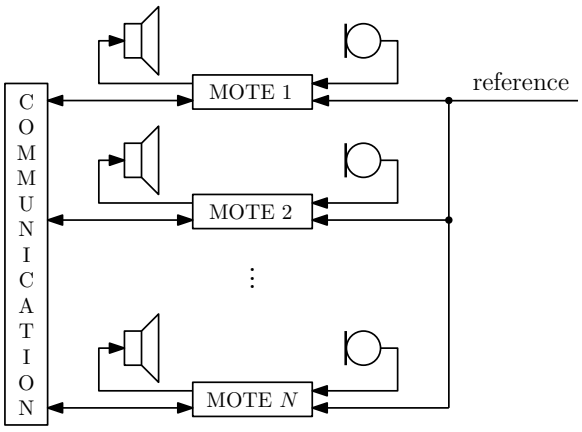



Fig. 5. Completely distributed architecture.

The architecture shown in Fig. 5 lacks a designated central processing unit; the computations of the FxLMS algorithm are carried out entirely by the motes in an evenly distributed fashion. The operations executed by the $k$th mote can be summarized as follows:

---

**In every step:**

For all $j = 1 \dots N$:

$r_{j,k}(n) \leftarrow \mathbf{x}^{\mathsf{T}}(n)\hat{\mathbf{s}}_{j,k}$

$\delta\mathbf{w}_{j,k} \leftarrow \delta\mathbf{w}_{j,k} + 2\mu e_k(n)\mathbf{r}_{j,k}(n)$

$\mathbf{w}_k \leftarrow \mathbf{w}_k + \delta\mathbf{w}_{k,k}$

$\delta\mathbf{w}_{k,k} \leftarrow \mathbf{0}$

$y_k(n) \leftarrow \mathbf{x}^{\mathsf{T}}(n)\mathbf{w}_k$

---

**If it is time to send $\delta\mathbf{w}_{j,k}$ to mote $j$:**

$\text{Send}_j(\delta\mathbf{w}_{j,k})$

$\delta\mathbf{w}_{j,k} \leftarrow \mathbf{0}$

---

**If $\delta\mathbf{w}_{k,j}$ was received from mote $j$:**

$\mathbf{w}_k \leftarrow \mathbf{w}_k + \delta\mathbf{w}_{k,j}$

---

In this architecture, each mote accumulates the filter updates for every other mote for a preset number of steps, after which the updates are sent to the other motes over the communication network. This operation is similar to that described in [4] and [5] – however, the authors proposed a frequency-domain implementation, which only allows blockwise processing. Contrarily, a time-domain implementation allows every mote to apply an update based on its own error signal in each step, which may lead to faster convergence. Furthermore, the algorithm described above makes it possible to tune transmission periods individually, allowing the available communication bandwidth to be distributed among the motes arbitrarily.

Since each mote needs to carry out $N + 1$ FIR filtering operations and $N+1$ vector additions in the majority of steps, the per-mote computational complexity scales linearly with the number of nodes.

## III. Simulation Results

Each of the active noise cancellation architectures described in Section II was implemented in MATLAB for two channels. Simulations were run with multiple physical configurations; the results presented in this section were obtained under the following common circumstances:

- The speakers and the microphones were located in the vertices of a square, except for the microphone of channel 2, which was moved significantly farther.
- The acoustic paths were chosen as simple allpass filters with delay and attenuation corresponding to the geometry.
- 200 Hz sinusoidal signal was used as reference.
- The sampling frequency was 8 kHz.

In each simulation, the step size was tuned to obtain the fastest possible settling time. The error signal was considered settled when its RMS (root mean square) decreased below 10% of its initial value.

### A. Comparison of the Architectures

The first experiment was carried out in order to compare the fastest attainable settling time with the three previously described ANC architectures, under identical circumstances. The error signals obtained in the three simulations are shown in Figures 6-8, in decreasing degree of centralization.

As anticipated, the fastest settling was achieved with the fully centralized system, where every filter update is applied as soon as it becomes available.

The partially distributed architecture provided significantly worse results, with an average settling time nearly twice longer than in the previous case. This is not surprising, considering the delay introduced by the buffering nature of this system.

The fully distributed system provided results comparable to the centralized case, the average settling time being only 8% longer. This superior behavior presumably stems from two characteristics of this distributed algorithm. First, all filters are updated in every step based on the locally available error signal. Secondly, all filter updates are still *calculated* in every step – it is only the application that is delayed.

