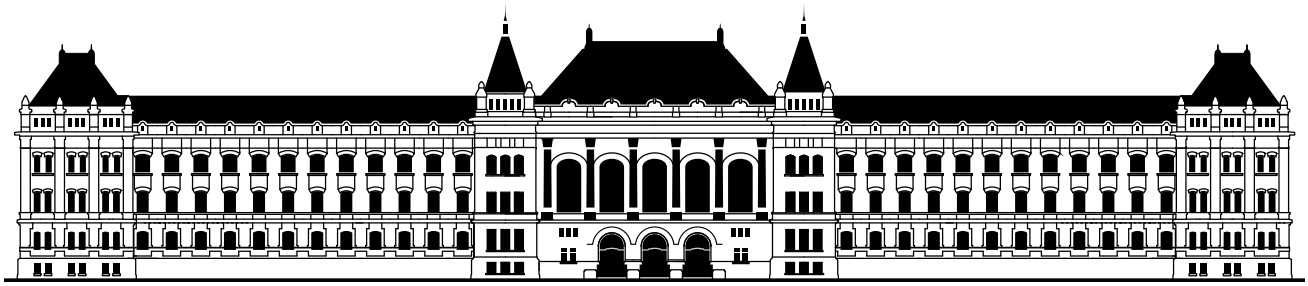
PROCEEDINGS
OF THE
24TH PHD MINI-SYMPOSIUM
(MINISY@DMIS 2017)

JANUARY 30–31, 2017
BUDAPEST UNIVERSITY OF TECHNOLOGY AND ECONOMICS
BUILDING I



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

Head of the Department: Dr. Tamás Dabóczi

Conference chairman:
Béla Pataki

Organizers:
Csaba Debreceni
András Szabolcs Nagy
András Pataki
Tamás Tóth

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

Sponsored by:

Schnell László Foundation

ISBN 978-963-313-243-2

FOREWORD

This proceedings is a collection of the lectures of the 24th Minisymposium held at the Department of Measurement and Information Systems of the Budapest University of Technology and Economics. In the previous years the main purpose of these symposiums was to give an opportunity to the PhD students of our department to present a summary of their work done in the preceding year. It is an interesting additional benefit, that the students get some experience: how to organize such events. Beyond this actual goal, it turned out that the proceedings of our symposiums give an interesting overview of the research and PhD education carried out in our department. Last year the scope of the Minisymposium had been widened; foreign partners and some of the best MSc students were also involved. This was a real benefit, therefore, this year we have kept this widened scope.

The lectures reflect partly the scientific fields and work of the students, but we think that an insight into the research and development activity of our and partner departments is also given by these contributions. Traditionally our activity was focused on measurement and instrumentation. The area has slowly changed, widened during the last few years. New areas mainly connected to embedded information systems, new aspects e.g. dependability and security are now in our scope of interest as well. Both theoretical and practical aspects are dealt with.

The lectures are at different levels: some of them present the very first results of a research, others contain more new results. Some of the first year PhD students have been working on their fields only for half a year. Therefore, there are two types of papers. One is a short independent publication; it is published in the proceedings. The other is simply a summary of the PhD student's work. This second one is intended to give an overview of the research done during the last year; therefore, it could contain shorter or longer parts of the PhD student's other publications. It does not necessarily contain new results, which have not been published earlier. It is clearly indicated in each paper that which category it belongs to. To avoid copyright conflicts, these papers are not published in the proceedings. Anyone interested, please contact the author.

During this twenty-three-year period there have been shorter or longer cooperation between our department and some universities, research institutes, organizations and firms. Some PhD research works gained a lot from these connections. In the last year the cooperation was especially fruitful with the European Organization for Nuclear Research (CERN), Geneva, Switzerland; Vrije Universiteit Brussel Dienst ELEC, Brussels, Belgium; Robert Bosch GmbH., Stuttgart, Germany; Department of Engineering, Università degli Studi di Perugia, Italy; National Instruments Hungary Kft., Budapest; University of Innsbruck, Austria; University of Geneva, Italy; University of Florence, Italy.

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

Budapest, January, 2017

Béla Pataki
Chairman of the PhD
Mini-Symposium

LIST OF PARTICIPANTS

Participant	Supervisor	Programme
Búr, Márton	Varró, Dániel	PhD
Darvas, Dániel	Majzik, István	PhD
Debreceni, Csaba	Varró, Dániel	PhD
Farkas, Rebeka	Hajdu, Ákos	MSc
Ferencz, Bálint	Kovácsházy, Tamás	PhD
Graics, Bence	Molnár, Vince	BSc
Gujgiczer, Anna and Elekes, Márton	Vörös, András	BSc
Hadházi, Dániel	Horváth, Gábor	PhD
Hajdu, Ákos	Micskei, Zoltán	PhD
Hajdu, Csaba	Dabóczi, Tamás	PhD
Häusler, Martin	Breu, Ruth	PhD
Honfi, Dávid	Micskei, Zoltán	PhD
Klenik Attila	Pataricza András	MSc
Klikovits, Stefan	Gonzalez-Berges, Manuel	PhD
Molnár, Vince	Majzik, István	PhD
Nagy, András Szabolcs	Varró, Dániel	PhD
Palkó, András	Sujbert, László	BSc
Sallai, Gyula	Tóth, Tamás	BSc
Schiavone, Enrico	Bondavalli, Andrea	PhD
Schoukens, Maarten	Rolain, Yves	Post PhD
Semeráth, Oszkár	Varró, Dániel	PhD
Szárnyas, Gábor	Varró, Dániel	PhD
Szilágyi, Gábor	Vörös, András	MSc
Tóth, Tamás	Majzik, István	PhD
Virosztek, Tamás	Kollár, István	PhD
Zoppi, Tommaso	Bondavalli, Andrea	PhD

PAPERS OF THE MINI-SYMPOSIUM

Author	Title	Page
Búr, Márton	Towards Modeling Cyber-Physical Systems From Multipe Viewpoints	6
Darvas, Dániel	Well-Formedness and Invariant Checking of PLCspecif Specifications	10
Debreceni, Csaba	Approaches to Identify Object Correspondences Between Source Models and Their View Models	14
Farkas, Rebeka	Activity-Based Abstraction Refinement for Timed Systems	18
Ferencz, Bálint	Effects of Memory Errors in IEEE 1588 Clock Networks	**
Graics, Bence	Formal Compositional Semantics for Yakindu Statecharts	22
Gujgiczer, Anna and Elekes, Márton	Towards Model-Based Support for Regression Testing	26
Hadházi, Dániel	Spectral Leakage in Matrix Inversion Tomosynthesis	30
Hajdu, Ákos	Exploratory Analysis of the Performance of a Configurable CEGAR Framework	34
Hajdu, Csaba	Possibilities for Implementing a Signal Model Within a Fourier Analyzer	38
Häusler, Martin	Sustainable Management of Versioned Data	42
Honfi, Dávid	User-defined Sandbox Behavior for Dynamic Symbolic Execution	46
Klenik, Attila	Performance Benchmarking Using Software-in-the-Loop	**
Klikovits, Stefan	Towards Language Independent Dynamic Symbolic Execution	50
Molnár, Vince	Constraint Programming with Multi-valued Decision Diagrams: A Saturation Approach	54
Nagy, András Szabolcs	Effects of Graph Transformation Rules to Design Space Exploration Problems	58
Palkó, András	Enhanced Spectral Estimation Using FFT in Case of Data Loss	62
Sallai, Gyula	Boosting Software Verification with Compiler Optimizations	66
Schiavone, Enrico	Securing Critical Systems through Continuous User Authentication and Non-repudiation	70
Schoukens, Maarten	Block-Oriented Identification using the Best Linear Approximation: Benefits and Drawbacks	74
Semeráth, Oszkár	Towards the Evaluation of Graph Constraint over Partial Models	**
Szárnyas, Gábor	Formalizing openCypher Graph Queries in Relational Algebra	**
Szilágyi, Gábor	Distributed Runtime Verification of Cyber-Physical Systems Based on Graph Pattern Matching	78
Tóth, Tamás	Timed Automata Verification using Interpolants	82
Virosztek, Tamás	Theoretical Limits of Parameter Estimation Based on Quantized Data	**
Zoppi, Tommaso	Executing Online Anomaly Detection in Complex Dynamic Systems	86

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

Towards Modeling Cyber-Physical Systems From Multiple Approaches

Márton Búr[1,2], András Vörös[1,2], Gábor Bergmann[1,2], Dániel Varró[1,2,3]
[1]Budapest University of Technology and Economics, Department of Measurement and Information Systems, Hungary
[2]MTA-BME Lendület Research Group on Cyber-Physical Systems, Hungary
[3]McGill University, Department of Electrical and Computer Engineering, Canada
Email: {bur, vori, bergmann, varro}@mit.bme.hu

***Abstract*—Cyber-physical systems are gaining more and more importance even in critical domains, where model-based development and runtime monitoring is becoming an important research area. However, traditional approaches do not always provide the suitable toolset to model their dynamic characteristics. In this paper, we aim to overview and highlight the strengths and limitations of existing runtime and design time modeling techniques that can help runtime monitoring and verification from the viewpoint of dynamic cyber-physical systems. We evaluated instance modeling, metamodeling, and metamodeling with templates, and provided example use-case scenarios for these approaches. We also overview the applicability of SysML in these contexts.**

I. Introduction

Critical cyber-physical systems (CPS) are appearing at our everyday life: healthcare applications, autonomous cars and smart robot and transportation systems are becoming more and more widespread. However, they often have some critical functionality: errors during the operation can lead to serious financial loss or damage in human life. Ensuring trustworthiness of critical CPS is an important task in their development and operation. CPSs have complex interactions with their environment, however, environmental conditions are rarely known at design time. In addition, the behavior of CPSs is inherently data dependent and they have smart/autonomous functionalities. These properties make design time verification infeasible. In order to ensure the safe operation of CPSs, one can rely on runtime verification. Various techniques are known from the literature for monitoring the different components constituting a CPS [1], however they do not provide system level assurance. Moreover, traditional monitoring techniques do not cover data dependent behavior and structural properties of the system.

Runtime verification is a technique to check if a system or a model fulfills the specification during operation by observing the inputs and outputs. It extracts information of a running system and checks whether it violates certain properties. We plan to use models at runtime as a representation of our knowledge of the systems. The model is built and modified according to the various information gathered for runtime verification.

This paper is partially supported by the MTA-BME Lendület 2015 Research Group on Cyber-Physical Systems.

Models created at design time can be subject to traditional verification techniques. They can guarantee the correctness of the system design. In contrast, runtime techniques analyze the runs of the system during its operation and they can reveal errors in the real life implementation.

Our approach is similar to the one of model-driven engineering (MDE) that facilitates problem description by utilizing domain specific languages such as domain specific models. In terms of this paper models are considered as typed graphs.

In this paper we introduce our envisioned approach for modeling dynamically changing, extensible cyber-physical systems. We aim to overview possible model-based approaches for IoT/cyber-physical system development, and will investigate how these techniques can provide support for runtime modeling of such systems.

II. Vision

According to our approach, there are two main modeling aspects: *design time* and *runtime*. For each aspect there are three main pillars of modeling:

- *Requirements* specify the properties the system must hold.
- *Environment information* represents the physical environment.
- *Platform information* describes the available sensors, computation nodes and actuators in the system, as well as encapsulates functional architecture with deployment information.

In the followings we summarize the goals of modeling at various phases of the development.

a) Design time models: Requirements, environment information, and execution platform information are present in a model of a CPS. The design time models describe (i) initial configuration, such as structure and topology, or (ii) behavior. The information contained in such models is static, the information captured by design time models does not change during runtime. The implementations of such designs are typically deployed software components or configuration files. During the design process the system also has to be prepared for different operational contexts, because most of its details are only known at runtime.

b) Runtime models: Runtime models, also called *live models*, capture information about the system dynamically. At runtime the actual system configuration is known, as well as sensors can provide details about the operational context of the system. From design time models deployment artefacts can be obtained, which link the design time information to the runtime models. This information is then used in the live model, eventually amended with additional knowledge.

Since the role of the design time and runtime models differ in the system life cycle, parts of runtime information may be omitted from design time models, such as values of data streams and events. Similarly, design time information may only be present in the runtime model in an abstract way.

For example, a camera and a computer is represented in the design time model, but the stream data is not available at design time, so that it is not present in the design model. Another example is when a controller and its parameters are represented in a design time model, but the live model for the controlled process only has a boolean value expressing whether the output complies to the requirements, but the used controller parameters are not included.

The purpose of the live model is both (i) to capture domain-specific information about the system and its functions, (ii) to describe the heterogeneous, dynamically changing platform, (iii) to describe its operational context and environment, and also (iv) to check runtime verification objectives.

Our vision of using live model-based computations for different purposes such as operation and monitoring of cyber-physical systems is depicted in Figure 1. In order to efficiently handle data obtained by sensors, we envision *sensor data integration* step to normalize and transform data so that it is adapted to the *live model*. *Computation* refers to the process of determining the required actuation to the physical environment and live model updates using the current state of the runtime model. This concept shown in Figure 1 can be used both for *live model-based control* and *runtime monitoring* of the system. The former actuates system processes, while the latter only observes the system and the environment using the live model.

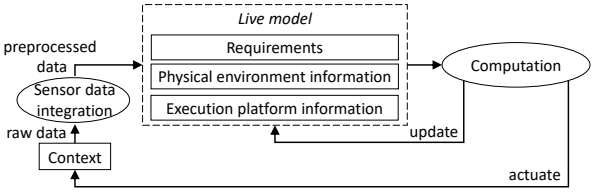


Fig. 1. Vision of using a live model

In terms of classical control theory concepts, this approach represents a *closed-loop control*-like mechanism, where the *physical processes within the system context* together are regarded as the *plant*, and *controller* tasks are realized by the *live model-based computation*. The processes in the system context may already be controlled processes.

III. Modeling Aspects

In this section we introduce and discuss both design time and runtime modeling approaches for dynamic cyber-physical systems. We detail the support provided by the *SysML* standard [2] for the approaches, as well as point out their missing features. We also illustrate the main challenges of defining and creating both design time and live models using an example of a fictitious smart warehouse. In example autonomous *forklifts* are operating, which are equipped with onboard cameras to detect changes in their environment. Due to space limitations we include examples about execution platform models, while modeling the requirements and environment information are not discussed.

A. Metamodeling

One of the basic modeling approaches is metamodeling. Using metamodels, one can define (i) node types, (ii) node attributes and (iii) relationship types. This allows the modeler to describe constraints regarding the overall structure of the system model, on the type-level.

For cyber-physical systems we consider each attribute as read-only by default, as they represent information sources, e.g. sensors. A special type `signal` denotes that the value of the property is time-varying, based on the data received. Signals can be *discrete time signals* or *continuous time signals*. In case of discrete time signals, their value may be changed at given time instants, but stays constant in between. For continuous time signals change in its value may occur anytime. Similarly to `signal`, the type `event` represents time-dependent information, but it is provided only at certain discrete time instants. There are signals of different types in the metamodel fragment depicted in Figure 2 as well, such as *feed* for a camera or *currentRPM* for ECUs.

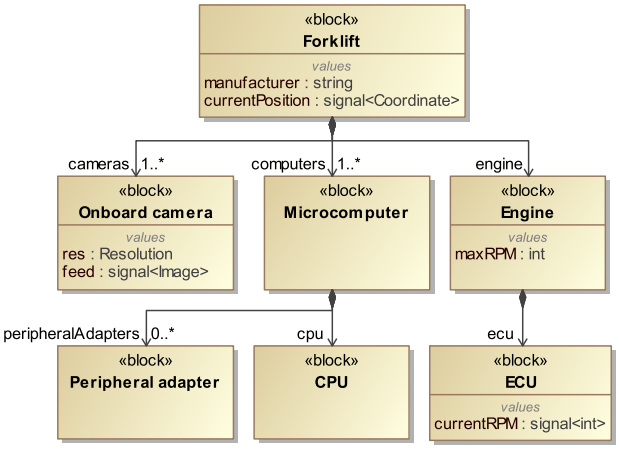


Fig. 2. Example metamodel with containment edges and attributes

Metamodels at design time can be used to create a functional model that satisfies the requirements, define platform model structure, and to describe the possible entities in the environment.

The part of the design time metamodel for the system representing the relevant platform information in our example system is illustrated in Figure 2. Containment edges show that the root container element is the *Forklift*, which contains an *Engine*, and at least one *Microcomputer*, and at least one *Onboard camera*. Microcomputers are further decomposed into *CPU*s and *Peripheral adapters*, while an engine encapsulates its corresponding *ECU*. Cross references between types in the model are not shown in the diagram.

According to our vision, models also hold information about runtime properties based on the requirements. In order to represent requirements as well, we defined *Goal*s for the type *SafetyCriticalDevice*, which is added as a supertype of forklift, as show in Figure 3. Goals are functions in the system that check whether the system holds the properties specified by the requirements. If a requirement stated that a device shall not collide with other devices, the corresponding goal would be a function that checks whether the device keeps enough distance from other trucks.

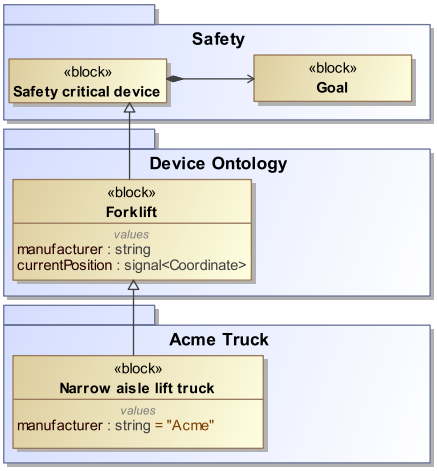


Fig. 3. Example for inheritance and packages for the forklift metamodel

Reusability is an important aspect in engineering. If a metamodel is already given for a domain, it is the best to have the corresponding model elements grouped as a toolset to make it available for reuse. For this reason, one can define *packages* that are similar to *libraries* in programming, containing model elements for the same domain.

When creating concrete applications, elements in general packages shall be specialized by *inheritance*, that can be used to specify fixed values for properties. Multiple packages can be used and specialized for the same application.

Figure 3 shows a possible packaging of types in our example. The package *Safety* contains essential concepts to include safety-related verification information in a model, the *Device Ontology* package is intended to hold different device types, such as forklift, and the *Acme Truck* package is the application-specific container. In our example *Narrow aisle lift truck* is a special type of forklifts, and each of its instances are manufactured by `Acme`.

The presented example figures show only views of the metamodel, so that additional relationships (such as containment, inheritance) as well as model elements, which are present in the model, are omitted from Figure 2 and Figure 3.

Design time metamodeling is facilitated at runtime to create instances based on the defined metamodel, where the model elements, relationships and properties are representing the knowledge-base of the system.

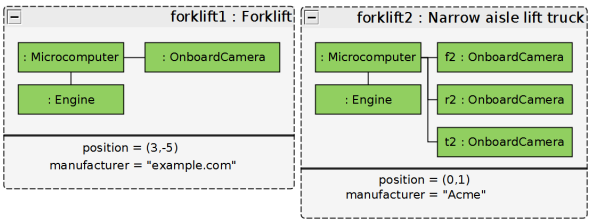
Support in SysML: SysML supports metamodeling by *block definition diagrams*. Focusing on cyber-physical systems, however, there is a need for elements that are not necessarily required for traditional software development. First, *flow properties* can be used to represent signals in SysML. Second, binding parameters can be expressed using the combination of default values and marking the parameter as read-only.

B. Instance Modeling

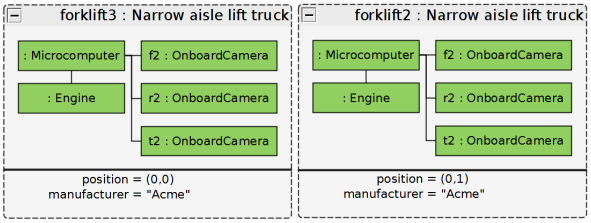
Instance models can describe a concrete configuration of the system, for which they can show multiple views. A typical usage for modeling requirements at design time is to create behavior models, such as statecharts.

Additionally, the environment and the platform can also be modeled on instance level at design time. However, it only has a limited usage to describe concrete arrangements – for example specifying test cases.

At runtime, however, views of the instance level model of the system platform and physical environment are essential. Considering our smart warehouse example, we can represent the system platform at time point t_0 with two *forklift*s, from which one is a special type of forklift named *narrow aisle lift truck*. The model is depicted in Figure 4a. Forklift main properties and their internal elements are also present. At a later point of time t_1, if the forklift leaves the warehouse and a new narrow aisle lift truck appears, the live model changes to as depicted in Figure 4b.



(a) Live model instance at t_0



(b) Live model instance at $t_1 > t_0$

Fig. 4. Live models at different points of time

Support in SysML: SysML has no dedicated support for instance-level modeling. However, the standard and best practices recommend using *block definition diagrams* (BDDs) to model snapshots of the system at design time.

C. Metamodels with Templates

One can define the structure of the instance models with metamodels. However, there are cases when it is desired to describe configurations including predefined attribute values and reference edges between certain types. For this purpose *templates* provide a solution to describe patterns in instance models. In our example metamodel, the forklift can have onboard cameras. Additionally in the template shown in Figure 5, we declare that *forklift* instances shall have a microcomputer unit, which has access to the engine and an onboard camera.

Fig. 5. Structure template of forklifts

Another template is defined for a microcomputer, where a CPU communicates with an ECU via a peripheral adapter. The ECU is not contained within the microcomputer, yet the peripheral adapter controls it, so that it is marked with dashed lines in Figure 6.

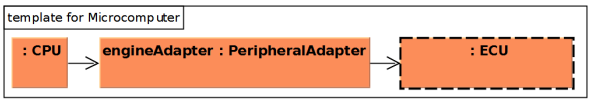


Fig. 6. Structure template of microcomputers

Furthermore, for the subtype *narrow aisle lift truck* this structure is changed, and the central microcomputer communicates with three different cameras, as depicted in Figure 7. This can be interpreted as there are exactly three onboard cameras in this type of truck, and the relation binding is formulated as $cameras = \{top, rear, front\}$. Additionally, for this specific type the *maxRPM* of the engine is 12000, and the front, rear and top camera resolutions are also bound to 800x600, 800x600, and 1504x1504, for each instance respectively.

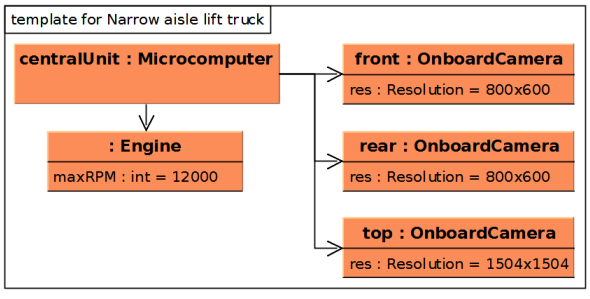


Fig. 7. Structure template of narrow-aisle lift truck

One of the main benefits of this approach is the description of certain runtime changes in the live model are more simple than in a purely metamodel-based case. For example, when a new forklift is added to the system, the change does not need to include the elements contained within the forklift or truck, for they are known from the template of the type, which can be a huge advantage when the model is changing frequently.

Support in SysML: SysML has *internal block diagrams* (IBDs) for template-like purposes. This description is also connected to a type, but is not found in traditional metamodels.

D. Strengths and Limitations of the Approaches

To conclude the introduction of the approaches, we summarize their strengths and limitations.

Strengths: The introduced modeling techniques support both design time and runtime modeling and analysis as long as the metamodel of the system is known at design time and remains unchanged during runtime.

Limitations: The cornerstone of each of the introduced approaches above is a metamodel introducing domain-specific types. It provides a basis for prescriptive modeling, which means model instances can only have elements of the types and relations defined within the metamodel. However, in case of dynamic cyber physical systems and IoT applications it is possible to extend the system at runtime with new components having new types. A solution for this issue can be to use ontologies, where types are assigned to the entities in a descriptive way. In such cases new objects can be classified using the types included in the ontology based on their capabilities.

IV. Conclusion

We overviewed design time and runtime modeling solutions to describe cyber-physical systems. We discussed metamodeling, instance modeling, and metamodeling with templates approaches, and provided use-case scenarios for them, as well as added examples how SysML supports each technique.