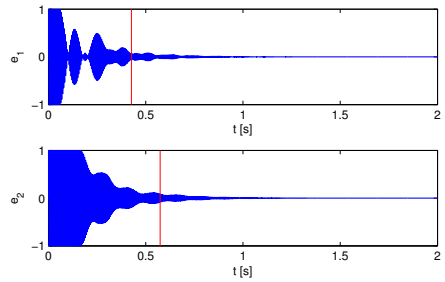



Fig. 6. Settling – centralized (427 ms and 574 ms)

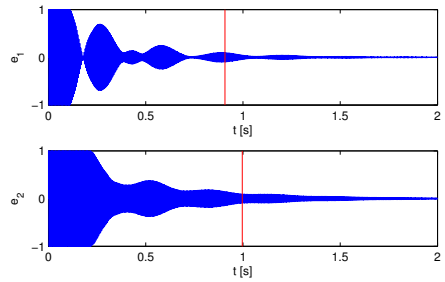



Fig. 7. Settling – distributed acquisition (908 ms and 997 ms)

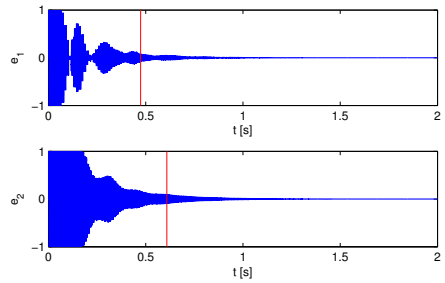



Fig. 8. Settling – distributed (474 ms and 609 ms)

### B. Effect of Transmission Period

In the next experiments, the behavior of the fully distributed ANC system was further investigated.

First, multiple simulations were run with different transmission periods (equal in the two motes), under otherwise identical conditions. As shown in Fig. 9, this had practically no effect on the settling time until the transmission time became comparable to the settling time itself. This finding is consistent with the analytical results obtained in [6].

### C. Effect of Bandwidth Distribution

In our final experiment, the distribution of the available network bandwidth among the motes was variable. Fig. 10 shows that this also had very little effect on the settling time; the slower channel could not be made to settle any faster by varying the bandwidth distribution. (A distribution of 0 corresponds to the case when all network resources are allocated to mote 2, and mote 1 is unable to transmit – and vice versa.)

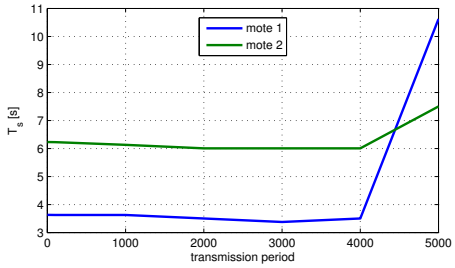



Fig. 9. Settling time vs. transmission period

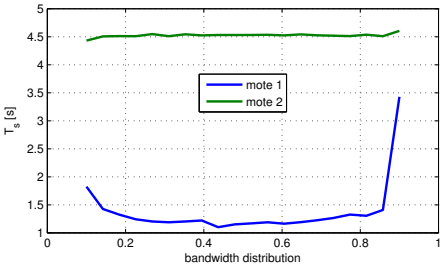



Fig. 10. Settling time vs. bandwidth distribution

## IV. Conclusion

In this paper, a time-domain implementation of a distributed active noise cancelling algorithm was proposed and its properties were compared to other similar methods. Its performance was found to be superior to a simpler, more centralized ANC architecture. Future research should focus on the analytical derivation of settling parameters, investigation of the algorithms for the case of more than two channels, as well as on formulating feasible design guidelines.

## Acknowledgement

The research reported in this paper was supported by the Higher Education Excellence Program of the Ministry of Human Capacities in the frame of Artificial Intelligence research area of Budapest University of Technology and Economics (BME FIKP-MI/SC).

## References

[1] S. J. Elliot, P. A. Nelson, “Active noise control,“ IEEE Signal Processing Magazine, 10(4):12–35, October 1993.

[2] D. Miljković, “Active Noise Control: From Analog to Digital – Last 80 Years,“, 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 1358–1363, June 2016.

[3] B. Widrow, D. Shur, S. Shaffer, “On adaptive inverse control,“ Proceeding of the 15th Asilomar Conference on Circuits, Systems and Computers, pp. 185–189, November 1981.

[4] C. Antoñanzas, M. Ferrer, M. de Diego, A. Gonzalez, “Blockwise Frequency Domain Active Noise Controller Over Distributed Networks,“ Applied Sciences, 6 (5), 124, April 2016.

[5] J. Lorente, C. Antoñanzas, M. Ferrer, A. Gonzalez, “Block-based distributed adaptive filter for active noiose control in a collaborative network,“ 23rd European Signal Processing Conference (EUSIPCO), pp. 310–314, Nice, France, 2015.

[6] G. A. Clark, S. K. Mitra, S. R. Parker, “Block Implementation of Adaptive Digital Filters,“ IEEE Transactions on Circuits and Systems, vol. CAS-28, no. 6, pp. 584–592, June 1981.