The provided overview has only covered a few main modeling aspects. In [3] the authors introduce their concept of evolutionary design time models that try to minimize the discrepancy between the design time and runtime concepts. They aim to minimize static information in design time models.

Additionally, there are many ways to extend the system description, one of them is modeling uncertainty. Uncertainty in cyber-physical system modeling is discussed in [4]. It is also a possible direction for model-based description of such systems to include probability and uncertainty models in graph-like live model-based representations.

References

[1] S. Mitsch and A. Platzer, "Modelplex: verified runtime validation of verified cyber-physical system models," *Formal Methods in System Design*, vol. 49, no. 1-2, pp. 33–74, 2016.

[2] Object Management Group, "OMG Systems Modeling Language," p. 320, 2015. [Online]. Available: http://www.omg.org/spec/SysML/1.4/PDF/

[3] A. Mazak and M. Wimmer, "Towards liquid models: An evolutionary modeling approach," in *18th IEEE Conference on Business Informatics (CBI)*, 2016, pp. 104–112.

[4] M. Zhang, B. Selic, S. Ali, T. Yue, O. Okariz, and R. Norgren, "Understanding uncertainty in cyber-physical systems: A conceptual model," in *Proc. of Modelling Foundations and Applications (ECMFA)*, 2016, pp. 247–264.

Well-Formedness and Invariant Checking of PLCspecif Specifications

Dániel Darvas*†, István Majzik* and Enrique Blanco Viñuela†
*Budapest University of Technology and Economics, Department of Measurement and Information Systems
Budapest, Hungary, Email: {darvas,majzik}@mit.bme.hu
†European Organization for Nuclear Research (CERN), Beams Department
Geneva, Switzerland, Email: {ddarvas,eblanco}@cern.ch

***Abstract*—Developers of industrial control systems constantly quest for quality in order to improve availability and safety. Some of the threats to quality are the development errors due to incorrect, ambiguous or misunderstood requirements. Formal specification methods may reduce the number of such issues by having unambiguous, mathematically sound semantics, which also allows the development of precise analysis methods. In this paper we present two of the analysis methods developed for PLCspecif, our formal specification language targeting PLC modules: well-formedness analysis and invariant analysis.**

I. Introduction and Background

Industrial control systems (ICS) operate a wide variety of plants: e.g. chemical, pharmaceutical, or water treatment plants [1]. Typically these systems rely on programmable logic controllers (PLCs) which are robust industrial machines, programmed in variants of the languages defined in IEC 61131-3 [2]. As we rely more and more on such systems, their safety and availability is a priority. While these systems are often not safety-critical as there are dedicated systems to ensure the safety, an outage might cause significant economic loss.

The development errors of the PLC software are threats to availability and safety. Many of them are caused by ambiguous, imprecise, or incorrect requirements. Formal specifications provide ways to improve the quality of the specification both by providing restricted and precise languages (compared to natural languages) and by having unambiguous semantics, thus allowing the usage of automated analysis methods which can reveal potential problems, such as contradictions, impossible cases, etc.

The usage of formal specification in the ICS domain is not wide yet, as the existing formal specification languages are not well-suited for the target audience. The existing and widely-known, general-purpose formal specification methods are not adapted to the ICS domain, therefore their usage necessitates deep knowledge and excessive effort. Such effort can only be justified in case of highly critical systems.

The authors analysed the specialities of the ICS domain [3] and proposed PLCspecif, a formal language to specify the behaviour of PLC software modules [4]. We claim that the specification of PLC software modules (either isolated safety logics or reusable objects) are the first targets for formal specification, because their specification provides a good effort–benefit ratio. However, a formal specification method does not guarantee by itself that the specified behaviour is well-formed, correct and matches the intentions. In this paper we target these challenges by providing two analysis methods for PLCspecif.

The rest of the paper is structured as follows. Section II briefly overviews the related work. Section III introduces the key concepts of PLCspecif, our formal specification language for PLC software modules. Next, two analysis methods are discussed: Section IV shows the well-formedness checking of PLCspecif specifications, then Section V presents invariant checking. Finally, Section VI concludes the paper.

II. Related Work

Neither formal specification, nor static analysis is a widely-used technique in the field of PLC software development.

The static analysis of PLC programs was targeted in [5], [6]. Some commercial tools are available as well, such as *PLC Checker*[1] or *logi.LINT*[2]. Although static code analysis may point out mistakes or code smells without requiring excessive effort, due to the wide variety of PLC languages and the different needs in the various application domains, the usage of static analysis tools is not general yet.

Formal specification is even less wide-spread in the PLC software development domain. As discussed earlier, the usage of general-purpose specification languages need great effort, therefore they are only applied in highly critical systems. There were several attempts to provide formal or semi-formal specification methods, directly targeting PLC programs. *ST-LTL* [7] is a variant of the LTL formalism, specifically targeting PLC programs. As this method is rather simple, there is no need for static analysis methods.

ProcGraph [8], [9] is a semi-formal specification method based on state machines. According to the published examples, the specifications may become large and complex. The specifications often include PLC code snippets, this also increases the complexity and the difficulty of understanding and therefore the specification might be error-prone. However, static analysis or invariant checking methods are not mentioned as part of the proposed approach [9].

NuSCR [10] is a formal specification method included in the NuSEE environment for specification and verification of

[1] http://www.itris-automation.com/plc-checker/
[2] http://www.logicals.com/en/add-on-products/151-logi-lint

PLC-based safety-critical systems. Supposedly NuSRS, the requirement specification editor based on the NuSCR approach included in NuSEE contains certain well-formedness checks, but static analysis and invariant checking is not explicitly mentioned in [10]. The NuSDS tool that is to be used in the design phase contains certain consistency checks, but the details are not discussed.

III. The PLCspecif Specification Language

The PLCspecif language is designed to provide an easy-to-use, practice-oriented, yet formal specification of PLC software module behaviours. The behaviour description is complete (the description does not contain abstract parts left for later decision), this is why we emphasise that PLC software modules are targeted, where such complete specification is feasible.

In [3] the authors reviewed the literature and the real needs experienced in the PLC-based ICS development processes at the European Organization for Nuclear Research (CERN). Such requirements towards the formalism are domain-specificity, appropriate event semantics, support for a variety of formalisms and decoupling of input/output handling (pre- and post-processing) from the core logic definition [3].

Based on these identified needs, PLCspecif was designed to be a hierarchical, module-based specification language. Each module is either a composite module (whose behaviour is described by submodules) or a leaf module (describing a certain part of the global behaviour). Each module has several parts: input and event definitions, core logic definition, and output definitions. The core logic of the leaf modules can be described using one of the three defined formalisms: state machines, input-output connection diagrams (a data-flow description formalism adapted to PLC programs) or PLC timers. These formalisms have domain-specific, special, unified semantics which allow to mix multiple formalisms in the same specification and to use the most appropriate formalism for each part of the described behaviour.

The semantics of PLCspecif is described formally. For this we have defined a simple, low-level timed automata (TA) formalism, and a precise mapping of PLCspecif specifications to this TA formalism. TA was chosen to be the underlying formalism for semantics definition to facilitate the usage of formal verification directly on the specification.

Due to space restrictions we omit the detailed discussion of the syntax and semantics of PLCspecif. The reader find the formal definition of PLCspecif in our report [11].

The formal specification defines the desired behaviour in an unambiguous way. However, besides the clean description, various methods were developed to make PLCspecif more useful in practice. A *code generation* method was designed [12] that constructs PLC code with a behaviour that corresponds to its specification. In case of legacy or safety systems this method might not be appropriate or applicable, thus a method is required to show the correspondence of an existing PLC code to a specification. For this reason, a *conformance checking* method was designed too [13]. It checks the conformance between the implementation and the specification using model checkers, with configurable level of conformity.

TABLE I
Categories of Well-Formedness Rules

	Category	# Rules
(1)	Field value uniqueness	3
(2)	Object-local checks	12
(3)	Reference restrictions	4
(4)	Restricting non-local references	9
(5)	Expression element restrictions	13
(6)	Complex structural checks	18
(7)	Type constraints	10
(8)	Complex semantic checks	3
	Total	72

However, neither code generation, nor conformance checking can guarantee a correct implementation if the specification is incorrect, contradictory or malformed. Therefore we developed additional methods to detect malformed or unintentional behaviour descriptions and therefore to increase the confidence in the correctness of the developed formal specifications. In the following sections two such methods are introduced: well-formedness checking and invariant checking.

IV. Well-Formedness Checking

The syntax and semantics of the PLCspecif language is defined in our report [11]. It provides a metamodel (abstract syntax) and a concrete syntax for the language, furthermore informal and formal semantics. However as usual, the metamodel could not express all constraints that a specification (instance model of the metamodel) has to respect in order to be considered correct and meaningful. Therefore further restrictions are formulated as well-formedness rules.

We defined 72 well-formedness rules for PLCspecif in [11], which are additional restrictions to the abstract syntax besides the metamodel of the language, ensuring that the specification is well-formed, meaningful and deterministic. A possible categorisation of the rules is shown in Table I. For example, rules in group *(3)* define additional restrictions for the references of the objects. Rules in group *(4)* restrict the reference to non-local elements (e.g. referring to a state of a state machine from a different module). The rules in group *(5)* restrict the set of elements of an expression (e.g. an expression which is not in a state machine module should not refer to a state). Group *(7)* contains rules constraining the expression types (e.g. a transition guard should have a Boolean type).

Most of these rules (mainly in groups *(1)–(5)*) are simple, restricting the use of certain incorrect or undefined constructs that are not forbidden by the metamodel; restricting the types of references; or prescribing name uniqueness.

Obviously, the definition of these rules is not enough, it has to be checked automatically whether they are respected or not. This checking is done after reading and parsing the specification, on its abstract syntax graph (instance model of the defined metamodel). Most of the simple well-formedness

rules (in groups *(1)–(5)*) are implemented in Java. Many of these rules can be efficiently implemented in a dozen lines of code, not necessitating any more advanced methods. Others, typically rules in groups *(6)–(7)*, need more implementation effort, e.g. to check whether the guard of a transition has a Boolean type or not, the type inference has to be implemented.

The implementation of the complex semantics checks (rules in group *(8)*) is more difficult, for example to analyse that the outgoing transitions of a certain state are mutually exclusive (no conflict is possible); or that it is not possible to have an infinite transition firing run in the state machines. In these cases the manual implementation of checking these rules would require significant effort. However, these rules can be transformed into a SAT (Boolean satisfiability) problem. For example, to check whether a state does not have any conflicting outgoing transitions, the guards and priorities of these transitions have to be collected, then it has to be checked whether there exists a pair of transitions on the same priority level with guards that can be satisfied at the same time.

We have used the Microsoft Z3 SAT solver [14] for this purpose, as it provides state-of-the-art algorithms, good performance and Java integration. The usage of Z3 provides an efficient and automated way (thus hidden from the user) to check these more complex rules. Besides checking the satisfaction, the witness ("model" in SAT terminology) generated by Z3 can help the user by pointing out the source of the violation of the well-formedness rules.

PLCspecif was used for the specification of two real examples since its creation: for a reusable PLC module library of the UNICOS framework[3], and the logic of a safety-critical controller used at CERN [15]. In both cases the well-formedness checking was able to identify mistakes made during specification, for example conflicting definitions, unused variables, ambiguous priority settings and conflicting guard expressions.

V. Invariant Checking

The static analysis of well-formedness rules helps to ensure that the specification follows the rules of PLCspecif which are not enforced by the syntax itself. This is required for any specification.

However, in certain cases additional requirements have to be checked too. The formalisms provided for the core logic definition in PLCspecif (state machines, input-output connection diagrams, PLC timers) focus mainly on the *functionality* of the defined module. The specification of the behaviour may hide safety or invariant properties that are required to be satisfied by the specified module. For example, it might not be obvious to see the satisfaction of requirements such as "Outputs `a` and `b` shall not be *true* at the same time." or "Output `c` shall always be within the range `d..e`." based on the state machine-based core logic definitions and the output definitions.

Previous work on the model checking of PLC programs [16] demonstrated that checking various, typically invariant or safety properties on PLC programs using model checkers is feasible and may uncover well-hidden faults in the implementations. One of the difficulties of this approach is the formalisation of models and the properties to be checked for the model checkers. To reduce this obstacle, we have developed *PLCverif* [17], a tool that automatically generates artefacts for model checkers using inputs that are known to the PLC developers. The *models* are generated from the source code of the PLC programs, via an intermediate model (IM). The IM formalism allows the model checker-independent reduction of the formal models and facilitates to use multiple widely-known model checkers (e.g. nuXmv, UPPAAL, ITS). The *property* for model checking is defined using requirement patterns which are easy-to-fill fixed sentences in English with given placeholders, e.g. "If α is true (at the end of the PLC cycle), then β is true (at the end of the same cycle)", where α and β are placeholders of Boolean expressions, composed of input and output signals, constants, comparison and logic operators. The result of the verification in PLCverif is a human-readable verification report. It demonstrates the violation of properties (if any) by diagnostic traces from the model.

The defined PLC modules should often satisfy certain safety and invariant properties. Explicitly declaring and verifying these properties may greatly improve the quality of the PLC software modules. Therefore we have included specific support in PLCspecif to capture these invariant properties. In each module the specifier may define properties which have to be always satisfied by the given (sub)module, more precisely after the execution of the (sub)module the defined invariant properties have to be satisfied. As discussed above, the satisfaction of these requirements may be checked using model checkers, similarly to the approach followed in PLCverif. This is discussed in the rest of the section.

a) Representing the Requirement for Model Checking: For model checking, the only need is to be able to represent the invariant properties as temporal logic formulae. The usage of requirement patterns to hide the complexity of describing properties with temporal logic seems to be a convenient way, but in the future additional property specification methods can also be incorporated, provided that they can be represented as computation tree logic (CTL) or linear temporal logic (LTL) formulae. For example, the above-presented requirement pattern can be represented in CTL as "$\mathsf{AG}((EoC \wedge \alpha) \rightarrow \beta)$", where EoC is true at the end of the PLC cycle only.

b) Representing the Specification for Model Checking: We recall that the formal semantics of PLCspecif is given as a construction of an equivalent timed automaton. The choice of this semantics description was partially to facilitate the formal verification of the PLCspecif specifications. The intermediate model used in PLCverif is also an automata-based formalism with matching semantics. The timed automata elements (e.g. edges) can be systematically translated to the corresponding IM elements (e.g. transitions). The only exception is the clock of the timed automata, which do not have corresponding element in IM. The representation of time-related behaviour in the IM is further discussed in [18].

The semantics of the corresponding elements are defined

[3] http://cern.ch/unicos/

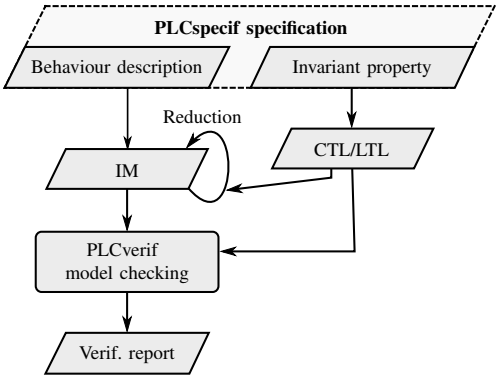


Fig. 1. Workflow of invariant checking

identically. Therefore by using the semantics description of PLCspecif it is feasible to develop a systematic, element-by-element mapping from a PLCspecif specification to a PLCverif IM. Then the IM can be transformed by PLCverif into the format required by the applied model checker tool. This method also benefits from the model reduction methods that are already included in PLCverif.

The complete invariant checking workflow based on PLCverif is depicted in Figure 1. The behaviour description of the specification is transformed into the IM formalism, then automatically reduced. The invariant properties are represented in CTL or LTL. The CTL/LTL formulae are not only used to check the satisfaction of the invariant property, but they also influence the reductions. PLCverif executes the selected external model checker tool, then the result is presented to the user.

The invariant checking was used in the re-specification of the previously mentioned reusable PLC module library of UNICOS. It was possible to define and verify invariant properties such as "If the *manual* mode is inhibited, the module should not switch to *manual* mode", directly on the intermediate model generated from the specification, before generating the corresponding source code.

VI. Summary

This paper discussed the static well-formedness analysis and invariant property checking features that are incorporated in the PLCspecif specification approach. The well-formedness checking methods help to ensure that the formal specification is consistent and well-formed; that it respects properties that are required for any formal specification. By using invariant checking it can also be checked whether the designed specification matches the intentions of the specifier. If the user declares safety or invariant properties explicitly in a PLCspecif specification, their satisfaction can be checked by reusing the model checking approach included in the PLCverif tool. This may reveal problems which would be hidden despite the use of formal specification. By checking these general and specific properties of the specification, the quality of the specification can be improved, which has a positive effect on the correctness of the implementation through the code generation and conformance checking methods.

References

[1] K. Stouffer, V. Pillitteri, S. Lightman, M. Abrams, and A. Hahn, "Guide to industrial control systems (ICS) security," National Institute of Standards and Technology, Special Publication 800-82 rev. 2, 2015.

[2] *IEC 61131-3:2003 Programmable controllers – Part 3: Programming languages*, IEC Std., 2003.

[3] D. Darvas, I. Majzik, and E. Blanco Viñuela, "Requirements towards a formal specification language for PLCs," in *Proc. of the 22nd PhD Mini-Symposium*. Budapest University of Technology and Economics, Department of Measurement and Information Systems, 2015, pp. 18–21.

[4] D. Darvas, E. Blanco Viñuela, and I. Majzik, "A formal specification method for PLC-based applications," in *Proc. of the 15th International Conference on Accelerator and Large Experimental Physics Control Systems*, L. Corvetti *et al.*, Eds. JACoW, 2015, pp. 907–910.

[5] S. Stattelmann, S. Biallas, B. Schlich, and S. Kowalewski, "Applying static code analysis on industrial controller code," in *19th IEEE International Conference on Emerging Technology and Factory Automation (ETFA)*. IEEE, 2014.

[6] H. Prähofer, F. Angerer, R. Ramler, and F. Grillenberger, "Static code analysis of IEC 61131-3 programs: Comprehensive tool support and experiences from large-scale industrial application," *IEEE Transactions on Industrial Informatics*, 2016, advance online publication, http://doi.org/10.1109/TII.2016.2604760.

[7] O. Ljungkrantz, K. Åkesson, M. Fabian, and C. Yuan, "A formal specification language for PLC-based control logic," in *8th IEEE International Conference on Industrial Informatics (INDIN)*, 2010, pp. 1067–1072.

[8] T. Lukman, G. Godena, J. Gray, M. Heričko, and S. Strmčnik, "Model-driven engineering of process control software – beyond device-centric abstractions," *Control Engineering Practice*, vol. 21, no. 8, pp. 1078–1096, 2013.

[9] G. Godena, T. Lukman, M. Heričko, and S. Strmčnik, "The experience of implementing model-driven engineering tools in the process control domain," *Information Technology and Control*, vol. 44, no. 2, 2015.

[10] S. R. Koo, P. H. Seong, J. Yoo, S. D. Cha, C. Youn, and H. Han, "NuSEE: An integrated environment of software specification and V&V for PLC based safety-critical systems," *Nuclear Engineering and Technology*, vol. 38, no. 3, pp. 259–276, 2006.

[11] D. Darvas, E. Blanco Viñuela, and I. Majzik, "Syntax and semantics of PLCspecif," CERN, Report EDMS 1523877, 2015. [Online]. Available: https://edms.cern.ch/document/1523877

[12] ——, "PLC code generation based on a formal specification language," in *14th IEEE International Conference on Industrial Informatics (INDIN)*. IEEE, 2016, pp. 389–396.

[13] D. Darvas, I. Majzik, and E. Blanco Viñuela, "Conformance checking for programmable logic controller programs and specifications," in *11th IEEE International Symposium on Industrial Embedded Systems (SIES)*. IEEE, 2016, pp. 29–36.

[14] 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, C. R. Ramakrishnan and J. Rehof, Eds. Springer, 2008, vol. 4963, pp. 337–340.

[15] D. Darvas, I. Majzik, and E. Blanco Viñuela, "Formal verification of safety PLC based control software," in *Integrated Formal Methods*, ser. Lecture Notes in Computer Science, E. Ábrahám and M. Huisman, Eds. Springer, 2016, vol. 9681, pp. 508–522.

[16] 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 Transactions on Industrial Informatics*, vol. 11, no. 6, pp. 1400–1410, 2015.

[17] D. Darvas, B. Fernández Adiego, and E. Blanco Viñuela, "PLCverif: A tool to verify PLC programs based on model checking techniques," in *Proc. of the 15th International Conference on Accelerator and Large Experimental Physics Control Systems*, L. Corvetti *et al.*, Eds. JACoW, 2015, pp. 911–914.

[18] B. Fernández Adiego, D. Darvas, E. Blanco Viñuela, J.-C. Tournier, V. M. González Suárez, and J. O. Blech, "Modelling and formal verification of timing aspects in large PLC programs," in *Proc. of the 19th IFAC World Congress*, ser. IFAC Proceedings Volumes, E. Boje and X. Xia, Eds., vol. 47 (3). IFAC, 2014, pp. 3333–3339.

Approaches to Identify Object Correspondences Between Source Models and Their View Models

Csaba Debreceni[1,2], Dániel Varró[1,2,3]
[1]Budapest University of Technologies and Economics, Department of Measurement and Information Systems, Hungary
[2]MTA-BME Lendület Research Group on Cyber-Physical Systems, Hungary
[3]McGill University of Montreal, Department of Electrical and Computer Engineering, Canada
Email: {debreceni,varro}@mit.bme.hu

***Abstract*—Model-based collaborative development of embedded, complex and safety critical systems has increased in the last few years. Several subcontractors, vendors and development teams integrate their models and components to develop complex systems. Thus, the protection of confidentiality and integrity of design artifacts is required.**

In practice, each collaborator obtains a filtered local copy of the source model (called view model) containing only those model elements which they are allowed to read. Write access control policies are checked upon submitting model changes back to the source model. In this context, it is a crucial task to properly identify that which element in the view model is associated to which element in the source model.

In this paper, we overview the approaches to identify correspondences between objects in the filtered views and source models. We collect pros and cons against each approach. Finally, we illustrate the approaches on a case-study extracted from the MONDO EU project.

I. Introduction

Model-based systems engineering has become an increasingly popular approach [1] followed by many system integrators like airframers or car manufacturers to simultaneously enhance quality and productivity. An emerging industrial practice of system integrators is to outsource the development of various components to subcontractors in an architecture-driven supply chain. Collaboration between distributed teams of different stakeholders (system integrators, software engineers of component providers/suppliers, hardware engineers, specialists, certification authorities, etc.) is intensified via the use of models.

In an *offline collaboration* scenario, collaborators check out an artifact from a version control system (VCS) and commit local changes to the repository in an asynchronous long transaction. Several collaborative modeling frameworks exist (CDO [2], EMFStore [3]), but security management is unfortunately still in a preliminary phase. Traditional VCSs (Git [4], Subversion [5]) try to address secure access control by splitting the system model into multiple fragments, but it results in inflexible model fragmentation which becomes a scalability and usability bottleneck (e.g. over 1000 model fragments for automotive models).

This paper is partially supported by the EU Commission with project MONDO (FP7-ICT-2013-10), no. 611125. and the MTA-BME Lendület 2015 Research Group on Cyber-Physical Systems.

In our previous works [6], we introduced a novel approach to define fine-grained access control policies for models using graph queries. A bidirectional graph transformation so called *lens* is responsible for access control management. The forward transformation derives consistent *view models* by eliminating undesirable elements from the *source model* according to the access control rules that restrict the visibility of the objects, references and attributes. In contrast, the backward transformation propagate the changes executed on the view models back to the source model. It also enforces the write permission defined in the policies by rejecting all the changes if any of them violates an access control rule.

At commit time, the executed operations and their orders are not available in the most cases, only the deltas are sent to the VCS. It is an urgent task to correctly identify the modified elements of the model to recognize whether an access control rule is violated. Thus object correspondences need to be built between the element of the source and the modified views.

In this paper, first we motivate the need of identification of object correspondences using a case study from MONDO EU FP7 project. Then we overview the possible approaches and discuss their advantages and disadvantages that can be applied onto our existing lens-based approach.

II. Preliminaries

A. Instance Models and Modeling Languages

A metamodel describes the abstract syntax of a modeling language. It can be represented by a type graph. Nodes of the type graph are called classes. A class may have attributes that define some kind of properties of the specific class. Associations define references between classes. Attributes and references altogether are called features.

The instance model (or, formally, an instance graph) describes concrete systems defined in a modeling language and it is a well-formed instance of the metamodel. Nodes and edges are called objects and links, respectively. Objects and links are the instances of modeling language level classes and associations, respectively. Attributes in the metamodel appear as slots in the instance model.

B. Enforce Access Control Policies by Graph Transformation

In the literature of bidirectional transformations [7], a lens (or view-update) is defined as an asymmetric bidirectional transformations relationship where a source knowledge base completely determines a derived (view) knowledge base, while the latter may not contain all information contained in the former, but can still be updated directly.

The kind of the relationship we find between a source model (containing all facts) and a view model (containing a filtered view) fits the definition of a lens. After executing the transformation rules, model objects of the two models reside at different memory addresses, so the transformation must set up a one-to-one mapping called *object correspondence*, that can be used to translate model facts when propagating changes.

We assume that the forward transformation of the lens builds correspondences between objects of source and view models. But these correspondence relation cannot be guaranteed when the derived view model is reloaded as a new model because the new objects will not share the same memory addresses.

Rebuilding correspondence mapping between the source model and the modified view model is cumbersome, where the view may hide most of the sensitive information. Instead, correspondences are easier to build between the unmodified and modified view model as it is depicted in Fig. 1, then the originally achieved mapping can be used.

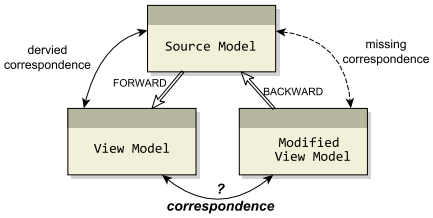


Fig. 1. Request to build correspondence between view models

C. Model Comparison

Building correspondences between two version of the same model is common problem in model versioning called *model comparison*. Model comparison process is responsible for identifying differences of two model and translate them into elementary model operations such as *create*, *update* and *delete*. A common issue in this context is to recognize whether an object is moved to another place or an existing object is deleted and a completely new one is created in the model with the same attribute values. Thus it is required to build correspondences between the two model to properly identify the differences.

III. Motivating Example

Several concepts will be illustrated using a simplified version of a modeling language (metamodel) for system integrators of offshore wind turbine controllers, which is one of the case studies [8] of the MONDO EU FP7 project. The metamodel, depicted by Fig. 2, describes how the system is modeled as modules providing and consuming signals. Modules are organized in a containment hierarchy of composite modules, ultimately containing control unit modules responsible for a given type of physical device (such as pumps, heaters or fans). Composite modules may be shipped by external vendors and may express protected intellectual property (IP).

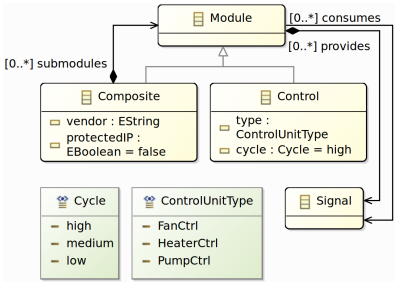


Fig. 2. Simplified wind turbine metamodel

A sample instance model containing a hierarchy of 2 `Composite` modules and a `Control` units, providing a `Signal` altogether, is shown on the top left side of Fig. 3 called *source model*. Boxes represent objects (with attribute values as entries within the box), while arrows represent containment edges and cross-references.

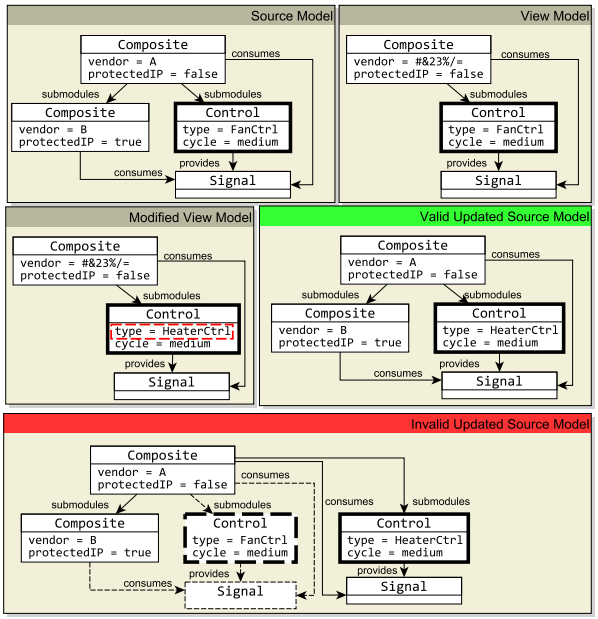


Fig. 3. Example instance model

Access Control. Specialists are engineers responsible for maintaining the model of control unit modules and have specific access to the models, described in the following:

R1. Intellectual properties have to be hidden from specialists.
R2. Objects cannot be created or deleted in the system model.
R3. Vendor attribute of visible composites must be obfuscated.

R4. Control units and their attributes can be modified.

According to the aforementioned access control rules, view models depicted in the top middle of Fig. 3 are derived for specialists where the protected IP objects are not visible and the vendor attributes are obfuscated. Only the control unit (marked with bold border) is allowed to be modified by specialists.

Scenario. At a given point, a specialist changes the type of the control unit from `FanCtrl` to `HeaterCtrl` represented on the top right side of Fig. 3 and propagate the modifications from view model back to the source model. It is need to be decided whether the change was allowed or not. Two cases can arise: (i) the VCS realizes that only the type attribute was modified; or (ii) the VCS interprets the change as the deletion of the original control unit and an addition of a new control unit. The former case will be accepted (*valid updated source model* on Fig. 3) while the latter one need to be rejected ((*invalid updated source model* on Fig. 3)) as it removes the `control` unit and its `signal` with the related references (marked with dashed borders and edges). Thus, the VCS has to identify which object has changed to be able to make a proper decision.

IV. Overview of the Approaches

In this section, we categorize the possible approaches to based on comparison techniques collected in [9]. For each approach we discuss its advantages and disadvantages and provide their application onto our running example.

A. Static Identifiers

Several modeling environments automatically provide unique identifiers for each object. The requirements against the identifiers are the following:

SI1. Identifiers need to assign to all objects.
SI2. Recycling of identifiers are not allowed.
SI3. Identifiers cannot be changed after serialization.
SI4. After deserialization, the identifiers need to remain.

For instance, the Industry Foundation Classes [10] (IFC) standard, intended to describe building and construction industry data, assigns unique number at element creation time. At the beginning it assigns 0 for the first object and then it increases the previous assigned identifier with 1. In case of the Eclipse Modeling Framework [11] (EMF), unique identifiers are assigned at serialization time if the serialization format supports this features (e.g. *XMI* format supports, but *Binary* not). In practical, a universally unique identifier (UUID) is generated for each object that still does not have any.

Advantages. Static identifiers require no user specific configuration. Always provides a perfect match for correspondences.

Disadvantages. Modeling environments or serialization format need to be changed. Moreover, it is possible, that the modeling tools do not support these formats.

Example. For our running example, static identifiers can be achieved using a proper serialization format that provide unique identifiers.

B. Custom Identifiers

In practice, domain language developers usually prepare their languages to support identifiers by adding a common ancestor for all classes that provides an identifier attribute. During the development phase, engineers need to manually set the identifiers for each object where the uniqueness cannot be guaranteed. Moreover, access control rules must make that attribute visible (at least in an obfuscated form) in views.

Advantages. There is no need to change modeling environment or model serialization.

Disadvantages. Existing languages need to be modified which may lead to inconsistencies. Uniqueness is questionable.

Example. Fig. 4 shows a possible extension of the aforementioned metamodel with a `NamedElement` interface. All the classes inherit the `id` attributes.

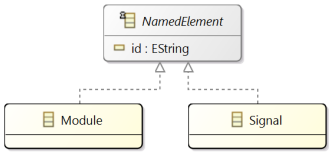


Fig. 4. Identifier introduced in a common ancestor

C. Signature-based Matching

Signature-based matching does not rely on unique identifiers, instead it calculates signatures of the objects. A signature is a user-defined function described as a model query. This approach is introduced in [12], whereas Hegedus et al. [13] described a similar approach so called *soft traceability links* between models. All the references and attributes involved in the calculation of a signature need to be visible (at least in and obfuscated form).

Advantages. There is no need to change modeling environment or serialization format, thus the modeling tools will still support the models.

Disadvantages. Users need to specify how to calculate the signature, which might lead to several false positive results.

Example. For our running example, a simple signature query defined in ViatraQuery language [14] is represented in Listing 1, which makes two control units identically equal if their `container` modules, the value of their `cycle` attributes and their provided `signal` objects are identically equals. This query successfully identifies changes introduced in the running example. However, it cannot recognize the deletion and addition of two different control units at the same position.

```
1 pattern sign(ctrl,cycle,sig,container) {
2   Control.eContainer(ctrl,container);
3   Control.cycle(ctrl,cycle);
4   Control.provides(ctrl,sig);
5 }
```

Listing 1. Example Signature Query

D. Similarity-based Matching

Similarity-based matching tries to measure the similarity between objects based on the *similarity value*. In contrast,

identifiers and signatures directly decide whether a correspondence exists between two objects. Similarity is calculated by the values of each features. For each feature, users need to specify a weight that define how important is it in the identification. Using these weights, meta-model independent algorithms derive the correspondences between the objects.

For instance, EMF Compare [15] is comparison tools to compare EMF models, and use similarity based-matching. Its calculation includes analyzing the name, content, type, and relations of the elements, but it also filters out element data that comes from default values etc.

Advantages. The identification is based on general heuristics and algorithms, where the users do not need to provide complex description on how to identify an object.

Disadvantages. Users need to specify weight for the features to fine-tune the similarity algorithms.

Example. A possible list of weights is defined in Listing 2, where the references of the aforementioned metamodel have more influence on the similarity than the attributes. In this case, our example modifications will be successfully recognized. However, Listing 3 describes a context, where the attributes are more important than the others. Thus, if we change the value of an attribute, it will be recognized as a deletion of an object and the creation of a new one.

```
1 wieghts
2 * container: 2
3 * provides:  2
4 * type: 0
5 * cycle: 0
6 * vendor: 0
7 * consumes: 0
8 * submodules: 0
9 * protectedIP: 1
```

Listing 2.
Weights with environment pressure

```
1 wieghts
2 * container: 0
3 * provides:  0
4 * type: 5
5 * cycle: 2
6 * vendor: 2
7 * consumes: 0
8 * submodules: 0
9 * protectedIP: 1
```

Listing 3.
Weights with attribute pressure

E. Language-specific Algorithms

Language-specific algorithms are designed to a given modeling language. Thus these approaches can take the semantics of the languages into account to provide more accurate identification of objects. For instance, a UML-specific algorithm can use the fact that two classes with the same name mean a match and it does not matter where they were moved in the model. UmlDiff [16] tool uses similar approach for differencing UML models. To ease the development of such a matching algorithms, the Epsilon Comparison Language (ECL) [17] can automate the trivial parts of the process, where developers only need to concentrate on the logical part.

Advantages. Semantics of the language are used and there is no need to any modification in the model or modeling tools.

Disadvantages. Users need to specify a complete matching algorithm for a given language which can be challenging.

Example. A simple matching rule defined with ECL is presented in Listing 4. It matches a `s` control unit with Fig.t control unit (declared in `match-with` part) if their container and the provided signals are equal (declared in `compare` part).

```
1 rule MatchControls
2   match s : Control
3   with t : Control {
4     compare {
5       return s.container = t.container
6       and s.provides = t.provides
7     }
8 }
```

Listing 4. Example Rule in Epsilon Comparison Language

V. Conclusion and Future Work

In this paper, we aimed to overview the approaches to identify correspondences between an original model and its filtered and modified version. We categorized these approaches into 5 groups - using *static identifiers* or *custom identifiers*, calculating *signature-based matches*, aggregating values of features using *similarity-based matching* and providing *language specific algorithms*. We introduced their application on a case study extracted from MONDO EU project and discussed their pros and cons.

As future work, we plan to integrate these approaches into our query-based access control approach [6] and evaluate them from the aspects of usability and scalability.

References

[1] J. Whittle, J. Hutchinson, and M. Rouncefield, "The state of practice in model-driven engineering," *IEEE Software*, vol. 31, no. 3, pp. 79 – 85, 2014.

[2] The Eclipse Foundation, "CDO," http://www.eclipse.org/cdo.

[3] ——, "EMFStore," http://www.eclipse.org/emfstore.

[4] Git, "Git," https://git-scm.com/.

[5] Apache, "Subversion," https://subversion.apache.org/.

[6] G. Bergmann, C. Debreceni, I. Ráth, and D. Varró, "Query-based Access Control for Secure Collaborative Modeling using Bidirectional Transformations," in *ACM/IEEE 19th Int. Conf. on MODELS*, 2016.

[7] Z. Diskin, "Algebraic models for bidirectional model synchronization," in *International Conference on Model Driven Engineering Languages and Systems*. Springer, 2008, pp. 21–36.

[8] A. Bagnato, E. Brosse, A. Sadovykh, P. Maló, S. Trujillo, X. Mendialdua, and X. De Carlos, "Flexible and scalable modelling in the mondo project: Industrial case studies." in *XM@ MoDELS*, 2014, pp. 42–51.

[9] D. S. Kolovos, D. Di Ruscio, A. Pierantonio, and R. F. Paige, "Different models for model matching: An analysis of approaches to support model differencing," in *Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models*. IEEE Computer Society, 2009, pp. 1–6.

[10] M. Laakso and A. Kiviniemi, "The IFC standard: A review of history, development, and standardization, information technology," *ITcon*, vol. 17, no. 9, pp. 134–161, 2012.

[11] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, *EMF: eclipse modeling framework*. Pearson Education, 2008.

[12] R. Reddy, R. France, S. Ghosh, F. Fleurey, and B. Baudry, "Model composition-a signature-based approach," in *Aspect Oriented Modeling (AOM) Workshop*, 2005.

[13] Á. Hegedüs, Á. Horváth, I. Ráth, R. R. Starr, and D. Varró, "Query-driven soft traceability links for models," *Software & Systems Modeling*, pp. 1–24, 2014.

[14] G. Bergmann, Z. Ujhelyi, I. Ráth, and D. Varró, "A graph query language for emf models," in *International Conference on Theory and Practice of Model Transformations*. Springer, 2011, pp. 167–182.

[15] C. Brun and A. Pierantonio, "Model differences in the eclipse modeling framework," *UPGRADE, The European Journal for the Informatics Professional*, vol. 9, no. 2, pp. 29–34, 2008.

[16] Z. Xing and E. Stroulia, "Umldiff: an algorithm for object-oriented design differencing," in *Proceedings of the 20th IEEE/ACM Int. Conf. on Automated Soft. Eng.* ACM, 2005, pp. 54–65.

[17] D. S. Kolovos, R. F. Paige, and F. A. Polack, "Model comparison: a foundation for model composition and model transformation testing," in *Proceedings of the 2006 international workshop on Global integrated model management*. ACM, 2006, pp. 13–20.

Activity-Based Abstraction Refinement for Timed Systems

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

***Abstract*—Formal analysis of real time systems is important as they are widely used in safety critical domains. Such systems combine discrete behaviours represented by control states and timed behaviours represented by clock variables. The counterexample-guided abstraction refinement (CEGAR) algorithm utilizes the fundamental technique of abstraction to system verification. We propose a CEGAR-based algorithm for reachability analysis of timed systems. The algorithm is specialized to handle the time related behaviours efficiently by introducing a refinement technique tailored specially to clock variables. The performance of the presented algorithm is demonstrated by runtime measurements on models commonly used for benchmarking such algorithms.**

I. Introduction

Safety critical systems, where failures can result in serious damage, are becoming more and more ubiquitous. Consequently, the importance of using mathematically precise verification techniques during their development is increasing.

Formal verification techniques are able to find design problems from early phases of the development, however, the complexity of safety-critical systems often prevents their successful application. The behaviour of a system is described by the set of states that are reachable during execution (the state space) and formal verification techniques like model checking examine correctness by exploring it explicitly or implicitly. However, the state space can be large or infinite, even for small instances. Thus, selecting appropriate modeling formalisms and efficient verification algorithms is very important. One of the most common formalisms for describing timed systems is the formalism of timed automata that extends finite automata with clock variables to represent the elapse of time.

When applying formal verification, reachability becomes an important aspect – that is, examining whether a given erroneous state is reachable from an initial state. The complexity of the problem is exponential, thus it can rarely be solved for large models. A possible solution to overcome this issue is to use abstraction, which simplifies the problem to be solved by focusing on the relevant information. However, the main difficulty when applying abstraction-based techniques is finding the appropriate precision: if an abstraction is too coarse it may not provide enough information to decide reachability, whereas if it is too fine it may cause complexity problems.

There are several existing approaches in the literature for CEGAR-based verification of timed automata, including [1] where the abstraction is applied on the locations of the automaton, [2] where the abstraction of a timed automaton is an untimed automaton and [3]–[5] where abstraction is applied on the clock variables of the automaton.

Our goal is to develop an efficient model checking algorithm applying the CEGAR-approach to timed systems. The above-mentioned algorithms modified the timed automaton itself to gain a finer state space: our algorithm combines existing approaches with new techniques to create a refinement strategy that increases efficiency by refining the state space directly.

II. Background

A. Timed Automata

Clock variables (*clocks*, for short) are a special type of variables, whose value is constantly and steadily increasing. Naturally, their values can be modified, but the only allowed operation on clock variables is to *reset* them – i.e., to set their value to 0. It's an instantaneous operation, after which the value of the clock will continue to increase.

A *valuation* $v : \mathcal{C} \to \mathbb{R}$ assigns a non-negative real value to each clock variable $c \in \mathcal{C}$, where $\mathcal{C}$ denotes the set of clock variables. In other words a valuation defines the values of the clocks at a given moment of time.

A *clock constraint* is a conjunctive formula of atomic constraints of the form $x \sim n$ or $x-y \sim n$ (*difference constraint*), where $x, y \in \mathcal{C}$ are clock variables, $\sim \in \{\leq, <, =, >, \geq\}$ and $n \in \mathbb{N}$. In other words a clock constraint defines upper and lower bounds on the values of clocks and the differences of clocks. Note, that bounds are always integer numbers. The set of clock constraints are denoted by $\mathcal{B}(\mathcal{C})$.

A *timed automaton* extends a finite automaton with clock variables. It can be defined as a tuple $\mathcal{A} = \langle L, l_0, E, I \rangle$ where

- L is the set of locations (i.e. control states),
- $l_0 \in L$ is the initial location,
- $E \subseteq L \times \mathcal{B}(\mathcal{C}) \times 2^{\mathcal{C}} \times L$ is the set of edges and
- $I : L \to \mathcal{B}(\mathcal{C})$ assigns invariants to locations [6].

The automaton's edges are defined by the source location, the guard (represented by a clock constraint), the set of clocks to reset, and the target location.

A *state* of $\mathcal{A}$ is a pair $\langle l, v \rangle$ where $l \in L$ is a location and v is the current valuation satisfying $I(l)$. In the initial state $\langle l_0, v_0 \rangle$ v_0 assigns 0 to each clock variable.

Two kinds of operations are defined that modify the state of the automaton. The state $\langle l, v \rangle$ has a *discrete transition* to $\langle l', v' \rangle$ if there is an edge $e(l, g, r, l') \in E$ in the automaton

such that v satisfies g, v' assigns 0 to any $c \in r$ and assigns $v(c)$ to any $c \notin r$, and v' satisfies $I(l')$.

The state $\langle l, v\rangle$ has a *time transition* (or *delay*, for short) to $\langle l, v'\rangle$ if v' assigns $v(c) + d$ for some non-negative d to each $c \in \mathcal{C}$ and v' satisfies $I(l)$.

B. Reachability Analysis

In case of timed automata the reachability problem can be defined as follows.

Input: An automaton $\langle L, l_0, E, I\rangle$, and a location $l_{err} \in L$.

Output: An execution trace $\sigma = l_0 \xrightarrow{t_0} l_1 \xrightarrow{t_1} \cdots \xrightarrow{t_n} l_{err}$ from l_0 to l_{err} or *No*, if l_{err} is unreachable.

One of the most efficient algorithms for deciding reachability is the one used by *Uppaal*[1], a model checker for timed automata. The core of the algorithm is published in [6]. Before presenting the approach, some basic definitions are provided.

A *zone* z is a set of non-negative clock valuations satisfying a clock constraint. A *zone graph* is a finite graph consisting of $\langle l, z\rangle$ pairs as nodes, where $l \in L$ refers to some location of a timed automaton and z is a zone. Edges represent transitions.

A node $\langle l, z\rangle$ of a zone graph represents all states $\langle l, v\rangle$ where $v \in z$. Since edges of the zone graph denote transitions, a zone graph can be considered as an (exact) abstraction of the state space. The main idea of the algorithm is to explore the zone graph of the automaton, and if a node $\langle l_{err}, z\rangle$ exists in the graph for some $z \neq \emptyset$, l_{err} is reachable, and the execution trace can be provided by some pathfinding algorithm.

The construction of the graph starts with the initial node $\langle l_0, z_0\rangle$, where l_0 is the initial location and z_0 contains the valuations reachable in the initial location by time transitions. Next, for each outgoing edge e of the initial location (in the automaton) a new node $\langle l, z\rangle$ is created (in the zone graph) with an edge $\langle l_0, z_0\rangle \rightarrow \langle l, z\rangle$, where $\langle l, z\rangle$ contains the states to which the states in $\langle l_0, z_0\rangle$ have a discrete transition through e. Afterwards z is replaced by $z^{\uparrow}$ where $\langle l, z^{\uparrow}\rangle$ represents the set of all states reachable from a zone $\langle l, z\rangle$ by time transitions. The procedure is repeated on every node of the zone graph. If the states defined by a new node $\langle l, z\rangle$ are all contained in an already existing node $\langle l, z'\rangle$ ($z \subseteq z'$), $\langle l, z\rangle$ can be removed, and the incoming edge can be redirected to $\langle l, z'\rangle$.

Unfortunately, it is possible that the described graph becomes infinite. In order to prevent this, [6] introduces an operation called *normalization* to apply on $z^{\uparrow}$ before inclusion is checked. Let $k(c)$ denote the greatest value to which clock c is compared in the automaton. This operation overapproximates the zone treating the interval $(k(c), \infty)$ as one, abstract value for each $c \in \mathcal{C}$, since for any valuation v such that $v(c) > k(c)$ constraints of the form $c > n$ are satisfied, and constraints of the form $c = n$ or $c < n$ are unsatisfied.

Using normalization the zone graph is finite, and if there are no difference constraints in the automaton, reachability will be decided correctly, however, in case of difference constraints the algorithm may terminate with a false positive result.

[1] http://www.uppaal.org/

The operation *split* [6] is introduced to assure correctness. Instead of normalizing the complete zone, it is first split along the difference constraints, then each subzone is normalized, and finally the initially satisfied constraints are reapplied to each normalized subzone. The result is a set of zones (not just one zone like before), which means multiple new nodes have to be created in the zone graph (with edges from the original node). Applying split results in a zone graph, that is a correct and finite representation of the state space [6].

Implementation is also provided in [6]. The zones are stored in an $n \times n$ matrix form (the so-called *Difference Bound Matrix*, DBM), where $n = |\mathcal{C}| + 1$, and each row and column represents a clock, except for the first ones that represent the constant 0. An entry $D[i, j] = (m, \prec)$, where $m \in \mathbb{Z} \cup \{\infty\}, \prec \in \{<, \leq\}$ of the DBM D represents the constraint $c_i - c_j \prec m$, where $c_0 = 0$. (It is proven, that all atomic clock constraints can be transformed to this form.) Each entry of a DBM represents the strongest bound that can be derived from the constraints defining the zone.

Pseudocodes are also provided for operations, such as *add()* (adds an atomic constraint to the zone), *reset()* (resets the given clock), *up()* (calculates $z^{\uparrow}$), *norm()* and *split()* to calculate successor states automatically, as well as some additional operations, such as *free()* (removes all constraints on a clock).

C. Activity

The (exact) *activity* abstraction is proposed in [7] to reduce the number of clock variables without affecting the state space. A clock c is considered *active* at some location l (denoted by $c \in Act(l)$) if its value at l may influence the future operation of the system. It might be because c appears in the $I(l)$, or in the guard g of some outgoing edge (l, g, r, l'), or because $c \in Act(l')$ for some l' reachable from l without resetting c.

If $Act(l) < |\mathcal{C}|$ holds for each $l \in L$, the number of clock variables can be reduced by reconstructing the automaton, by removing all $c \notin Act(l)$ and *renaming* $c \in Act(l)$ for each $l \in L$ such that after renaming less clocks remain. This is possible, even if all $c \in \mathcal{C}$ is active in at least one location, since clocks can be renamed differently in distinct locations.

Before presenting how activity is calculated some new notations are introduced. Let $clk : \mathcal{B}(\mathcal{C}) \rightarrow 2^{\mathcal{C}}$ assign to each clock constraint the set of clocks appearing in it. Define $clk : L \rightarrow 2^{\mathcal{C}}$ such that $c \in clk(l)$ iff $c \in clk(I(l))$ or there exist an edge (l, g, r, l') such that $c \in clk(g)$.

Activity is calculated by an iterative algorithm starting from $Act_0(l) = clk(l)$ for each $l \in L$. In the i^{th} iteration $Act_i(l)$ is derived by extending $Act_{i-1}(l)$ by $Act_{i-1}(l') \setminus r$ for each edge (l, g, r, l'). The algorithm terminates when it reaches a fix point, i.e. when $Act_i(l) = Act_{i-1}(l)$ for each $l \in L$.

D. CEGAR

In order to increase the efficiency of model checking, (approximate) abstraction can be used [8]: a less detailed system model is constructed with a state space overapproximating that of the original one, model checking is applied to this simple model, and if the erroneous state is unreachable in the abstract

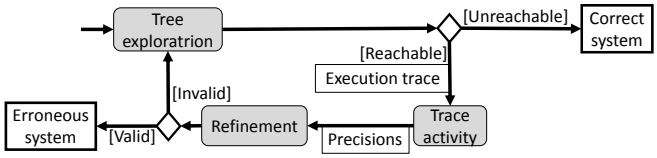


Fig. 1. Algorithm overview

model, the system is correct. Otherwise the model checker produces an abstract counterexample that is examined on the original system, and if it is feasible, the system is incorrect. If it is invalid, the abstraction is too coarse to decide reachability.

Counterexample-guided abstraction refinement (CEGAR) [9] extends this approach into an iterative algorithm, by refining the abstract state space in order to eliminate the invalid counterexample. Model checking is applied on the refined state space (that is still an abstraction of the original one) and the so-called CEGAR-loop starts over.

III. Activity-Based Abstraction Algorithm

The main idea of our new algorithm is to explore the state space without considering clock variables and refining it (calculating zones) trace by trace, based on the discovered counterexamples. Figure 1 depicts the basic operation of the algorithm. Note, that the phases of this algorithm correspond to the phases of CEGAR.

To increase efficiency not all clock variables are included – the relevant clocks for each node in the path (the *precision*) are chosen by an algorithm we have developed based on the one described in Section II-C. To avoid confusion, zones will appear with their precision denoted, e.g. z_C denotes a zone z of precision $C \subseteq \mathcal{C}$.

A. Data structure

In our algorithm the formalism that represents the abstract state space can be defined as a tuple $\langle N_e, N_u, E^{\uparrow}, E^{\downarrow} \rangle$ where

- $N_e \subseteq L \times \mathcal{B}(\mathcal{C})$ is the set of explored nodes,
- $N_u \subseteq L \times \mathcal{B}(\mathcal{C})$ is the set of unexplored nodes,
- $E^{\uparrow} \subseteq (N_e \times N)$, where $N = N_e \cup N_u$ is the set of upward edges and
- $E^{\downarrow} \subseteq (N_e \times N)$ is the set of downward edges.

The sets N_e and N_u as well as the sets $E^{\uparrow}$ and $E^{\downarrow}$ are disjoint. $T^{\downarrow} = (N, E^{\downarrow})$ is a tree.

Nodes are built from a location and a zone and (downward) edges represent transitions like in the zone graph but in this case nodes are distinguished by the trace through which they are reachable. This means the graph can contain multiple nodes with the same zone and the same location, if the represented states can be reached through different traces.

The root of T is the initial node. Downward edges have similar roles to edges of the zone graph, while upward edges are used to avoid exploring the same states multiple times. An upward edge from a node n to a previously explored node n' means that the states represented by n are a subset of the states represented by n', thus it is unnecessary to keep searching for a counterexample from n, because if there exists one, another one will exist from n'. Searching for new traces is only continued on nodes without an outgoing upward edge. This way, the graph can be kept finite.

Initially, the graph contains only one, unexplored node $n_0 = \langle l_0, z_\emptyset \rangle$, and as the state space is explored, unexplored nodes become explored nodes, new unexplored nodes and edges appear, until a counterexample is found, or there are no remaining unexplored nodes. During the refinement phase zones are calculated, new nodes and edges appear and complete subtrees disappear. State space exploration will then be continued from the unexplored nodes, and so on.

B. State space exploration

State space exploration is performed in the following way. In each iteration a node $n = \langle l, z_C \rangle \in N_u$ is chosen. First, it is checked if the states n represents are included in some other node $n' = \langle l, z'_{C'} \rangle$ where $C = C'$. In this case an upward edge $n \to n'$ is introduced and n becomes explored. Otherwise, n has yet to be explored. For each outgoing edge $e(l, g, r, l')$ of l in the automaton a new node $\langle l', z_\emptyset \rangle \in N_u$ is introduced with an edge pointing to it from n, which becomes explored. If any of the new nodes contains l_{err}, the state space exploration phase terminates and the proposed counterexample $\sigma = n_0 \xrightarrow{t_0} n_1 \xrightarrow{t_1} \cdots \xrightarrow{t_n} n_{err} = \langle l_{err}, z_\emptyset \rangle$ is the trace reaching n_{err} in $T^{\downarrow}$. Otherwise, another $n \in N_u$ is chosen, and so on.

If the state space is explored and l_{err} does not appear in it, the erroneous states are unreachable, and the system is correct.

C. Trace activity

After finding a possible counterexample the next task is to calculate the necessary precisions. The presented algorithm is a modified version of the one described in Section II-C.

Based on *activity* we introduce a new abstraction $Act_\sigma(n)$, called *trace activity* which assigns precisions to nodes on the trace (instead of locations of the automaton), that only include the clocks whose value affects the reachable states of the trace.

Trace activity is calculated iterating backwards on the trace. In the final node n_{err} the valuations are not relevant, as the only question is whether it is reachable – $Act_\sigma(n_{err}) = \emptyset$. For $n_i \neq n_{err}$, $Act_\sigma(n_i)$ can be calculated from $Act_\sigma(n_{i+1})$ and the edge $e_i(l_i, g_i, r_i, l_{i+1})$ used by transition t_i. Since all $c \in r_i$ are reset, their previous values will have no effect on the system's future behaviour – they can be excluded. It is necessary to know if t_i is enabled, so $clk(g_i)$ must be active, as well as $clk(I(l_i))$ since $I(l_i)$ have to be satisfied. This gives us the formula $Act_\sigma(n_i) = (Act_\sigma(n_{i+1}) \backslash r_i) \cup clk(g_i) \cup clk(I(l_i))$.

D. Refinement

The task of the refinement phase is to assign correct zones of the given precision for each node in the trace and to decide if the counterexample is feasible. It is important to mention that the zones on the trace may already be refined to some precision C' that is independent from the new precision C. In this case the zone has to be refined to the precision $C \cup C'$.

Refinement starts from the initial zone z_0 that can be refined to $z_{C_0} = \bigwedge_{c_i,c_j \in C_0} c_i = c_j$, where C_0 is the required precision. After that z_{C_i} of node n_i on the trace can be calculated from $z_{C_{i-1}}$ of node n_{i-1} with the operations mentioned in Section II-B, with some modifications to handle the precision change.

First, the guard has to be checked. If there are no states in $z_{C_{i-1}}$ that satisfy g_{i-1}, the counterexample is invalid, and the abstract state space has to be refined: since t_{i-1} is not enabled, the corresponding edge, and the belonging subtree has to be removed from the graph, and the algorithm can continue by searching for another counterexample.

Next, the clocks in r_{i-1} are reset. This can be performed using operation *reset()*. Change of precision is also applied at this point. Assume that the precision of the source zone is C_i and the target zone has to be refined to precision C_{i+1}.

Variables $C_{old} = C_i \setminus C_{i+1}$ have to be excluded before executing the transition. Consider the DBM implementation of zones. Excluding the unnecessary clocks from the zone can be performed by *free(c)* for each $c \in C_{old}$, but according to the pseudocode in [6] the operation only affects the row and the column belonging to c. Thus, for space saving purposes, the row and column of c can simply be deleted from the DBM.

Variables $C_{new} = C_{i+1} \setminus C_i$ have to be introduced. *Trace activity* guarantees that clocks are only introduced when they are reset, thus, it can be performed by adding a new row and column to the DBM, that belong to c and calling *reset(c)*.

The next step is to apply the invariant. If this results in an empty zone, the transition is not enabled – the subtree has to be deleted, and the algorithm continues by searching for another counterexample. Otherwise, *up(), split()*, and *norm()* has to be applied to calculate the precise zone (or zones).

The node n_i can be refined by replacing the current zone with the calculated one, however, the incoming upward edges have to be considered first. An edge $n \to n_i \in E^{\uparrow}$ means n_i represents all states that n represents – this may not be true after the refinement. Thus, the upcoming edges are removed and their sources are marked *unexplored*.

It is important to consider that sometimes the *split()* operation results in more than one zones. Similarly to the case of the zone graph, this can be handled by replicating the node. Refinement has to be continued from each new node, thus the refinement of a trace may introduce new subtrees. The tree structure allows this, however, it is important to mark the new nodes *unexplored*, since only the outgoing edges representing the transition on the trace are created, the other possible outgoing edges have yet to be explored.

IV. Evaluation

We evaluated the performance of the presented algorithm with measurements. The inputs are scalable automata chosen from Uppaal's benchmark data[2] that is widely used for comparing the efficiency of such algorithms. Network automata with discrete variables were unfolded to timed automata before the measurements. The results are depicted in Table I.

[2]https://www.it.uu.se/research/group/darts/uppaal/benchmarks/

TABLE I
Measurement results (ms)

CSMA2	CSMA3	CSMA4	Fisch2	Fisch3
264	1 113.5	9 808	292	5 650
Token8	Token32	Token128	Token512	Token2048
838	2 173	4 966	12 580	100 892

The models are denoted by CSMAn, Fisch$n$, and Token$n$ for the CSMA/CD, Fischer and Token ring/FDDI protocols of n participants, respectively. The Token ring protocol is a special input, since the examined safety property can be proven solely based on the structure of the automaton, thus the analysis of the initial abstraction is able to prove the property. This proves how useful abstraction is, but the measurements on this automaton can only demonstrate the efficiency of the pathfinding algorithm, which turned out to be $\mathcal{O}(n^2)$. Memory problems occurred at the Fischer protocol of four processes and the CSMA/CD protocol of five stations. For smaller instances the algorithm always terminated with the expected result.

V. Conclusions

This paper provided a CEGAR-based algorithm for reachability analysis of timed automata, that applies abstraction on the zone graph, and calculates the required precision for the refinement using *trace activity*. The efficiency of the algorithm was demonstrated by measurements. Results suggest that the pathfinding algorithm is efficient, but the memory usage has yet to improve.

References

[1] S. Kemper and A. Platzer, "SAT-based abstraction refinement for real-time systems," *Electronic Notes in Theoretical Computer Science*, vol. 182, pp. 107–122, 2007.

[2] T. Nagaoka, K. Okano, and S. Kusumoto, "An abstraction refinement technique for timed automata based on counterexample-guided abstraction refinement loop," *IEICE Transactions*, vol. 93-D, no. 5, pp. 994–1005, 2010.

[3] H. Dierks, S. Kupferschmid, and K. G. Larsen, "Automatic abstraction refinement for timed automata," in *Formal Modeling and Analysis of Timed Systems, FORMATS'07*, ser. LNCS. Springer, 2007, vol. 4763, pp. 114–129.

[4] F. He, H. Zhu, W. N. N. Hung, X. Song, and M. Gu, "Compositional abstraction refinement for timed systems," in *Theoretical Aspects of Software Engineering*. IEEE Computer Society, 2010, pp. 168–176.

[5] K. Okano, B. Bordbar, and T. Nagaoka, "Clock number reduction abstraction on CEGAR loop approach to timed automaton," in *Second International Conference on Networking and Computing, ICNC 2011*. IEEE Computer Society, 2011, pp. 235–241.

[6] J. Bengtsson and W. Yi, "Timed automata: Semantics, algorithms and tools," in *Lectures on Concurrency and Petri Nets*, ser. LNCS. Springer Berlin Heidelberg, 2004, vol. 3098, pp. 87–124.

[7] C. Daws and S. Yovine, "Reducing the number of clock variables of timed automata," in *Proceedings of the 17th IEEE Real-Time Systems Symposium (RSS '96*. Washington - Brussels - Tokyo: IEEE, Dec. 1996, pp. 73–81.

[8] E. M. Clarke, O. Grumberg, and D. E. Long, "Model checking and abstraction," *ACM Transactions on Programming Languages and Systems*, vol. 16, no. 5, pp. 1512–1542, Sep. 1994.

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

Formal Compositional Semantics for Yakindu Statecharts

Bence Graics, Vince Molnár
Budapest University of Technology and Economics,
Department of Measurement and Information Systems
Budapest, Hungary
Email: bence.graics@inf.mit.bme.hu, molnarv@mit.bme.hu

***Abstract*—Many of today's safety-critical systems are reactive, embedded systems. Their internal behavior is usually represented by state-based models. Furthermore, as the tasks carried out by such systems are getting more and more complex, there is a strong need for compositional modeling languages. Such modeling formalisms start from the component-level and use composition to build the system-level model as a collection of simple modules. There are a number of solutions supporting the model-based development of safety-critical embedded systems. One of the popular open-source tools is Yakindu, a statechart editor with a rich language and code generation capabilities. However, Yakindu so far lacks support for compositional modeling. This paper proposes a formal compositional language tailored to the semantics of Yakindu statecharts. We propose precise semantics for the composition to facilitate formal analysis and precise code generation. Based on the formal basis laid out here, we plan to build a complete tool-chain for the design and verification of component-based reactive systems.**

I. Introduction

Statechart [1] is a widely used formalism to design complex and hierarchical reactive systems. Among the many statechart tools, our work is based on the open-source Yakindu[1], which supports the development of complex hierarchical statecharts with a graphical editor, validation and simulation features. Yakindu also supports source code-generation from statecharts to various languages (Java, C, C++).

The requirements embedded systems have to meet are getting more and more complex. Therefore, the models created for such systems tend to become unmanageably large, which encumbers extensibility and maintenance. Instead, the resulting models could be created by composing smaller units. These units interact with each other using the specified connections, thus implementing the original behavior. There are several tools that aim to support this methodology.

SysML [2], [3] tools have a large set of modeling elements which enables their users to express their thoughts and ideas as freely and concisely as possible. On the other hand, they rarely define precise semantics, which encumbers code generation and analysis. BIP [4]–[6] is a compositional tool with well-defined semantics that supports the formal verification of modeled systems. Source code generation is also possible with this tool. Scade[2] [7], [8] is a tool that unifies the advantages of design and analysis tools. It supports the generation of source code as well as the formal verification of the modeled system. It is a commercial tool and does not support extensibility. Matlab Stateflow [9] is an environment for modeling and simulating decision logic using statecharts and flow charts. It is a leading tool for composing state-based models in the domain of safety-critical embedded systems. It supports the encapsulation of state-based logics which can be reused throughout different models and diagrams.

Unfortunately, Yakindu does not support composition features. The main goal of our work is to create a tool that enables the users to compose individual statechart components into a single composite system by constructing connections through ports. The ultimate goal of this work is to enable code generation and formal verification of composite models with model transformations based on the proposed semantics.

We will call this type of composition an event-based automata network, as opposed to dataflow networks, which can be considered message-based automata networks in this sense. In event-based automata networks, data is only of secondary importance – the occurrence of the event is in focus. In message-based settings, data is more significant, thus message queues are desirable to buffer the simultaneous messages.

This paper is structured as follows. Section II presents the semantics of Yakindu statecharts serving as the basis of the compositional language. The syntax and semantics of the compositional language along with an example are introduced in Section III. Finally, Section IV provides concluding remarks and ideas for future work.

II. Abstracting Yakindu Statecharts

Yakindu adopts a statechart formalism which is the extension of the well-known state machine formalism. Statecharts support the definition of auxiliary variables as well as concurrency and state refinement. This section introduces a syntactical abstraction of Yakindu, i.e. the actual model elements are ignored. We deal only with the input and output events in addition to the actual state configuration, but not the semantics. This way we can generalize the composition

[1]https://itemis.com/en/yakindu/statechart-tools/

[2]http://www.esterel-technologies.com/products/scade-suite/

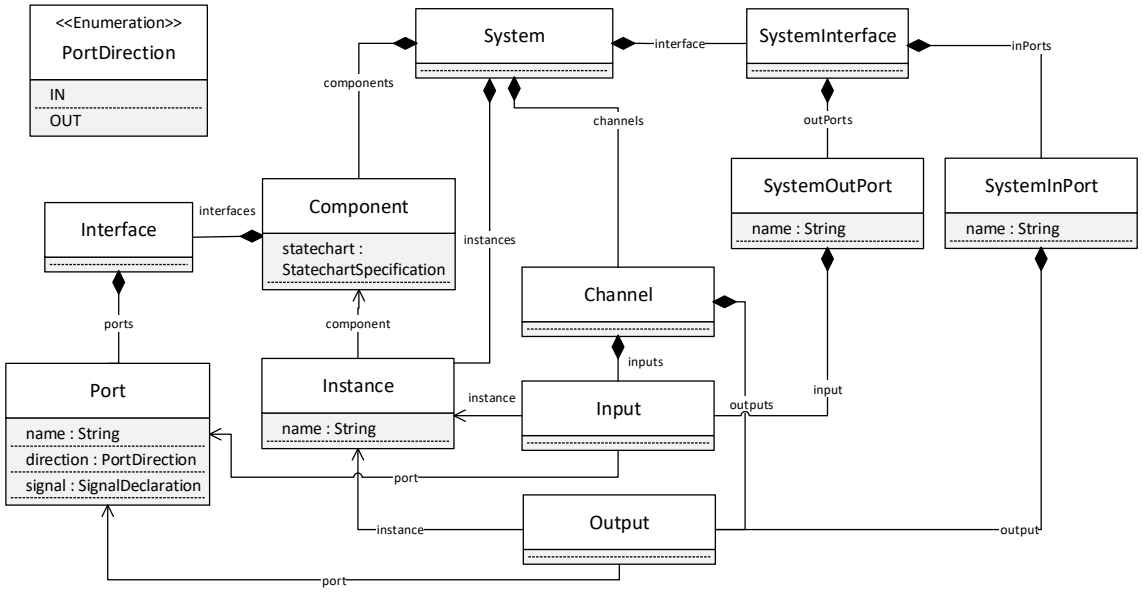


Fig. 1. Metamodel of the compositional language.

of abstract models with minimal restrictions to the usable formalisms. In this approach, a Yakindu statechart is considered a 5-tuple: $\mathbb{S} = \langle I, O, S, s_0, T \rangle$ where:

- I is a finite set of input events (from the environment)
- O is a finite set of output events (for the environment)
- $S = \{s_1, s_2, \cdots, s_n\}$ is a finite set of states, including a state configuration and values of variables
- $s_0 \in S$ is the initial state
- $T \subseteq (2^I \times S) \times (S \times 2^O)$ is a finite set of transitions, that represent changes of state in response to a set of input events and generate a set of output events

Yakindu statecharts adopt a turn-based semantics. The following paragraphs introduce the interpretation of *turns* as well as how the raising of events is associated to them.

Events represent signal receptions. There are two types of events: *simple* or *void* events and *parameterized* or *typed* events. The latter enables the modeling of parameterized signals, which can describe additional details. Note that multiple input events can be raised in a single turn according to the abstract formalism defined above. In Yakindu the raising of events is interpreted as setting a boolean flag to true. Yakindu therefore does not support message queues. Owing to this semantics raising the same *simple* event in a particular turn once or several times has the same effect. On the other hand, *parameterized* events are defined by their parameter as well, so a new event raising with a different parameter overwrites the former one. Although this behavior is an essential part of the semantics of Yakindu, it is not relevant either in the abstract formalism presented above or the semantics of the composition language defined in Section III-B.

All turns consist of two distinct sections, a *raising* section and a *running* section. In the raising section input events of the statechart are raised as presented in the previous paragraph. This is followed by the running section where a new stable state of the statechart is defined. It starts with the examination of the transitions going out of the particular state configuration. The goal is to specify the firing transition. At this point a *race condition* might exist if multiple outgoing transitions are enabled, e.g. more of them are triggered by raised input events. Yakindu intends to solve ambiguity by introducing the concept of *transition priority*: users can specify which of the outgoing transitions of a state should be fired in case of a race condition by defining a total ordering of the transitions. The firing transition specifies the next stable state of the statechart, including the state configuration, values of variables and events for the environment.

III. Language for Composition

This section defines the syntax of the compositional language and introduces the semantics the composite system conforms to. This semantics is heavily influenced by the statechart semantics defined by Yakindu and strives to address some of its problems, e.g. gives the ability to parallel regions to communicate with each other.

A. Syntax

Figure 1 depicts the metamodel of the compositional language. The root element in the metamodel is the *System*. A *System* contains *Components* which refer to Yakindu statecharts as well as *Instances* of such *Components*. Each *Component* has an *Interface* that contains *Ports*. Through *Ports*, signals of statecharts can be transmitted or received according to their directions.

Channels can be used for defining the emergent behavior of the composite system. A *Channel* has one or more *Inputs* and one or more *Outputs*. An Input of a *Channel* connects to an output *Port* of an *Instance* and vice versa. Whenever a

Channel receives a signal through any of its *Inputs*, the signal is sent to each *Output*, i.e. to the corresponding input *Ports* of *Instances*. The language does not support connecting *Ports* of the same direction and a validation rule is defined that marks incorrect connections.

The language supports the definition of an interface through which the composite system interacts with its environment. This is the *SystemInterface* that contains *SystemPorts*. *SystemPorts* are aliases of *Ports* of *Instances*. If a signal arrives to a *SystemInPort*, it will be forwarded to the *Port* of the referred *Instance* instantly. *SystemOutPorts* work similarly, but with output *Ports* of *Instances*.

For ease of understanding, an example is presented that defines a composition of statecharts using the specified compositional language. The system consists of two *Components*, *CoffeMachineComponent* and *LightComponent* referring to a coffee machine (CoffeMachine) statechart and a light switch (LightSwitch) statechart, respectively. CoffeMachine has signal declarations for turning it on and off, for ordering a cappuchino and for putting its light on and off. A LightSwitch models a lamp that can be turned on and off.

```
// System interface definition
interface {
    in {
        on : machine.on
        off : machine.off
        cappuchino : machine.cappuchino
    }
}

// Component interface definitions
CoffeeMachine CoffeMachineComponent {
    interface {
        on : IN on
        off : IN off
        cappuchino : IN cappuchino
        lightOn : OUT flashLight
        lightOff : OUT turnOffLight
    }
}

LightSwitch LightComponent {
    interface {
        on : IN onButton
        off : IN offButton
    }
}

// Component instantiations
CoffeMachineComponent machine
LightComponent light

// Channel definitions
channels {
    [machine.lightOn] -> [light.on]
    [machine.lightOff] -> [light.off]
}
```

Note that a composite system description constists of the following parts:

- System interface definition: All input *Ports* of *machine* are published to the interface of the system enabling the users to turn *machine* on and off or order a cappuchino.
- Component interface definitions: *CoffeMachineComponent* refers to *on*, *off* and *cappuchino* through input *Ports* (denoted by the IN keyword) and *flashLight*, *turnOffLight* through output *Ports* (denoted by the OUT keyword). Both signal declarations of LightSwitch are referred to by input *Ports*.
- Component instantiations: Both *Components* are instantiated: *machine* and *light*.
- Channel definitions: The output *Ports* of *machine* are connected to the input *Ports* of *light*, making it possible for *machine* to turn on *light* at choice.

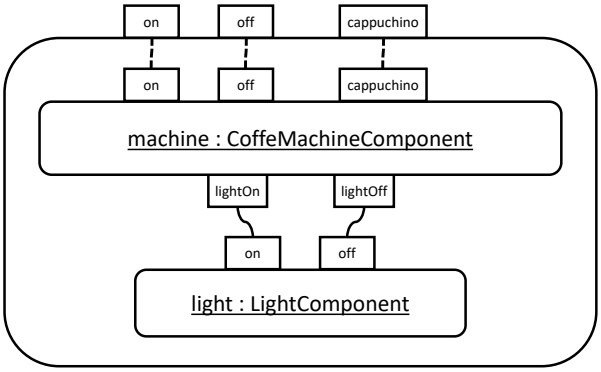


Fig. 2. A composite system of a CoffeMachine and a LightSwitch statechart.

Figure 2 depicts the composite system described by the previous code section. Note that the individual components of the system are encapsulated. Interactions can be specified only through the defined interface.

B. Semantics

During the design of the semantics one of our goal was to define a language that enables the reuse of the source code generator of Yakindu. Therefore the semantics of supported Yakindu statecharts elements had to be considered, most importantly event raising.

This section introduces the semantics of the compositional language. The compositional language enables to create a *composite system*, that is formally a 4-tuple: $C = \langle SC, CA, IN, OUT \rangle$ where:

- $SC = \{\langle S_1, s_1^0, T_1, I_1, O_1 \rangle, \cdots, \langle S_n, s_n^0, T_n, I_n, O_n \rangle\}$ is a finite set of state machines.
- $I = \bigsqcup_{j=1}^{n} I_j$, i.e. the union of all in events of state machine components
- $O = \bigsqcup_{j=1}^{n} O_j$, i.e. the union of all out events of state machine components
- $CA \subseteq 2^O \times 2^I$, i.e. channel associations relate a finite set of outputs to a finite set of inputs
- $IN \subseteq I$, i.e. the input interface is a subset of the union of the in events of state machine components
- $OUT \subseteq O$, i.e. the output interface is a subset of the union of the out events of state machine components

A sequence of steps $\varrho = (\tau_1, \tau_2, \cdots)$ is called a *complete run* of C if the following conditions hold.

- $\tau_j = (\underline{s}_j, i_j, \underline{s}'_j, \underline{o}_j)$ is a single step that consists of a state vector representing each state of each component before the step, a finite set of inputs, a state vector representing each state of each component after the step and a finite set of outputs generated by each state machine components, where for all $1 \leq k \leq n$ at least one of the following conditions holds:
  - $(i_j \cap I_k, \underline{s}_j[k], \underline{s}'_j[k], \underline{o}_j[k]) \in T_k$, i.e. if a transition is defined in a state machine component that is triggered by the input set, then the transition fires taking the state machine to its target state and producing the corresponding outputs;
  - $(\underline{s}_j[k] = \underline{s}'_j[k] \wedge \underline{o}_j[k] = \emptyset \wedge \nexists s', o' : (i_j \cap I_k, \underline{s}_j[k], s', o') \in T_j)$, i.e. a component is allowed to do nothing if and only if it has no transition that is triggered by input i_j in state $\underline{s}_j[k]$;
- $\underline{s}_1 = (s_1^0, s_2^0, \cdots, s_n^0,)$, i.e. at the beginning of the run, all state machine components are in their initial states;
- $\underline{s}'_j = \underline{s}_{j+1}$, i.e. the state vector at the end of a step and at the beginning of the next step are equal;
- $tgd(\bigcup_{k=1}^{n} \underline{o}_j[k]) \subseteq i_{j+1} \subseteq tgd(\bigcup_{k=1}^{n} \underline{o}_j[k]) \cup IN$ where $tgd(\Omega) = \bigcup_{\omega \in 2^{\Omega}} \omega \circ CA$, i.e. the inputs of a step is at least the inputs triggered through a channel by outputs of the previous step and maybe some additional events of the input interface;
- ϱ is either infinite or the following condition holds:
  - $\nexists (o, i) \in CA \colon o \cap o_n \neq \emptyset$, i.e. the execution of steps can terminate only if the last step does not produce any outputs that will be inputs in the next step.

A partial run of a composite system can be any prefix of a complete run (any other sequence is not considered to be a behavior of the composite system).

It is important to note that message queues (buffering) are not included, the semantics guarantees only that event raising and event receptions are in a causal relationship. Therefore, if a component does not buffer events (such as Yakindu), parameterized events may overwrite each other.

The operational semantics presented above provides a way to reduce the semantics of the composite system to the semantics of the components. To formally analyze the system, denotational semantics has to be provided, e.g. by model transformations converting the composite system model into a formal model, in accordance with the operational semantics.

IV. Conclusions and Future Work

Yakindu is a popular open-source tool for the design of statechart models with support for code generation. It has a rich language to model a single hierarchical statechart, but it lacks the ability to compose statecharts into a component-based model. For the design of complex, embedded reactive systems, compositionality is essential to handle the design complexity. Moreover, a precise formal semantics is necessary to facilitate code generation and formal analysis.

The defined compositional language enables to instantiate Yakindu statecharts, specify ports for these instances and join these instances through port connections. The semantics of the language is well-defined and suits the statechart semantics of Yakindu soundly.

Subject to future work, we plan to extend the compositional language to allow hierarchical compositions, i.e. the composition of composite systems. Additionally, we intend to design a whole framework around the language that *1)* enables the generation of source code which connects the Yakindu statecharts according to the defined semantics and *2)* provides automated model transformation to formal models of composite systems on which exhaustive analysis can be performed by model-checkers.

The automatic model transformers will utilize a graph-pattern-based approach to generate the traceability information that will facilitate the back-annotation of the results of formal analysis to the engineering domain. This way, we hope to support formal verification without requiring the designers to get familiar with the formal languages involved.

Acknowledgment

This work was partially supported by IncQuery Labs Ltd. and MTA-BME Lendület Research Group on Cyber-Physical Systems.

References

[1] D. Harel, "Statecharts: A visual formalism for complex systems," *Sci. Comput. Program.*, vol. 8, no. 3, pp. 231–274, Jun. 1987. [Online]. Available: http://dx.doi.org/10.1016/0167-6423(87)90035-9

[2] OMG, *OMG Systems Modeling Language (OMG SysML), Version 1.3*, Object Management Group Std., 2012. [Online]. Available: http://www.omg.org/spec/SysML/1.3/

[3] L. Delligatti, *SysML Distilled: A Brief Guide to the Systems Modeling Language*, 1st ed. Addison-Wesley Professional, 2013.

[4] A. Basu, M. Bozga, and J. Sifakis, "Modeling heterogeneous real-time components in BIP," in *Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods*, ser. SEFM '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 3–12. [Online]. Available: http://dx.doi.org/10.1109/SEFM.2006.27

[5] I. Konnov, T. Kotek, Q. Wang, H. Veith, S. Bliudze, and J. Sifakis, "Parameterized Systems in BIP: Design and Model Checking," in *27th International Conference on Concurrency Theory (CONCUR 2016)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), J. Desharnais and R. Jagadeesan, Eds., vol. 59. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016, pp. 30:1–30:16. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2016/6167

[6] M. D. Bozga, V. Sfyrla, and J. Sifakis, "Modeling synchronous systems in BIP," in *Proceedings of the Seventh ACM International Conference on Embedded Software*, ser. EMSOFT '09. New York, NY, USA: ACM, 2009, pp. 77–86. [Online]. Available: http://doi.acm.org/10.1145/1629335.1629347

[7] H. Basold, M. Huhn, H. Günther, and S. Milius, "An open alternative for SMT-based verification of SCADE models," in *Proc. 19th International Workshop on Formal Methods for Industrial Critical Systems (FMICS'14)*, ser. Lecture Notes Comput. Sci., F. Lang and F. Flammini, Eds., vol. 8718. Springer, 2014, pp. 124–139.

[8] R. Venky, S. Ulka, A. Kulkarni, and P. Bokil, "Statemate to scade model translation," in *ISEC '08: Proceedings of the 1st conference on India software engineering conference*. New York, NY, USA: ACM, 2008, pp. 145–146. [Online]. Available: http://dx.doi.org/10.1145/1342211.1342245

[9] J. Chen, T. R. Dean, and M. H. Alalfi, "Clone detection in matlab stateflow models," *Software Quality Journal*, vol. 24, no. 4, pp. 917–946, 2016. [Online]. Available: http://dx.doi.org/10.1007/s11219-015-9296-0

Towards Model-Based Support for Regression Testing

Anna Gujgiczer, Márton Elekes, Oszkár Semeráth, András Vörös
Budapest University of Technology and Economics
Department of Measurement and Information Systems
Budapest, Hungary
Email: gujgiczer.anna@gmail.com, marci543@gmail.com, semerath@mit.bme.hu, vori@mit.bme.hu

***Abstract*—Software is a continuously evolving product: modifications appear frequently to follow the changing requirements or to correct errors. However, these modifications might introduce additional errors. Regression testing is a method to verify the modified system, whether the development introduces new problems into the system or not. Regression testing involves the execution of numerous tests, usually written manually by the developers. However, construction and maintenance of the test suite requires huge effort. Many techniques exist to reduce the testing efforts. In this paper we introduce a model-based approach to reduce the number of regression tests by using abstraction techniques and focusing on the changing properties of the unit under test.**

I. Introduction

The development of complex, distributed and safety-critical systems yield a huge challenge to system engineers. Ensuring the correct behavior is especially difficult in evolving environments, where the frequent changes in demands lead to frequent redesign of the systems. This rapid evolution raises many problems: the new version of the system has to be verified in order to detect if it still fulfills the specification, i.e. the developments do not introduce additional problems or undesired modifications in the existing functionality. Regression testing is selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements. [5] It uncovers newly introduced software bugs, or regressions. Regression testing can determine whether a change in one part of the software affects other parts or functionalities.

Supporting regression testing is an important though difficult task. Creating a model for the desired behavior of the system could significantly help regression testing and would enable the application of model-based testing approaches. However developers usually do not have time and effort to create the specification model during the development and it is costly to construct the model afterwards from the source code and configuration files.

In this paper we introduce a model-based approach to support the regression testing of software components. We developed a methodology to automatically synthesize behavior models by using automata learning algorithms. However, traditional automata learning algorithms proved to be insufficient for this task, as they are unable to handle the sheer complexity of existing software components. Therefore some kind of abstraction framework is required to simplify the observable behavior of the unit under learning: these improvements can support the construction of a behavior model of even complex software components, which the state-of-the-art learning algorithm fails to learn. In our approach we use a feature model based abstraction on the interface of the software component. Based on the learned models we automatically generate a set of test sequences. Additionally, comparing the behavior models of the different versions of the software components is able to highlight unwanted changes. We also implemented the proposed approach in a prototype framework to prove its feasibility.

The structure of the paper: first of all, in Section II we introduce the required preliminaries and an example to guide the reader through the paper, later, Section III recommends a general approach for regression testing. Section IV introduces language support for defining the relevant behavior of the analysed software component. Section V shows preliminary measurements. Finally, Section VI concludes the paper.

II. Preliminaries

A. Example

A chess clock software used at the System Modeling Course [2] at BME serves as a motivating example. This chess clock has two main functionalities, let call them *menu* and *game* function. The game function enables the switch of the active player and descending its remaining time. In the menu the players can configure the settings of the game. The input interface of this chess clock contains four buttons (Start, Mode, White and Black) and the output interface consists of three displays (Main, White time, Black time).

B. Feature Model

Feature models [6] are widely used in the literature to provide a compact representation for software product lines. A feature model contains features in a tree structure, which represents the dependencies between the features.

The possible relations between a feature and its child- or subfeatures are categorized as:

- *Mandatory*: in this case the child feature is required.
- *Optional*: in this case the child feature is optional.

- *Or*: at least one of the subfeatures must be active, i.e. contained in the input or output, if the parent is contained in the message.
- *Alternative (Xor)*: exactly one subfeature must be selected.

Beside that, cross-tree constraints can be represented by additional edges.

A feature model configuration is a concrete set of features from a feature model which satisfies the constraints expressed by the relations of the feature model: a valid configuration does not violate any parent-child dependency or any cross-tree constraint.

C. Automata Learning

Automata learning is a method for producing the automaton-based behavior model of a unit by observing the interactions, i.e. inputs and outputs with its environment. There are two main types of automata learning, active and passive learning. An active automata learning algorithm was chosen in our work, as it can produce more accurate behavior models.

In active automata learning [1], [9], models of a Unit Under Learning (UUL) are created through active interaction, i.e. driving the UUL and by observing the output behavior. For this procedure the algorithm needs to be able to interact with the target unit in several ways, such as:

- reset the UUL,
- execute action on the UUL, i.e. drive it with an input,
- observe the outputs of the UUL.

Given the possible input and output alphabets of the software component the algorithm learns by constructing queries composed of input symbols from the alphabet, then these queries are asked from the UUL which responds by processing the inputs and providing the outputs.

III. Regression Testing Approach

In this section our regression testing approach and its basic steps are introduced. The proposed approach is depicted on Fig. 1. The method uses a user defined version of the software component as a reference: from this software component version the algorithm synthesize a behaviour model which can be used for test generation. Various test coverage criteria can be supported and many algorithms and tools are available to perform the test generation. This generated set of tests are then used for the later versions of the software component to check its conformance with the reference version.

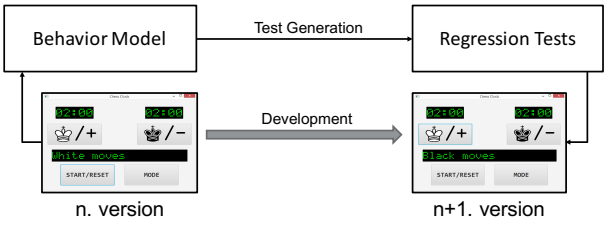


Fig. 1. Regression testing

Challenges. However, the envisioned approach faces some challenges. Automata learning algorithms construct the behavior model of the software component which is a *complex task in itself*. In addition, as a software usually expresses data dependent behavior, learning the automata model often becomes infeasible. Another important issue is that learning the automata model of the former version of the software component and generating test from it yields many test cases which should not hold in the newer version and will lead to many *false positive tests*. Our approach supports the learning algorithm with a specification language and abstraction technique to:

- enable the automata learning of software components with even complex, data dependent behaviour, and to
- focus the regression testing into the relevant parts of the software component, where the test engineer expect no change.

The overview of this process is depicted on Fig. 2. In the first step the framework learns the relevant behavior of the unit, defined by the user using feature models and abstraction. The result of this process is an automaton describing the behavior of the unit. Regression test cases can be easily generated using this automaton: a model-based test generation algorithm or tool can be chosen arbitrarily at this phase, the only question is the expected coverage criterion the tester needs.

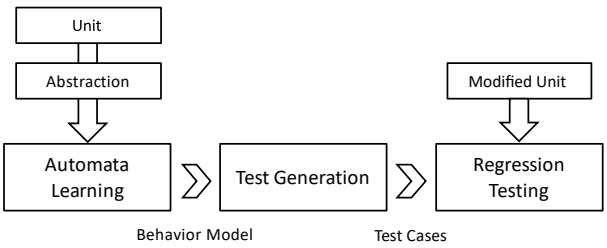


Fig. 2. Overview

In the following we introduce the main steps of the approach – construction of the behavior model and regression testing the modified component – in more details.

A. Constructing the Behavior Model

The first step of the process is the learning of the behavior model. In our work, we have integrated an active automata learning algorithm of LearnLib [8], which produces the behavior model as a Mealy machine [7] – a finite-state machine.

We used abstraction during the learning, as it hides the irrelevant parts of the behaviour and gives the user the means to focus the testing into the functions which are the scope of the regression testing. The user of our approach is able to formulate abstraction rules on the inputs and outputs, i.e. the alphabet of the learning process.

Figure 3. illustrates the role of abstraction during the learning process in the communication between the learning algorithm and the UUL. The queries generated by the learning algorithm are sequences of input symbols of the abstract

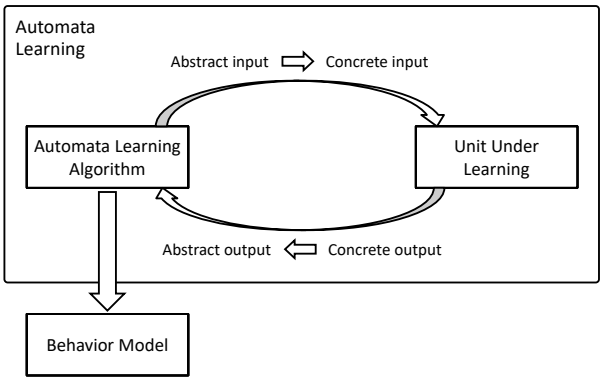


Fig. 3. Automata learning through abstraction

alphabet. These symbols are not directly executable inputs for the UUL, but they represent equivalence classes defined by the abstraction rules. Each abstract input needs to be concretized (choose a concrete executable action). The unit under learning will produce an answer for that particular action. According to the abstraction rules, the concrete response provided by the UUL is mapped to an abstract symbol consumed by the learning algorithm.

B. Regression Testing

Testing consists of the following steps: at first tests are generated from the previously learned behavior model. In the prototype a depth-first-search based test generation method was implemented which provided full state and transition coverage. Various test generation algorithms can be chosen to ensure the desired coverage. These tests can than be executed on the next version of the software component.

The automata learning algorithm constructs an automaton which is an abstract model in the sense that inputs and outputs can not directly drive the software component under test. In order to gain executable test, the abstraction and concretization steps used during the learning are saved so the prototype implementation can use it to produce executable tests.

The verification consists of two main steps:

- At first, the generated tests are executed to examine the new version of the software component.
- If the testing was successful, the framework compares the behavior model of the new version to the former one.

In the first step of verification we run the previously saved test cases on the newer version of the unit. If the result of any tests is an error, manual investigation is needed to decide the reason for the failing test. The reasons for a failing test can be the following:

- A real problem is found. The developers have to fix it.
- False positive occurs because of the inaccurate or not properly defined abstraction.

In order to further increase the efficiency of the analysis, the framework learns the new version of the software component to compare it to the former behavior model by using automaton minimization and equivalence checking.

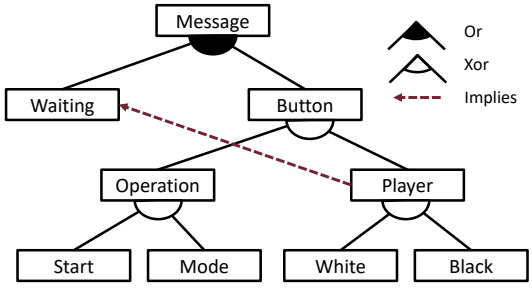


Fig. 4. Feature model

If any of these mentioned verification steps results in failure, we can assume that the modified unit does not conform to the desired behavior model.

IV. Language Support for Regression Testing

The presented approach requires the definition of the interfaces of the UUL, i.e. the possible inputs and outputs. In addition, the user has to define the relevant functionalities for the regression testing, i.e. the abstraction used in the learning process. In the framework, the interfaces and the abstractions are defined with the help of a feature model [6] language.

A. Specifying Communication Interfaces by Feature Models

In our setting, the feature model describes the set of possible input or output messages. Our feature model representation supports two types of features:

- Integer features: have a range of possible values from the integer domain. They can not have child elements.
- Boolean features: have Boolean values.

And it allows one kind of cross-tree constraint:

- Implication: if feature A requires feature B, the selection of A in an input or output implies the selection of B.

An example feature model is depicted by a feature diagram in Fig. 4. It defines the input interface of the chess clock, i.e. the set of all possible input messages. The root is the *Message* feature, which contains a *Waiting* time period, or a *Button* (specified by Or dependency). This represents that the players can wait or push a button to produce a kind of button input. The Waiting feature is an integer feature. The Button feature and its children are of Boolean types. The Button feature can either be an Operation or a Player button press (specified by Xor dependency).

An example input of the chess clock is the following: (1) pushing the white button, then (2) waiting $1s$. This is a possible valid configuration for the previously mentioned feature model.

B. Abstraction and concretization

We have chosen feature model based specification because it supports the formulation of the abstraction and concretization. In our work we implemented the following rules:

- Merge: Representing multiple features as an abstract one.
- Remove: The value of the feature will not be observed.

		Error in Menu 1	Error in Menu 2	Error in Game 1	Error in Game 2	Change
1)	Manual	10/18	2/18	1/18	0/18	8/18
2)	Without abstraction	-	-	-	-	-
3)	Game-focused abstr. 1	0/6	0/6	1/6	0/6	0/6
	Game-focused abstr. 2	0/51	0/51	9/51	0/51	0/51
	Menu-focused abstr. 1	4/6	1/6	0/6	0/6	2/6
	Menu-focused abstr. 2	4/6	1/6	0/6	0/6	2/6
	Menu-focused abstr. 3	6/6	1/6	0/6	0/6	6/6

TABLE I
COMPARISON OF MANUAL AND GENERATED TEST SUITES

Using the feature model of Fig. 4 we illustrate the effects of the abstractions: merging the White and Black buttons will lead to a simple feature model where the merged features will be represented by the Player button. Removing the Start button will result that its value is not observed by the learning algorithms.

V. EVALUATION

In order to evaluate the effectiveness of the proposed framework executed initial measurements on our prototype implementation.

Research questions. The goal of the measurements is to compare the effectiveness of the following sets of tests:

1) Manual test suite
2) Learning and test generation without abstraction
3) Generated test suite using a dedicated abstraction

We are interested in the following questions for each test suite:

RQ1 Is the test suite able to detect randomly injected errors?

RQ2 Is the test suite maintainable? How many modifications are required upon a change of the software?

Measurement method. For the measurements we used the previously presented chess clock statechart developed in YAKINDU [10]. This complex statechart has 12 states and 45 transitions and 9 variables, which results in several billion states when represented as a Mealy machine. In order to evaluate effectiveness of the previously mentioned methods we systematically injected 4 random atomic errors (in different regions of the state machine) to the state machine – motivated by [4] – and an intended change. The manual test suite covered all the transitions of the statechart. For the focused test suite we used 5 dedicated abstractions, e.g. removing the difference between the white and the black player.

Measurement result.: Table I summarizes the test results. Each row represents a testing method (denoted by 1)–3) in the research question). The columns represent various modifications in the software component: the first four are different kind of errors and the last one is an intended change. The cells represent the number of failed test cases in the form of *failed tests / all tests*. '-' represents timeout as we were not able to generate test cases without abstraction, because learning the unit timed out.

Analysis of the results: RQ1 The manually created tests were successfully executed and were able to detect several errors. However, when no abstraction was used, the technique was unable to learn the chess clock unit, thus it can not be directly used to generate test cases. But finally, when using suitable abstractions, the method was able to detect those kind of errors with less test cases. An error can remain hidden from both the manual and the generated test suites. In this case we can assume that we used a too coarse abstraction.

RQ2 Changes in the specification (in the functions of a program) invalidate many of the manual test cases, which have to be (partially) rewritten. However a suitable abstraction will reduce the number of false positive test cases so the test engineering efforts can be decreased.

VI. CONCLUSION AND FUTURE WORK

This paper introduced a model-based regression testing approach utilizing an automata learning algorithm to produce behavior model of a software component. A feature model based language is provided to support the definition of the relevant aspects of the interfaces and serve as the basis of the abstraction. User defined abstractions can drive the learning to focus on those parts of the software component that can be used as a specification model for the testing of the later versions. Our initial experiments showed that the direction is promising and hopefully it can reduce the regression testing efforts needed for testing software components.

In the future we plan to use an automatic abstraction refinement technique – based on the well-known CEGAR [3] – to automatically calculate abstractions.

ACKNOWLEDGMENT

This work was partially supported by the MTA-BME Lendület Research Group on Cyber-Physical Systems and the ÚNKP-16-1-I. New National Excellence Program of the Ministry of Human Capacities. Finally, we thank to Zoltán Micskei for his insightful comments.

REFERENCES

[1] Dana Angluin. Learning Regular Sets from Queries and Counterexamples. *Information and computation*, 75(2):87–106, 1987.

[2] Budapest University of Technology and Economics. *System Modeling course (VIMIAA00)*. https://inf.mit.bme.hu/en/edu/courses/remo-en.

[3] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-Guided Abstraction Refinement. In *International Conference on Computer Aided Verification*, pages 154–169. Springer, 2000.

[4] SC Pinto Ferraz Fabbri, Márcio Eduardo Delamaro, José Carlos Maldonado, and Paulo Cesar Masiero. Mutation Analysis Testing for Finite State Machines. In *Software Reliability Engineering, 1994. Proceedings., 5th International Symposium on*, pages 220–229. IEEE, 1994.

[5] ISO/IEC/IEEE. 24765:2017 Systems and Software Engineering-Vocabulary.

[6] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, DTIC Document, 1990.

[7] George H Mealy. A Method for Synthesizing Sequential Circuits. *Bell System Technical Journal*, 34(5):1045–1079, 1955.

[8] Harald Raffelt, Bernhard Steffen, and Therese Berg. Learnlib: A Library for Automata Learning and Experimentation. In *Proceedings of the 10th international workshop on Formal methods for industrial critical systems*, pages 62–71. ACM, 2005.

[9] Bernhard Steffen, Falk Howar, Malte Isberner, et al. Active Automata Learning: From DFAs to Interface Programs and Beyond. In *ICGI*, volume 21, pages 195–209, 2012.

[10] Yakindu Statechart Tools. *Yakindu*. http://statecharts.org/.

Spectral Leakage in Matrix Inversion Tomosynthesis

Dániel Hadházi, Gábor Horváth
Department of Measurement and Information Systems
Budapest University of Technology and Economics
Budapest, Hungary
Email: {hadhazi, horvath}@mit.bme.hu

***Abstract*— Matrix Inversion Tomosynthesis (MITS) is a linear MIMO system, which deblurs Shift And Add (SAA) reconstructed tomosynthesis slices by applying deconvolution in spectral domain. When implementing MITS, some difficulties, important from both theoretical and practical viewpoints should be considered. In this paper new combined methods are proposed to tackle one of them, the truncation artifact caused by spectral leakage in thoracic imaging using DTS. The effect of the proposed methods is illustrated by coronal human chest slices, validated and compared quantitatively.**

Keywords—MITS; tomosynthesis; deconvolution; spectral leakage; truncation artifact; MIMO system; inverse problem

I. INTRODUCTION

Digital tomosynthesis is a relatively new imaging modality that computes reconstructed slice images from a set of low-dose X-ray projections acquired over a limited angle range [1]. The basic arrangement of a linear chest tomosynthesis is shown in Fig. 1.

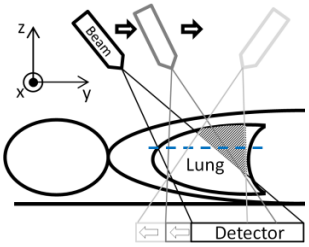


Fig. 1. Schematic arrangement of linear chest tomosynthesis

In a linear tomosynthesis arrangement the X-ray beam-source and a flat panel detector move continuously in parallel along the y axis in opposite directions. During this motion, at about 40-60 different positions, projection images are acquired. According to Fig. 1. these images are obtained in the x-y plane, where x axis denotes the row-, and y axis the column directions. From these projections coronal slice images of the 3D examined volume (e.g. the lung) are reconstructed (one of them is marked by the dashed line in the figure) using reconstruction algorithms. Most of these algorithms are modifications of the algorithms used in CT image reconstruction (e.g. BP, FBP, ART, ML-EM, [2]), with the exception of the matrix inversion tomosynthesis (MITS) [3], an algorithm developed directly for tomosynthesis, which deblurs the reconstructed slices calculated by the Shift and Add (SAA) algorithm [3], [4].

The input projection images that are acquired by the detector can be modelled as the sum of the shifted projections of the coronal slices of the examined volume:

$$I_j^{(l)} = \sum_k R_j^{(k)} * H^{(l,k)} \tag{1}$$

Here $I_j^{(l)}$ denotes the j-th column of the l-th input projection image, $R_j^{(k)}$ denotes the j-th column of the central projection of the k-th reconstructed coronal slice, $*$ denotes the convolution and, $H^{(k,l)}$ denotes the weight function between the k-th coronal slice and the l-th input projection image. The goal of the reconstruction method is to estimate the R functions from the input projections. $H^{(k,l)}$ is a shifted Dirac-delta function that can be derived from the acquisition geometry [4]. (1) can be also examined in Fourier domain:

$$\hat{\mathbf{i}}_j(\omega) = \hat{\mathbf{H}}(\omega) \cdot \hat{\mathbf{r}}_j(\omega) \tag{2}$$

$\hat{\mathbf{i}}_j(\omega)$ denotes the column vector containing the spectral components of the j-th columns of the input projections, $\hat{\mathbf{r}}_j(\omega)$ gives a column vector constructed similarly from the j-th columns of the slices of the examined volume:

$$\hat{\mathbf{i}}_j(\omega) = \left[FT_\omega\left\{I_j^{(1)}\right\} \quad FT_\omega\left\{I_j^{(2)}\right\} \quad \dots \quad FT_\omega\left\{I_j^{(N_P)}\right\} \right]^T \tag{3}$$

$FT_\omega\{\cdot\}$ is the operator of 1D continuous Fourier transform at ω frequency, and N_P denotes the number of input projection images. The matrix that is denoted by $\hat{\mathbf{H}}(\omega)$ is defined by:

$$\hat{\mathbf{H}}(\omega)_{(l,k)} = FT_\omega\left\{H^{(l,k)}\right\} \tag{4}$$

Although SAA is simple and easy to implement, its results, the reconstructed slice images are rather poor. The SAA reconstructed slices can be represented as convolution of the actual in-plane structures and a blurring function relating how structures located in one plane are reconstructed in another. So it can be modelled by a MIMO shift invariant linear system:

$$\hat{\mathbf{s}}_j(\omega) = \hat{\mathbf{G}}(\omega) \cdot \hat{\mathbf{i}}_j(\omega) \tag{5}$$

where $\hat{\mathbf{G}}(\omega)$ denotes the matrix of the SAA reconstruction, $\hat{\mathbf{s}}_j(\omega)$ gives a column vector containing the spectral com-

ponents of the j-th columns of the SAA reconstructed slices similarly to (3). The exact value of $\hat{\mathbf{G}}(\omega)$ is defined by:

$$\hat{\mathbf{G}}(\omega)_{(l,k)} = \overline{\hat{\mathbf{H}}(\omega)_{(k,l)}} \tag{6}$$

where $\bar{a}$ stands for the operator of complex conjugating. SAA slices can also be modelled as the sum of the shifted projections of the examined volume's ideally thin slices:

$$\hat{\mathbf{s}}_j(\omega) = \hat{\mathbf{F}}(\omega) \cdot \hat{\mathbf{r}}_j(\omega) \tag{7}$$

while $\hat{\mathbf{F}}(\omega) = \hat{\mathbf{H}}(\omega)^* \cdot \hat{\mathbf{H}}(\omega)$ denotes the blurring matrix at ω. Based on this equation, the MITS estimates the slice images by a deconvolution which is implemented in Fourier domain:

$$\hat{\mathbf{r}}_j(\omega) = \hat{\mathbf{F}}(\omega)^{-1} \cdot \hat{\mathbf{s}}_j(\omega) \tag{8}$$

Contrary to the classical image deconvolution problems, in this case a MIMO system is inverted, which requires consistency of the input projections.

The whole algorithm has three significant difficulties: (*) using discrete Fourier transform (DFT) for (2)-(8), spectral leakage may happen; (**) $\mathbf{F}(\omega)$ is ill-conditioned (mainly at low frequencies) as it is shown in [5], and (***) the examined 3D volume can't be modelled by finite number, ideally thin, zero-thick slices [4].

This paper focuses on the first problem. A composite method is proposed which can effectively handle the spectral leakage problem of the MITS reconstruction (here and hereafter in MITS we mean the cascade of SAA reconstruction (5) and the deblurring step (8)). In the next section the artifacts caused by spectral leakage and the proposed method to reduce these are described. The validation of the proposed method and a comparison to [5] is presented in the last section.

II. Artifacts Caused By Spectral Leakage

Discrete Fourier Transform (DFT) implicitly assumes that the sampled, finite length signal is only a period of a periodic signal. The side effect of the DFT calculated spectrum caused by the violation of this assumption is the so-called spectral leakage. In MITS reconstruction this causes *y*-directional intensity oscillation pattern over the whole image (called boundary artifact). In Fig. 3. a human thoracic coronal plane is shown (one of the corresponding input projections is in Fig. 2.), which is reconstructed by MITS without handling the spectral leakage problem. In the figure the area circumscribed by an ellipsoid shows low frequency artifacts, while the arrow points to an area where high frequency artifacts (horizontal lines) are generated, both are due to spectral leakage. The effect of spectral leakage can be reduced if the image is modified in such a way that there is a smooth continuation between the lower and the upper parts of the images (the image can be considered as one period of a periodic signal without any abrupt change at the borders of a period). Additionally due to the periodic assumption of the DFT in (8), the SAA reconstructed slices have to be extrapolated periodically in vertical direction (along *y* axis). Otherwise, due to the so called wrap-around effect, the reconstruction of anatomical structures located near to the top of a slice depends on lower placed regions of the SAA slices (e.g. the reconstruction of the apex of the lung depends on the SAA reconstruction of the diaphragm). The minimal height of the extrapolated area can be determined analytically from the inverse Fourier transform of the blurring matrix ($\hat{\mathbf{F}}$ in (7)).

Based on our previous publications ([6] and [4]), in the case of MITS it is reasonable to reconstruct significantly more (N_R) slices than the number of input projections (N_P), but this means that the deblurring equation (7) is overdetermined. This implies that more SAA slices are reconstructed than the number of input projections. If we extrapolate the SAA slices one by one, than probably there will not exist N_P realistic projection images that satisfy equation (5).

As our proposed extrapolating method can only extrapolate the slice or the projection images one by one (an alternative method that takes into account all of the projections/ SAA slices is under investigation) the input projection images are extrapolated. In this case (5) in discrete Fourier domain produces adequately extrapolated SAA slices.

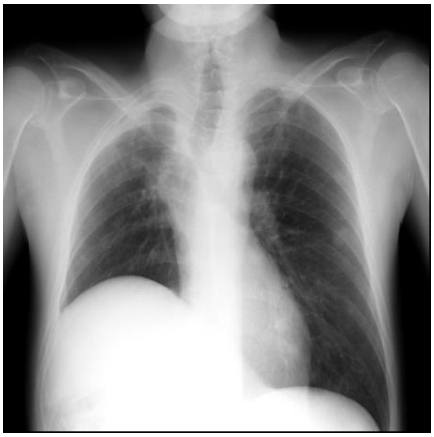


Fig. 2. Example of an input projection image.

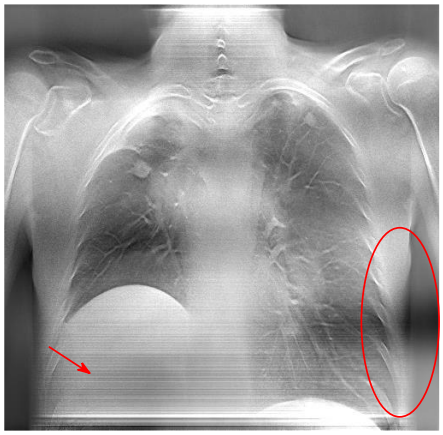


Fig. 3. MITS reconstructed thoracic plane without compenstaion of the spectral leakage artifact.

A. *Windowing and Post filtering*

In digital signal processing, the spectral leakage effect is usually reduced by windowing the signal before calculating its spectrum. However, windowing as an amplitude reduction effect is significant in the top and the bottom of the reconstructed slices if the input projections are windowed. This effect can't be effectively handled after the deconvolution step (8), as if we apply windowing, the whole reconstruction becomes a shift variant linear system. To compensate this phenomenon, the reconstructed slices are post-filtered by dividing the reconstructed slice images with the intensities of the reconstructed slices calculated from the above input simulated projection images:

$$W^{(i)}(j,k) = w(j) \tag{9}$$

where $w(\cdot)$ is the windowing function, and $W^{(i)}$ is the i-th generated windowing input image. In order to avoid dividing by a small number in the post-filtering step Hamming window function (with $\alpha = 0.54$) was chosen. It is important to note that post-filtering based compensation is only a heuristic step, it can not compensate perfectly the artifact caused by windowing. The effect of the whole compensation is illustrated in Fig. 4. due to the numerical instability caused by the post-filtering step, the uppermost and the lowermost rows of the reconstructed slices are usually noisy (marked by the ellipsoids in Fig. 4.). This effect can significantly weak the diagnostic value of the reconstruction, therefore some further steps are required to reduce this artifact.

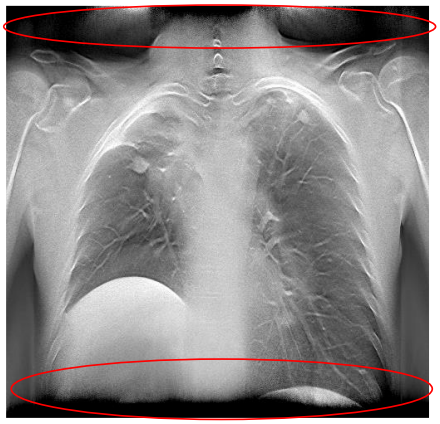


Fig. 4. Result of the windowing and post filtering compensation for the same slice as illustrated by Fig. 3.

B. *Smooth extrapolation*

The basics of this method are described in [7]. The main idea is based on that the vertical intensity oscillation pattern (which is an effect of the spectral leakage) on the reconstructed slices is caused by the discontinuities between the first and the last rows of the input projections. These discontinuities can be minimized by applying a maximally smooth interpolation between the upper and lower ends of the projection images:

$$\mathbf{E}^{(i)} = \begin{bmatrix} \mathbf{\Gamma}^{(i)} \\ \mathbf{I}^{(i)} \end{bmatrix} \tag{10}$$

Here $\mathbf{E}^{(i)}$ denotes the i-th extrapolated projection, $\mathbf{I}^{(i)}$ denotes the i-th input projection, $\mathbf{\Gamma}^{(i)}$ denotes the image part calculated by the extrapolation in the case of the i-th projection. $\mathbf{\Gamma}^{(i)}$ is calculated in order to get a maximally smooth extrapolated image ($\mathbf{E}^{(i)}$), where the metric of smoothness is defined by:

$$\left\| \mathbf{\Delta}\left\{ \mathbf{E}^{(i)} \right\} \right\|_F^2 \tag{11}$$

where $\mathbf{\Delta}\{\cdot\}$ denotes the discrete, two-dimensional, vertically circular Laplacian of Gaussian operator, and $\|\cdot\|_F$ denotes the Frobenius norm. The problem can be formalized as a quadratic programing (QP) problem, which can be effectively solved in this special case.

C. *Mixture of the two algorithms*

According to the validation of the whole process (it is described in details in the next section) the spectral distortion caused by the extrapolation is larger than the spectral distortion caused by windowing the projections, however the intensity reduction at the upper and lower bounds of the images does not occur. Therefore we blend the two reconstructions calculated from the two sets of preprocessed projections (one of them is modified by the windowing and post-filtering method, while the other is modified by smooth extrapolation) with spatial varying weights. Near the boundary of the reconstructed slices the reconstruction calculated after the smooth extrapolation dominates while the windowing based solution dominates in the central parts. The result of this blending is illustrated by Fig. 5. The difference between the results of the mixing and the smooth extrapolation based methods is moderated in real slice images. Based on the numerical validation the mixing based correction outperforms the smooth extrapolation based method.

III. Validation and Conclusions

The noise sensitivity of the MITS reconstruction depends on the spatial frequency of the input projections [4]. Therefore such a validation is used that examines the spectrum of the distortion of the reconstructed slices. As the applied extrapolation can't be modelled by shift invariant linear approach, the distortion can't be examined by analytic methods (e.g. measuring the MTF). However, to estimate the distortion is possible by comparing the original slices and the MITS calculated slices reconstructed from the projections of the same artificial volume.

In contrast to the CT, the DTS projections are obtained only from a limited angle range. This implies that the thickness of the reconstructed slices is bigger and depends on the spatial frequency of the images [5]. In the low frequency domain the reconstructed slices are significantly larger than in the higher frequency domain. This means that the low frequency components of the reconstruction depend on a thicker part of the examined volume (e.g. the DC component depends on the whole volume). In order to separate this effect from the

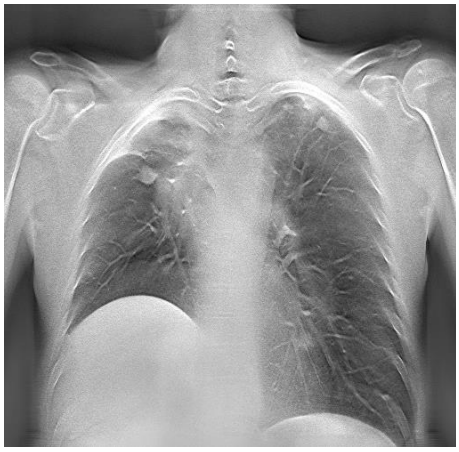


Fig. 5. Result of mixed reconstruction.

degradation caused by the spectral leakage, reconstructions are calculated from projections of artificial volumes containing only a few (in our case 4), ideally thin nonempty coronal slices of a chest reconstruction, placed as far from each other as possible (there was approx. 60 mm-s between adjacent slices). 15 such volumes were used. Formally, in the validation the error of reconstruction of the nonempty slices was computed as:

$$err(\omega) = \left\langle \frac{\left| FT_\omega\left\{\mathbf{R}_j^{(i,k)}\right\} - FT_\omega\left\{\mathbf{P}_j^{(i,k)}\right\}\right|}{\left| FT_\omega\left\{\mathbf{P}_j^{(i,k)}\right\}\right|} \right\rangle \qquad (12)$$

where $\mathbf{R}_j^{(i,k)}$ denotes the j-th column of the k-th nonempty slice of the i-th reconstruction, $\mathbf{P}_j^{(i,k)}$ denotes the j-th column of the corresponding projected slices, $\langle\ \rangle$ denotes averaging over i, j, k. $\lambda_{(i,k)}$ denotes the un-normalized MTF between the in-plane signal positioned at the i-th volume's k-th nonempty slice and its reconstruction. The ideal MTF of the MITS reconstruction can be calculated as it is described in [8]. In order to minimize the DFT caused spectral distortion windowing is applied before calculating the Fourier transform. It is important to note that this windowing makes this kind of validation biased towards to the windowing based extrapolation method since it underweights the border parts of the slices, where the windowing based extrapolation method mainly creates artifacts.

In the validation of the reduction of the spectral leakage 4 different methods were compared: (1) windowing and post filtering, (2) smooth extrapolation, (3) mixture of these two techniques, and (4) a method described in [3] ("base method"). The error curves are plotted in Fig. 6. Based on the result, we can see that both the windowing and post filtering and the smooth extrapolation methods outperform the base method. From the two methods the error of the windowing based extrapolation is less in wide part of frequency domain but in image domain the intensity artifact described before degrades the diagnostic value of the reconstructions. It is also concluded that the error of applying the mixture correction

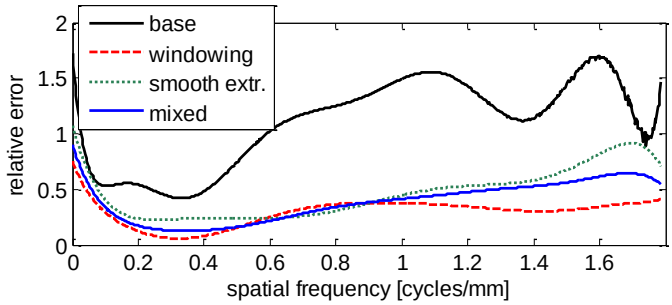


Fig. 6. Result of the validation. All of the three methodes introduced by this paper outperform the base method.

method is not significantly larger than the error of the windowing based method under the bandwidth of the projected slices (1.1 cylces/mm). This bandwidth is approximately the average bandwidth of the chest tomosynthesis slice images.

Based on our experiments and the results of the validation, the mixture of the two compensation methods effectively reduces the artifacts caused by the spectral leakage without degrading the slices at their borders. Also we believe that extrapolation methods can significantly be improved by taking into account all of the projections during extrapolation (now projections are extrapolated one by one). These methods can handle cases, when there exists part of the volume which is projected into only a subset of the projections (illustrated by the shaded area of Fig. 1.), so they better fit to the whole MIMO system of the reconstruction. Such a method is under investigation.

IV. References

[1] D.G. Gant, "Tomosynthesis: a three-dimensional radiographic imaging technique," *IEEE Trans. Biomed. Eng.*, 19, 20-28, 2003.

[2] M. Beister et al, "Iterative reconstruction methods in X-ray CT," *Physica Medica*, : 28 (2), 94-108, 2012.

[3] D.J. Godfrey et al, "Practical strategies for the clinical implementation of matrix inversion tomosynthesis (MITS)," *Proc. SPIE Physics of Medical Imaging 5030*, 379-39, 2003.

[4] D. Hadházi et al, "Partial Volume Artifact in MITS reconstructed Digital Chest Tomosynthesis," *Proc 23rd Phd Myni-Simposium*, 14-18, 2016.

[5] D.J. Godfrey et al, "Stochastic noise characteristics in matrix inversion tomosynthesis (MITS)," *Med. Phys.*, 36 (5), 1521-1532, 2009.

[6] D. Hadházi et al, "Measurement Quality of MITS reconstructed Tomosynthesis Slices Depending from Parameters of Images Acquiring and Reconstruction Algorithm," *IFMBE Proc.*, 45, 114-117, 2015.

[7] R. Liu et al, "Reducing boundary artifacts in image deconvolution," *IEEE Int. Conf. Image Proc.*, 2008, 505-508, 2008.

[8] D.J. Godfrey et al, "Optimization of the matrix inversion tomosynthesis (MITS) impulse response and modulation transfer function characteristics for chest imaging," *Med. Phys.*:33, 655-667, 2006.

Exploratory Analysis of the Performance of a Configurable CEGAR Framework

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

***Abstract*—Formal verification techniques can check the correctness of systems in a mathematically precise way. However, their computational complexity often prevents their successful application. The counterexample-guided abstraction refinement (CEGAR) algorithm aims to overcome this problem by automatically building abstractions for the system to reduce its complexity. Previously, we developed a generic CEGAR framework, which incorporates many configurations of the algorithm. In this paper we focus on an exploratory analysis of our framework. We identify parameters of the systems and algorithm configurations, overview some possible analysis methods and present preliminary results. We show that different variants are more efficient for certain tasks and we also describe how the properties of the system and parameters of the algorithm affect the success of verification.**

I. Introduction

As safety critical systems are becoming more and more prevalent, assuring their correct operation is gaining increasing importance. Formal verification techniques (such as model checking [1]) can check whether the model (a formal representation) of a system meets certain requirements by traversing its possible states and transitions. However, a typical drawback of using formal methods is their high computational complexity. Abstraction is a general technique to reduce complexity by hiding irrelevant details. However, finding the proper precision of abstraction is a difficult task. Counterexample-guided abstraction refinement (CEGAR) is an automatic verification algorithm that starts with a coarse initial abstraction and refines it iteratively until a sufficient precision is obtained [2].

In our previous work [3] we examined different variants of the CEGAR algorithm and concluded that each of them has its advantages and shortcomings. The foundations of a modular, configurable CEGAR framework were also developed that can incorporate the different CEGAR configurations (variants) in a common environment. The framework relies on first order logic (FOL): models are described with FOL formulas and the algorithms use SAT/SMT solvers [4] as the underlying engine.

The framework is under development, but it already realizes several configurations and permits the verification of some input models. We performed an experiment by evaluating these configurations on the models of some hardware and PLC systems. In this paper we present an exploratory analysis of the results: we identify parameters and metrics of the models and configurations as input and output variables. We give an overview on possible analysis methods and present preliminary comparisons, revealing that different configurations are more suitable for certain models. We show relationships between the properties of the input model, the parameters of the algorithm (e.g., abstraction method, refinement strategy) and the success and efficiency of verification.

II. Experiment Planning

In our experiment several *configurations* of the CEGAR algorithm were executed on various input *models* and the results were analyzed [5].

A. Variables

Variables of the experiment are grouped into three main categories: parameters of the model (input), parameters of the configuration (input), metrics of the algorithm (output). Variables along with their type and description are listed in Table I. Some other parameters of the input models were also identified, but these are domain specific and not applicable to all inputs (e.g., number of gates in a hardware circuit). Therefore, these parameters were omitted in this experiment.

There are some additional constraints on the variables. UNSC refinement cannot be used in PRED domain, but besides that, all combinations of the algorithm parameters are valid, yielding a total number of 20 configurations. It is also possible that the algorithm did not terminate within a specified time. In this case all output variables are NA (not available) values. Furthermore, the length of the counterexample is NA if the model is safe.

B. Objects

As the framework is under development, its current performance and limited input format only permits the verification of smaller input models from certain domains. Nevertheless, some smaller standard benchmark instances were used from the Hardware Model Checking Competition [11]. These models encode hardware circuits with inputs, outputs, logical gates and latches. Some industrial PLC software modules from a particle accelerator were also verified. A total number of 18 models were used, consisting of 12 hardware and 6 PLCs.

C. Measurement Procedure

The framework is implemented in Java and it was deployed as an executable `jar` file. Measurements were ran on a 64 bit Windows 7 virtual machine with 2 cores (2.50 GHz), 16 GB

TABLE I
VARIABLES OF THE EXPERIMENT.

Category	Name	Type	Description
Input (model)	Type	Factor	Type of the model. Possible values: hw (hardware), plc (PLC, i.e., Programmable Logic Controller).
	Model	String	Unique name of the model.
	Vars	Integer	Number of FOL variables in the model.
	Size	Integer	Total size of the FOL formulas in the model.
Input (config.)	Domain	Factor	Domain of the abstraction. Possible values: PRED (predicate [6]), EXPL (explicit value [7]).
	Refinement	Factor	Abstraction refinement strategy. Possible values: CRAIGI (Craig interpolation [8]), SEQI (sequence interpolation [9]), UNSC (unsat core [10]).
	InitPrec	Factor	Initial precision of the abstraction. Possible values: EMPTY (empty), PROP (property-based).
	Search	Factor	Search strategy in the abstract state space. Possible values: BFS, DFS (breadth- and depth-first search).
Output (metrics)	Safe	Boolean	Result of the algorithm, indicates whether the model meets the requirement according to the algorithm.
	TimeMs	Integer	Execution time of the algorithm (in milliseconds).
	Iterations	Integer	Number of refinement iterations until the sufficiently precise abstraction was reached.
	ARGsize	Integer	Number of nodes in the Abstract Reachability Graph (ARG), i.e., the number of explored abstract states.
	ARGdepth	Integer	Depth of the ARG.
	CEXlen	Integer	Length of the counterexample, i.e., a path leading to a state of the model that does not meet the requirement.

RAM and JRE 8. No other virtual machines were running on the host during the measurements. Z3 [12] was used as the underlying SAT/SMT solver. The measurement procedure was fully automated. The configurations and models were listed in `csv` files and a script was written that loops through each configuration and model pair and runs the framework with the appropriate parameters (based on the configuration and the model). The script waits until the verification finishes or a certain time limit is reached, outputs the result (or timeout) to a `csv` file and repeats the procedure a given number of times.

In our current setting, 20 configurations were executed on 18 input models and each execution was repeated 5 times, yielding a total number of $18 \cdot 20 \cdot 5 = 1800$ measurements. The time limit for each measurement was 8 minutes. With this limit, 1120 executions terminated (successful verifications).

D. Research Questions

In our current work we focus on a preliminary, exploratory analysis of the measurement results. The following research questions are investigated.

- RQ1 What are the overall, high level properties of the data set, e.g., distribution of execution time, percentage of safe models?
- RQ2 How do individual parameters of the configuration affect certain output variables, e.g., PRED or EXPL domain yields faster execution time?
- RQ3 Which input parameters influence certain output variables the most, e.g., is the Domain or the Refinement more influential on the execution time?

E. Analysis Methods

RQ1 can be answered with basic descriptive statistics and summarizing plots (e.g., box plots, heat maps), yielding a good overview on the characteristics of the data. RQ2 can be examined with the aid of interactive, visual tools. Parallel coordinates, scatter plots and correlation diagrams are suitable for this purpose, where relationships between the different dimensions can be quickly revealed. RQ3 can be analyzed with decision trees, principle component analysis and other methods that can extract the most relevant information from a set of data. This analysis can also provide an aid to pick the most appropriate configuration for a given task.

F. Threats to Validity

External validity can be guaranteed by selecting representative input models. As mentioned in Section II-B, smaller hardware and software instances were used. As the performance and the input formats of the framework will increase, it will be possible to verify more, larger instances and other kinds of models (e.g., tasks from the Software Verification Competition [13]), which improves the external validity of the analysis. Other tools were not evaluated, because this experiment focused only on our framework. Internal validity is increased by running the measurements repeatedly on a dedicated machine. Furthermore, the framework has also been undergone unit and integration testing.

III. ANALYSIS

This section presents the analyses and results to our research questions. The results of measurements were collected to a single `csv` file, which was analyzed using the R software environment version 3.3.2 [14].

Let D denote the whole data set, $D_{\text{succ}} \subseteq D$ the successful executions (no timeout) and $D_{\text{cex}} \subseteq D_{\text{succ}}$ the successful executions where the model is not safe (i.e., a counterexample is present). The relation of the data sets along with their size is depicted in Figure 1.

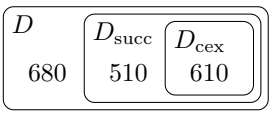


Fig. 1. Overview of the data sets with the number of measurements.

A. RQ1: High Level Overview

First we checked that (1) either all executions of a configuration on a model gives the same safety result and (2) all configurations agree on the results for each model. The lack of the previous properties would obviously mean that the algorithms are not sound, but for our data set they hold.

The histograms and box plots in Figure 2 give a high level overview of the distribution and range of output variables. It can be seen that for most of the variables, the IQR is small and there are many outliers.

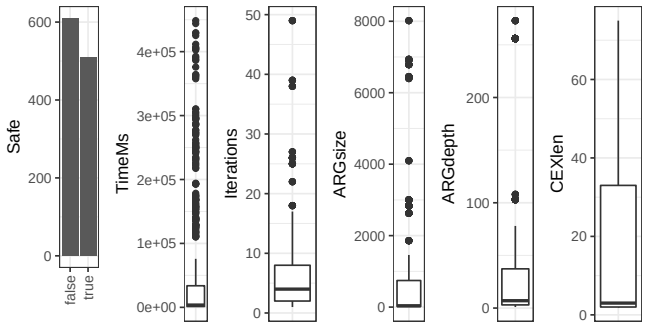


Fig. 2. Overview of individual output variables.

Figure 3 gives an overview on execution time. Each cell of the grid represents the average time of the 5 repeated executions of a configuration on a model. Configurations are abbreviated with the first letters of their parameters, e.g., PSED means PRED domain, SEQI refinement, EMPTY initial precision and DFS search. The maximal relative standard deviation (RSD) of the repeated executions is 10.5%, which is a low value. This is not surprising because our algorithms are deterministic, with the possible exception of some heuristics in external solvers. This low RSD value suggests internal validity and allows us to represent repeated measurements with their average. White cells mean that all executions timed out and colored cells correspond to a logarithmic scale in milliseconds. It can be seen that each model was verified by at least one configuration. It is interesting that plc3 was only verified by a single configuration, but in a rather short time. Also, there is no single configuration that can verify each model, but some of them can verify almost all models. Some of the models (e.g., hw9) are easy for all configurations, but some of them (e.g., plc1) expose 2–3 orders of magnitude difference between the configurations.

B. RQ2: Effect of Individual Parameters

Effect of individual parameters on certain output variables were also compared. The most interesting observation was the effect of the domain on the execution time. This analysis was done by forming pairs from the measurements, similarly to the join operation known from databases. We calculated $D \times D$ and kept rows where every input variable is the same, except the domain, which is different. This means that each row contains an execution for PRED and EXPL domains with the rest of

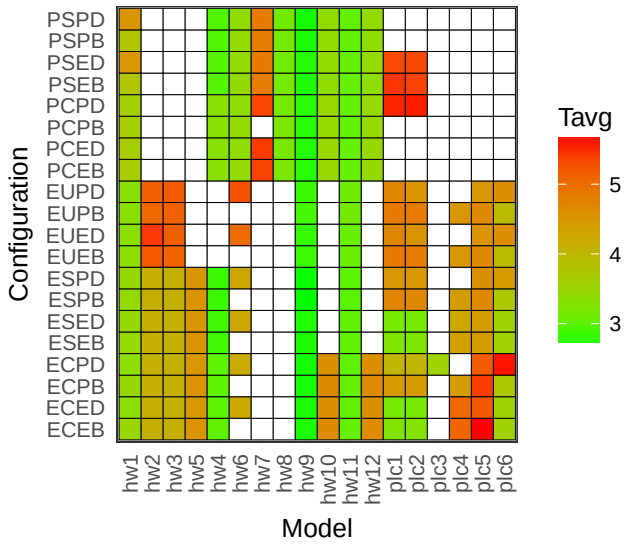


Fig. 3. Average execution time (milliseconds, logarithmic scale).

the configuration (and the model) being the same. Only those rows were kept where at least one domain was successful. Each point in Figure 4 represents a row, where the x and y coordinates correspond to the execution time of PRED and EXPL respectively. Furthermore, points have different colors based on Type. Points at the right and top edges correspond to timeouts. An important property of this kind of plot is that points above the identity line mean that PRED was faster, while points below mean the opposite. It can be observed that verification of PLC models is more efficient in the EXPL domain. Hardware models however, show some diversity, both domains have good results.

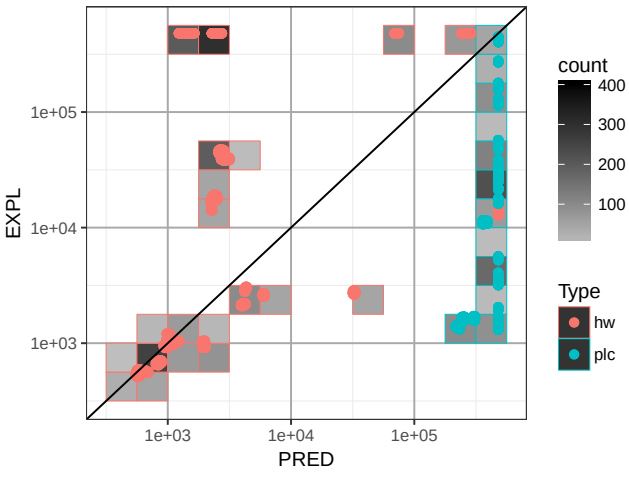


Fig. 4. Comparison of execution time for the different domains.

An other interesting result was the comparison of the number of iterations for CRAIGI and SEQI refinements. Only those rows were kept where both refinements were successful.

It can be seen in Figure 5 that SEQI yields less iterations in almost all cases. It can also be observed, that the difference between the two refinement strategies is small for hardware models, but it can be much larger for certain PLC models.

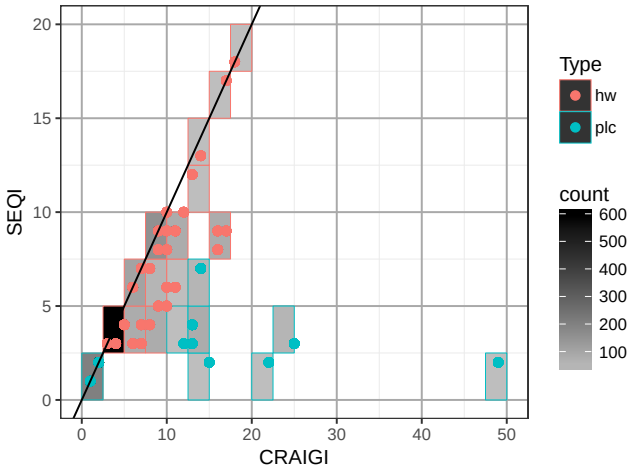


Fig. 5. Comparison of iterations for the different refinements.

C. RQ3: Influence of Input Parameters on Output Variables

The influence of input parameters on certain output variables were also examined. Amongst the observations, the most interesting was the influence of Type and the parameters of the configuration on the success of verification (i.e., a non-timeout). Figure 6 shows the decision tree. It can be seen that the most influential parameters are Domain, Type and Refinement. In the terminal nodes SUCC and TO represent success and timeout respectively. It can be observed that for example choosing PRED domain for PLCs will most likely not succeed. On the other hand, it is likely to succeed for hardware models. It can also be seen that EXPL domain with CRAIGI refinement is likely to succeed regardless of the type of the model. This fact is also confirmed by the small number of white cells in the bottom four rows of Figure 3.

IV. Conclusions

In our paper we evaluated various configurations of our CEGAR framework on different models, including hardware and PLCs. We identified properties of models and parameters of the algorithm that can serve as input and output variables. We presented some possible analysis methods with the corresponding results, including descriptive statistics, different plots and decision trees. Although the results being preliminary, we showed that different configurations are more suitable for certain tasks and we also revealed connections between the type of the model, the parameters of the algorithm and the success of verification. Based on these results we will be able to improve the framework and perform measurements with a larger number of input models and configurations, yielding a larger data set. We hope that further analysis on this data set will allow us to automatically determine the most appropriate configuration of the algorithm for a given verification task.

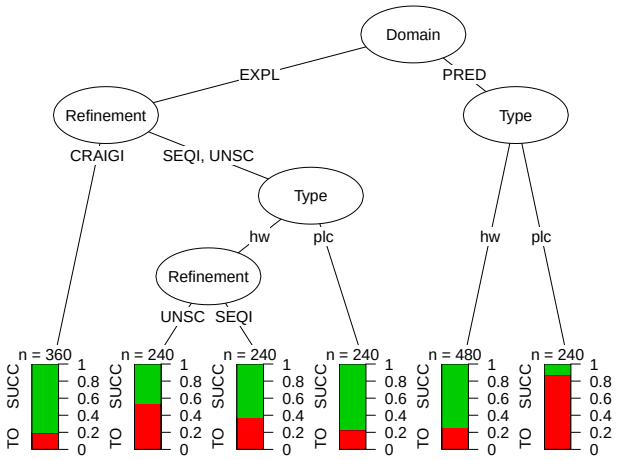


Fig. 6. Decision tree on the success of verification.

References

[1] E. M. Clarke, O. Grumberg, and D. Peled, *Model checking*. MIT Press, 1999.

[2] 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.

[3] A. 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.

[4] A. Biere, M. Heule, and H. van Maaren, *Handbook of Satisfiability*. IOS press, 2009.

[5] P. Antal, A. Antos, G. Horváth, G. Hullám, I. Kocsis, P. Marx, A. Millinghoffer, A. Pataricza, and A. Salánki, *Intelligens adatelemzés*. Typotex, 2014.

[6] 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.

[7] E. M. Clarke, A. Gupta, and O. Strichman, "SAT-based counterexample-guided abstraction refinement," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 7, pp. 1113–1123, 2004.

[8] K. McMillan, "Applications of Craig interpolants in model checking," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. LNCS. Springer, 2005, vol. 3440, pp. 1–12.

[9] Y. Vizel and O. Grumberg, "Interpolation-sequence based model checking," in *Formal Methods in Computer-Aided Design*. IEEE, 2009, pp. 1–8.

[10] M. Leucker, G. Markin, and M. Neuhäußer, "A new refinement strategy for CEGAR-based industrial model checking," in *Hardware and Software: Verification and Testing*, ser. LNCS, vol. 9434. Springer, 2015, pp. 155–170.

[11] G. Cabodi, C. Loiacono, M. Palena, P. Pasini, D. Patti, S. Quer, D. Vendraminetto, A. Biere, K. Heljanko, and J. Baumgartner, "Hardware model checking competition 2014: An analysis and comparison of solvers and benchmarks," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 9, pp. 135–172, 2016.

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

[13] D. Beyer, "Reliable and reproducible competition results with BenchExec and witnesses (report on SV-COMP 2016)," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. LNCS. Springer, 2016, vol. 9636, pp. 887–904.

[14] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2016. [Online]. Available: https://www.R-project.org/

Possibilities for Modeling a Signal Structure Within a Fourier Analyzer

Csaba F. Hajdu*†, Tamás Dabóczi†, Christos Zamantzas*
*European Organization for Nuclear Research (CERN), Geneva, Switzerland. Email: {cshajdu,czam}@cern.ch
†Budapest University of Technology and Economics, Department of Measurement and Information Systems, Budapest, Hungary. Email: {chajdu,daboczi}@mit.bme.hu

***Abstract*—This paper studies a way to extend the Fourier analyzer suggested by G. Péceli, with the aim of improving the detectability of a known periodic signal component in its input signal. Possibilities for modeling a signal structure assuming the amplitude and phase relationships between its components to be known are presented.**

I. Introduction

Signal parameters may be measured and various transforms can be calculated by using recursive estimation methods based on conceptual signal models. These methods are well suited for real-time applications due to their recursive nature.

The conceptual signal model used in these methods is a hypothetical dynamical system, and the signal being measured is assumed to be its output. The state vector of the conceptual signal model, corresponding to the parameter vector of the measured signal, will then be estimated by the structure. If the conceptual signal model is deterministic, the measurement system is referred to as observer.

Hostetter introduced a recursive observer calculating the discrete Fourier transform of the input signal [1]. Péceli extended this method and suggested an observer structure consisting of signal channels containing a discrete integrator, allowing the calculation of arbitrary linear transforms of the input signal [2]. When used as a spectral observer, this structure is referred to as Fourier Analyzer (FA).

The Fourier coefficients calculated by the Fourier analyzer may be used to detect the presence of a signal component with an arbitrary spectrum within the signal being measured. In this paper, we assume a known amplitude and phase relationship between the components of the signal structure we aim to detect, and we investigate possibilities for building a corresponding model into the Fourier analyzer.

II. The Fourier Analyzer

This section presents the fundamentals of the Fourier analyzer introduced by Péceli [2].

A. The Conceptual Signal Model

The conceptual signal model used in Péceli's signal observer [2] is shown in Figure 1.

Tamás Dabóczi acknowledges the support of ARTEMIS JU and the Hungarian National Research, Development and Innovation Fund in the frame of the R5-COP project.

When the observer structure is used as a spectral observer, the state variables of the conceptual signal model correspond to the complex Fourier components of the signal. Therefore, the conceptual signal model itself can be viewed as a complex multisine generator performing an inverse discrete Fourier transform (DFT) on the Fourier components of the signal. Each state variable represents a harmonic resonator of the corresponding frequency, thus these are often referred to as resonator channels.

In this paper, we will use the linear time-variant (LTV) version of the resonator-based observer as starting point. In this case, the conceptual signal model is a system described by time-variant equations, with state variables that do not vary in time: $\mathbf{x}_n$ is constant and $\mathbf{c}_n$ varies in time in (1)–(5) below. It is worth mentioning here that a linear time-invariant (LTI) realization has also been put forth [2].

The system equations describing the LTV conceptual signal model of the Fourier analyzer are as follows:

$$\mathbf{x}_{n+1} = \mathbf{x}_n, \tag{1}$$

$$\mathbf{x}_n = [x_{i,n}]^T \in \mathbb{C}^{N\times 1}, \qquad i = -K, \ldots, K, \tag{2}$$

$$y_n = \mathbf{c}_n^T \mathbf{x}_n, \tag{3}$$

$$\mathbf{c}_n = [c_{i,n}]^T \in \mathbb{C}^{N\times 1}, \qquad i = -K, \ldots, K, \tag{4}$$

$$c_{i,n} = e^{j2\pi i f_0 n} = z_i^n, \qquad z_i = e^{j2\pi i f_0}, \tag{5}$$

where y_n is the signal we intend to observe. The state vector $\mathbf{x}_n$ contains the $N = 2K + 1$ complex Fourier components, including DC, corresponding to the K harmonics of the signal.

With a real-valued input signal, the Fourier coefficients form complex conjugate pairs: $x_{i,n} = x^*_{-i,n}$.

The relative frequency of the fundamental harmonic with respect to the sampling frequency is $f_0 = {}^{f_1}/_{f_s}$, where f_1 is the frequency of the fundamental harmonic and f_s is the sampling frequency.

The number of harmonics K is such that the following inequality holds: $K \cdot f_1 < {}^{f_s}/_2 \leq (K+1) \cdot f_1$. In case of equality, $i = -K, \ldots, K+1$ and $N = 2K + 2$ above.

B. The Resonator-Based Observer

An appropriately designed observer can estimate the state variables of the conceptual signal model, and thereby the complex Fourier coefficients of the input signal. Figure 2 shows the block diagram of the observer matching the LTV conceptual signal model presented in Section II-A.

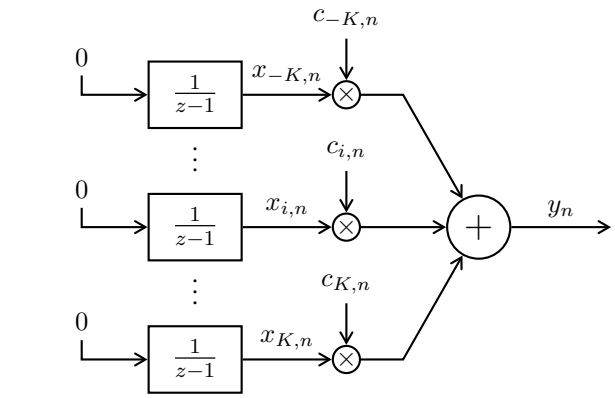


Fig. 1. Block diagram of the conceptual signal model, linear time-variant model. The integrators hold their intial preset values, corresponding to the complex Fourier coefficients of the signal. The output signal then arises as the linear combination of the integrator outputs with the time-varying $c_{i,n}$ coefficients.

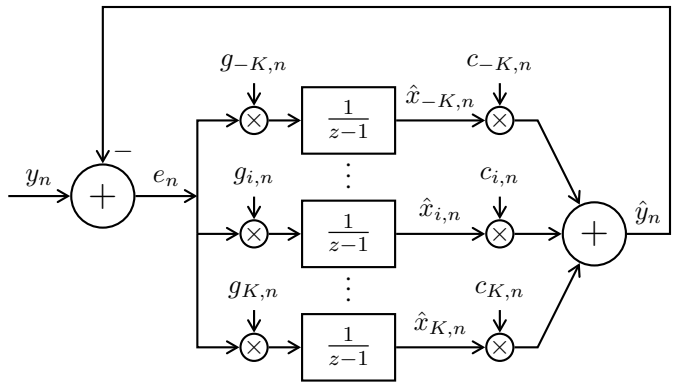


Fig. 2. Block diagram of the resonator-based observer, linear time-variant model. Notice the similarity between the structure of the observer and that of the conceptual signal model, shown in Figure 1. The observer, however, is extended by the time-varying $g_{i,n}$ coupling coefficients at the inputs of the integrators, and the common feedback.

The system equations of the observer are the following:

$$\hat{\mathbf{x}}_{n+1} = \hat{\mathbf{x}}_n + \mathbf{g}e_n = \hat{\mathbf{x}}_n + \mathbf{g}\left(y_n - \hat{y}_n\right), \tag{6}$$

$$\hat{\mathbf{x}}_n = \left[\hat{x}_{i,n}\right]^T \in \mathbb{C}^{N\times 1}, \qquad i = -K, \ldots, K, \tag{7}$$

$$\hat{y}_n = \mathbf{c}_n^T \hat{\mathbf{x}}_n, \tag{8}$$

$$\mathbf{g}_n = \left[g_{i,n}\right]^T \in \mathbb{C}^{N\times 1}, \qquad i = -K, \ldots, K, \tag{9}$$

$$g_{i,n} = \frac{\alpha}{N} c_{i,n}^*, \tag{10}$$

where $\hat{\mathbf{x}}_n$ is the estimated state vector, $\hat{y}_n$ is the estimate of the signal value and e_n is the error of the estimation.

The coupling vector $\mathbf{g}_n$ is the product of the observer gain α/N, a tunable parameter setting the dynamical behavior of the observer, and the coupling vector $\mathbf{c}_n$. The latter is a vector of complex roots of unity, setting the frequency response of the individual observer channels through their poles. If the coupling vectors are set according to (5) and (10), with $\alpha = 1$ and $f_1 = f_s/N$, a dead-beat observer is obtained. In this case, the transients of the observer settle in at most N steps and it produces the recursive DFT of the input signal afterwards [3].

C. On Dead-Beat Settling

The dead-beat property of the observer means the state variables of the observer converge to those of the conceptual signal model, and the error of estimation e_n becomes 0 in N (or fewer) steps. As mentioned in Section II-B, the Fourier analyzer possesses this property when the observer gain and the frequency of the fundamental harmonic are set appropriately. More generally speaking, the dead-beat nature of the observer relies on the set of the coupling vectors $\mathbf{c}_i^T$ and $\mathbf{g}_i$ constituting a biorthogonal system [2].

This corresponds to $\mathbf{C}^T = \mathbf{G}^{-1}$ with

$$\mathbf{C}^T = \left[\begin{array}{c} \mathbf{c}_1^T \\ \hline \mathbf{c}_2^T \\ \vdots \\ \mathbf{c}_n^T \end{array}\right], \quad \mathbf{G} = \left[\begin{array}{c|ccc} & & & \\ \mathbf{g}_1 & \mathbf{g}_2 & \cdots & \mathbf{g}_n \\ & & & \end{array}\right]. \tag{11}$$

The vectors $\mathbf{c}_i^T$ and $\mathbf{g}_i$ represent the values of the coupling coefficients corresponding to all channels at time instant i. In contrast, the column vectors of $\mathbf{C}^T$ and the row vectors of $\mathbf{G}$ contain the evolution of the coupling coefficients corresponding to a particular channel over an entire time period.

III. Modeling the Signal Structure

Let SM denote the set of positive harmonic indexes contained in the signal structure we intend to model: $SM = \{i_{s1}, i_{s2}, \ldots, i_{sm}\}$. Let w_i represent the complex amplitude of the signal structure component with index i: $w_i = A_i \cdot e^{\mathrm{j}\varphi_i}$. The time function of the signal structure we are modeling can then be expressed as:

$$y_{SM,n} = \sum_{i\in SM} 2 \cdot \mathrm{Re}\left\{w_i \cdot c_{i,n}\right\}, \tag{12}$$

since the $c_{i,n}$ coefficients form complex conjugate pairs.

We need to select the dominant harmonic of the signal structure. The harmonic with the highest signal-to-noise ratio is a good candidate. Let its index be i_{s1}.

In the following, we describe the procedures we considered focusing on the harmonics with positive indexes, i.e. frequencies $0 \le f_r \le f_s/2$. Since we are assuming a real-valued input signal, all suggested modifications need to be extended to the negative counterparts of the state variables and coupling coefficients concerned, being in a complex conjugate relationship with the corresponding positive ones.

A. The Intuitive Way of Modeling

As a first attempt, we can simply bind all other harmonics of the signal structure to the dominant harmonic. That is, after the state update of the observer as in (6), we adjust the state variables belonging to the signal structure model as follows:

$$\hat{x}_{i,n+1} = \hat{x}_{i_{s1},n+1} \cdot \frac{w_i}{w_{i_{s1}}}, \quad i \in SM,\ i \ne i_{s1}. \tag{13}$$

This method, however, excludes all bound signal structure harmonics from the common feedback loop in Figure 2.

As a result, we obtain an observer that no longer provides dead-beat settling. However, reasonable tracking performance

can be expected if we first let the original observer structure converge before activating the signal structure model described by (13). By design, the behavior of the signal structure model is entirely governed by the dominant harmonic.

An output waveform can be seen in Figure 3a.

B. Modifying the Basis Structure

As an attempt to improve the model, we modified the coupling vector corresponding to the dominant harmonic in such a way that it generates the whole signal structure on its own. In order to do this, we modified the appropriate column vector[1] of the coupling matrix $\mathbf{C}^T$ defined in (11):

$$\mathrm{col}_{i_{s1},\mathrm{new}}\,\mathbf{C}^T = \sum_{i\in SM} w_i \cdot \mathrm{col}_i\,\mathbf{C}^T. \tag{14}$$

The corresponding $\mathbf{g}$ coupling vectors can then be obtained as $\mathbf{G} = \left(\mathbf{C}^T\right)^{-1}$. It can be shown that the resulting values of the modified vectors can be expressed as

$$\mathrm{row}_{i_{s1},\mathrm{new}}\,\mathbf{G} = \frac{1}{w_{i_{s1}}} \cdot \mathrm{row}_{i_{s1}}\,\mathbf{G}, \tag{15}$$

$$\mathrm{row}_{i,\mathrm{new}}\,\mathbf{G} = -\frac{w_i}{w_{i_{s1}}} \cdot \mathrm{row}_{i_{s1}}\,\mathbf{G} + \mathrm{row}_i\,\mathbf{G},\ i \in SM,\ i \neq i_{s1}.$$

This yields a convergent observer with dead-beat properties by design (see Section II-C). However, as seen from (15), the behavior of the state variable carrying the signal structure model is determined by the dominant harmonic only.

An example output waveform is shown in Figure 3b.

Note. If we disable the non-dominant harmonics of the signal structure model by setting

$$\mathrm{col}_{i,\mathrm{new}}\,\mathbf{C}^T = \mathbf{0}, \quad i \in SM,\ i \neq i_{s1}, \tag{16}$$

we get a behavior matching that of the "intuitive way" described in Section III-A.

C. All Harmonics Contributing to the Signal Structure Model

By modifying the $\mathbf{C}^T$ matrix according to (14), the output of the channel carrying the modified dominant harmonic will contain all signal structure model harmonics with the amplitude and phase relations prescribed by the w_i coefficients.

We also found it desirable that all signal structure harmonics contribute to the input of this channel proportionally to their respective weights w_i. We achieved this by setting

$$\mathrm{row}_{i_{s1},\mathrm{new}}\,\mathbf{G} = \frac{1}{|SM|} \sum_{i\in SM} \frac{1}{w_i} \cdot \mathrm{row}_i\,\mathbf{G}. \tag{17}$$

By scaling with the reciprocal of the cardinality (number of elements) of the set SM, we maintain

$$\mathrm{row}_{i_{s1},\mathrm{new}}\,\mathbf{G} \cdot \mathrm{col}_{i_{s1},\mathrm{new}}\,\mathbf{C}^T = 1. \tag{18}$$

However, apart from the dominant harmonic, all signal structure harmonics are now coupled into two signal channels.

[1]The ith column and row of $\mathbf{M}$ are referred to by $\mathrm{col}_i\,\mathbf{M}$ and $\mathrm{row}_i\,\mathbf{M}$, respectively. The suffix new always indicates the new value to be assigned to the vector in question.

As a result, all harmonics of interest contribute to the signal structure model. However, dead-beat settling is not preserved and convergence becomes slower. By disabling the non-dominant signal structure harmonics according to (16), convergence can be accelerated. Since the basis vectors no longer span $\mathbb{C}^{N\times N}$, dead-beat behavior is not restored nevertheless. Figure 3c shows an output waveform.

D. A New Basis for the Subspace of the Signal Structure

We then aimed to restore the advantageous properties of the observer while keeping a single signal channel representing the entire signal structure, with all corresponding harmonics coupled onto its input. Thus we transformed the basis vectors of the subspace of $\mathbb{C}^{N\times N}$ spanned by the basis vectors contained in the signal structure model: $\left\{\mathrm{col}_i\,\mathbf{C}^T \mid i \in SM\right\}$.

We started by modifying the column corresponding to the dominant harmonic in the coupling matrix $\mathbf{C}^T$ according to (14). At this point, our goal was to transform the remaining basis vectors of the signal structure model into an orthogonal set, including the previously modified basis vector.

For the transformation, we first considered the Gram-Schmidt process [4]. Although easy to implement, the method has numerical problems, so we ended up resorting to QR decomposition by Householder reflections [5]. The leftmost columns of the resulting unitary matrix are then a set of orthonormal basis vectors spanning the subspace of $\mathbb{C}^{N\times N}$ corresponding to the signal structure model, with the first one being parallel (proportional) to the vector carrying the signal structure model we calculated earlier using (14).

Once more, the corresponding $\mathbf{g}$ coupling vectors can then be obtained as $\mathbf{G} = \left(\mathbf{C}^T\right)^{-1}$.

This yields an observer with attractive properties by design:

- The observer is convergent with dead-beat settling.
- All harmonics involved are proportionally represented in the channel carrying the signal structure model.

It still needs to be determined how the values of the other channels resulting from the QR decomposition relate to the discrepancy between the model and the actual signal.

An output waveform is shown in Figure 3d.

IV. Conclusions

We suggested several ways to model a signal structure within a Fourier analyzer. More detailed analysis will be required to ascertain whether these methods offer any actual advantage in signal detection applications.

References

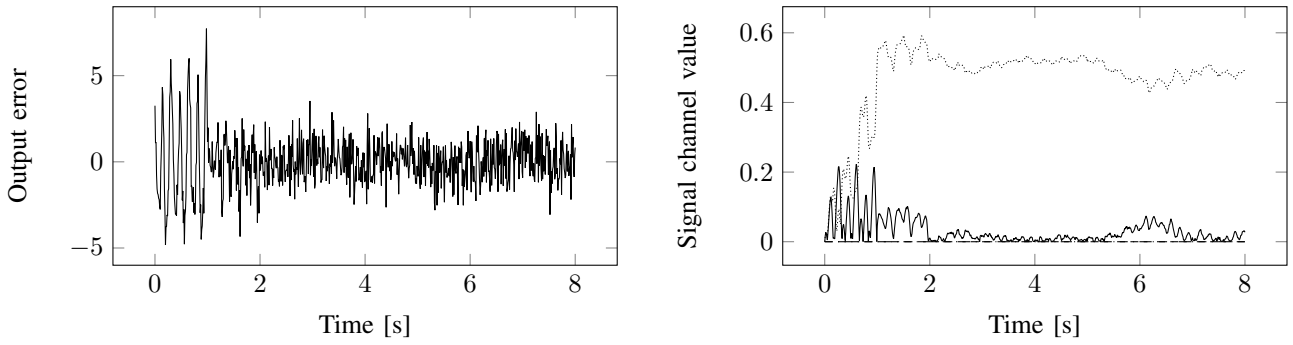
[1] G. H. Hostetter. Recursive discrete Fourier transformation. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 28(2):184–190, Apr 1980.

[2] G. Péceli. A common structure for recursive discrete transforms. *IEEE Transactions on Circuits and Systems*, 33(10):1035–1036, Oct 1986.

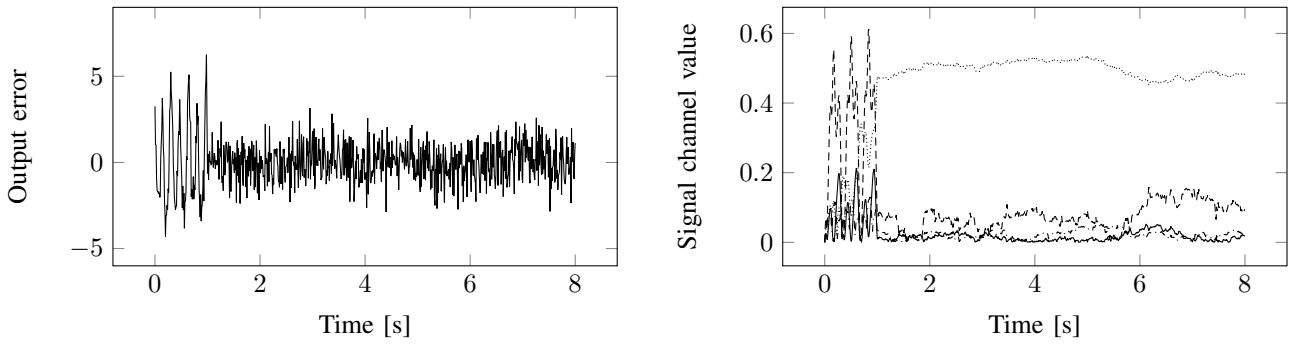
[3] Gy. Orosz, L. Sujbert, and G. Péceli. Analysis of resonator-based harmonic estimation in the case of data loss. *IEEE Transactions on Instrumentation and Measurement*, 62(2):510–518, Feb 2013.

[4] E. W. Cheney and D. R. Kincaid. *Linear Algebra: Theory and Applications*. Jones and Bartlett Publishers, 2009.

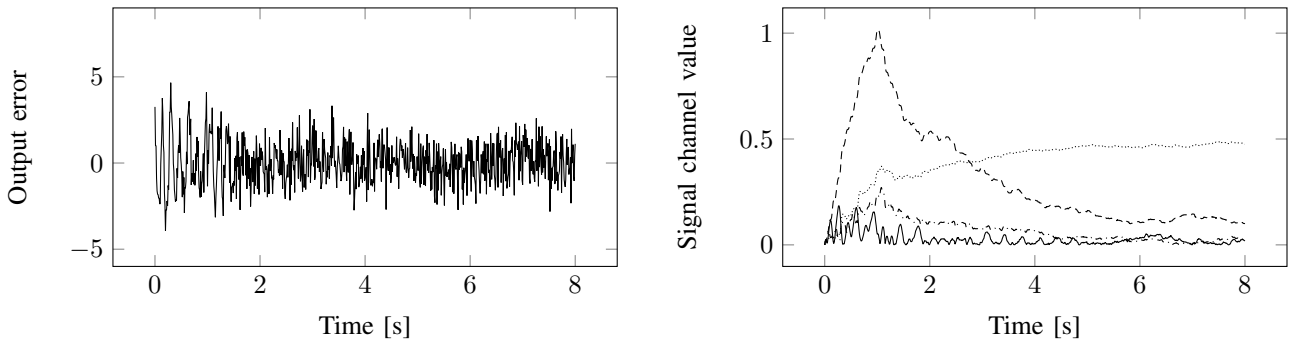
[5] A. S. Householder. Unitary triangularization of a nonsymmetric matrix. *Journal of the ACM (JACM)*, 5(4):339–342, 1958.



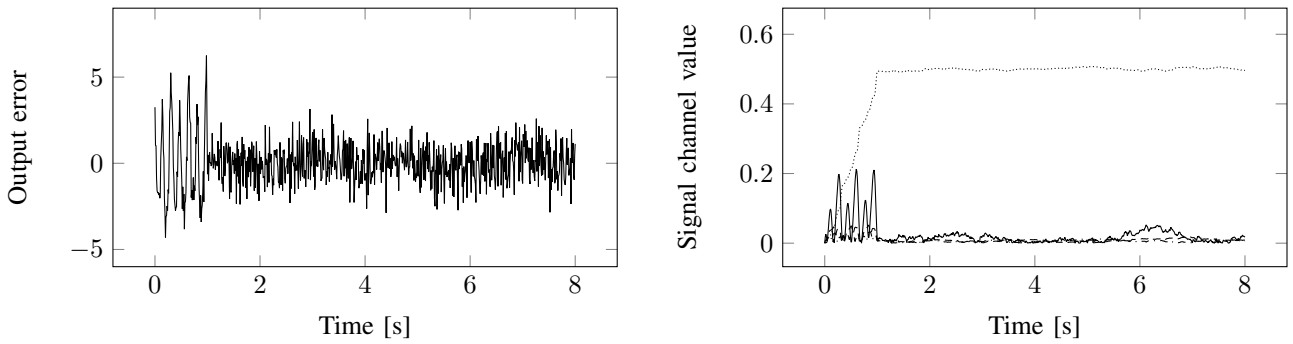
(a) The "intuitive way". The observer converges in this case, but dead-beat settling is not ensured.



(b) The dead-beat observer obtained through the first attempt at modifying the basis vectors.



(c) The signal structure model is incorporated into both the corresponding $\mathbf{g}$ and $\mathbf{c}^T$ vectors. The result is slower convergence.



(d) The dead-beat observer obtained by tayloring the basis vectors using QR decomposition.

Fig. 3. Output error and signal channel values obtained with different signal structure model realizations. The parameters of the signal structure model match those of the test signal: $y(t) = \cos(2\pi \cdot f_d t) + 3 \cdot \cos(2\pi \cdot 2 f_d t + 1) + 0.7 \cdot \cos(2\pi \cdot 4 f_d t + 2)$. The test signal is contaminated with white noise, $\mathrm{SNR} = 10\,\mathrm{dB}$. For illustrative purposes, the dominant harmonic was not selected following the guideline in Section III: $f_d = 3\,\mathrm{Hz}$. In certain cases, this results in reduced performance of the signal structure model.

Sustainable Management of Versioned Data

Martin Häusler, Ruth Breu (Supervisor)
University of Innsbruck
Institute for Computer Science
6020 Innsbruck, Austria
Email: martin.haeusler@uibk.ac.at

***Abstract*—Efficient management of versioned (temporal) data is an important problem for a variety of business applications that require traceability of changes, auditing, or history analysis. However, practical concerns regarding scalability and sustainability that arise when running a versioned system for extended periods of time often remain unadressed. In this paper, we present our approach for solving these issues in our implementation prototype[1]. We state the requirements, provide an overview over the algorithm and a comparison with related work alongside a discussion of individual advantages, disadvantages and challenges.**

I. Introduction

Working with versioned data, sometimes also referred to as *temporal* data, is an old problem. Early work dates back to 1986 when Richard Snodgrass presented his article *Temporal Databases* [1]. Since then, several authors have proposed various approaches [2][3][4] to efficiently solve the problem. In 2011, temporal features were introduced officially in the SQL Standard [5] which emphasizes the demand for such technology in practice. In 2016, we published a paper [6] that presents a novel approach to versioning using a matrix–based formalization, which will be summarized in Section II. This approach improved on the state of the art by offering equivalent "point-in-time" query performance on all versions, a compact format that maximizes sharing of unchanged data among versions and formalized operation semantics. Even though all of the mentioned publications offer solutions for the management of temporal data, there is one key aspect that has not yet been covered yet, which is the problem of building a system for temporal data management that can not only provide acceptable performance in short-term laboratory conditions, but can actually run for several years in a deployment, and is capable of dealing with the large volume of data accumulated over time via versioning. This issue is essential and even more prevalent than in non-versioned systems, because a deletion that would usually reduce the data volume actually results in a new version being created, increasing the number of managed elements instead. A similar line of argumentation holds for modifications. This sustainability problem is the main focus of this paper. We propose a new management technique for temporal data that expands upon our previous work and discuss its implications to implementation complexity and performance.

[1]This work was partially funded by the research project "txtureSA" (FWF-Project P 29022).

The remainder of this paper is structured as follows. In Section II we provide an overview of our past work and related concepts. Section III introduces the problem of sustainable management for versioning data and presents the individual requirements in detail. In Section IV we outline our proposed solution and elaborate on its implications. After comparing our approach with related work in Section V, we provide an outlook to our future work in Section VI and conclude the paper with a summary in Section VII.

II. Technical Background

In his paper *Efficient indexing for constraint and temporal databases* [2] from 1997, Sidhar Ramaswamy stated that the data versioning problem "is a generalization of two dimensional search which is known to be hard to solve". In our experience, the complexity of the problem further increases with the expressivity of the chosen data model. For that reason, we decided to choose a simple basic format and then transform more sophisticated structures into that format. The basic format of choice here is a Key–Value Store that is enhanced with temporal information. This is the foundation for our *ChronoDB*[2] project [6]. This Temporal Key–Value Store operates on triples

$$Entry := < t, k, v >$$

... where t is a Unix–style 64bit timestamp, k is an arbitrary string that acts as the key, and v is the value associated with the key, which is a byte array of arbitrary length greater zero. In contrast to other formalizations (e.g. [2]) our approach does not involve *validity ranges* in terms of lower and upper timestamp bounds. Instead, in our case the validity range is *implicit* – any given entry is valid until a new entry with the same key is inserted that has a higher timestamp, "overriding" the previous entry. This can be visualized as a matrix, as shown in Figure 1.

This figure shows the process of retrieving the value for key d at timestamp 5. The algorithm first tries to find the entry at the exact location, and then back-tracks and checks older timestamps. The first value encountered during this search is returned. This works because an entry that did not change from one timestamp to the next is assumed to have remained unchanged. Since entries, once written into the database, are

[2]Available on GitHub: https://github.com/MartinHaeusler/chronos/tree/master/org.chronos.chronodb

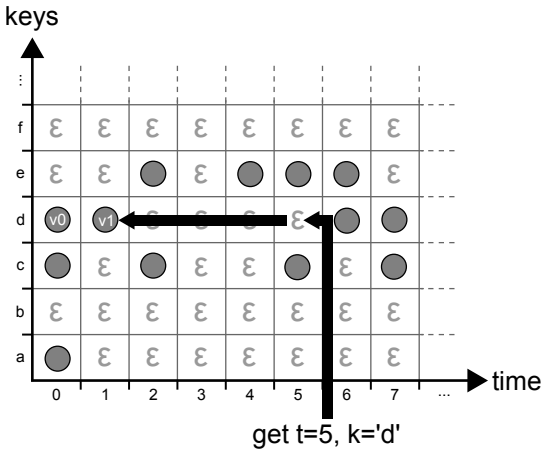


Fig. 1. Performing a *get* operation on a Temporal Data Matrix.

assumed to be immutable, copying their data is not necessary, thus maximizing sharing of unchanged data between versions. Furthermore, the fact that upper bounds of validity ranges are implicit in this structure, data that was once written to disk never needs to be modified again, allowing the implementation to be a true *append only* store. A modification or insertion both result in the execution of a *put* operation that inserts new entries. To ensure consistency of the history, new entries may only be added *after* the last entry on the time dimension, i.e. modification of the past is prohibited.

TABLE I
ASCENDING TEMPORAL KEY ORDERING BY EXAMPLE

Order	Temporal Key	User Key	Timestamp
0	a@0123	a	123
1	a@0124	a	124
2	a@1000	a	1000
3	aa@0100	aa	100
4	b@0001	b	1
5	ba@0001	ba	1

On the implementation side, our prototype ChronoDB relies on a B^+-Tree structure [7] for its primary index, which is a regular B-Tree where the leaf nodes in addition form a linked list for fast neighbor search. In this index, the key (which we refer to as *temporal key*) is a string consisting of the original user key, a separator, and the timestamp of insertion (left-padded with zeros to give equal length to all timestamps). The tree is ordered by lexicographic ordering on the key, resulting in the order displayed in Table I. This ordering is critical for the implementation, because executing a temporal *get* operation as depicted in Figure 1 is now equivalent to finding the temporal key where the user key is identical to the given one, and the timestamp is either equal to or the *next-lower* contained timestamp compared to the request timestamp. The B^+-Tree structure allows to perform this query efficiently with time complexity $O(\log n)$.

III. Problem Description & Requirements

In practice, a versioned data store may be deployed in a server backend, which implies usage 24 hours a day, potentially for several years. Even under the assumption that most accesses will be read-only, the amount of data stored in the database will inevitably increase over time, because every modification and even delete operation will result in an increase in data volume as new versions need to be managed. Even though our current implementation, which stores all data in a single B^+ tree, scales well to 100.000 entries and beyond, it will inevitably reach its limits when being operated for several years. In order to support this use case, the following criteria must be met:

- **R1: Support for an extended number of versions**
  As many versions will accumulate over years, the scalability limits of a single B-Tree structure will eventually be exceeded, causing greatly increased query times at best and system crashes at worst. The system must not be constrained by such limitations and offer support for a virtually unlimited number of versions.
- **R2: Support for efficient file-based backup**
  In server backend scenarios, the contents of a database are often backed up by automated tools at regular intervals. In order to reduce the load of such tools, the database must be segmented into files such that most files remain the same when new data is being added, allowing effective detection of unchanged files via checksums.
- **R3: Support for efficient access to old versions**
  Queries that request older versions of data should not be inherently slower than queries operating on recent versions. In particular, a linear correlation between the age of a requested version and resulting query performance must be avoided.

Together, those requirements describe the environment in which our database will need to operate. In the next section, we will outline our solution and how it meets those requirements.

IV. Proposed Solution

The primary requirement, as outlined in Section III, is the ability to support a virtually unlimited number of versions **[R1]**. Also, we must not store all data in a single file, and old files should ideally remain untouched when inserting new data **[R2]**. For these reasons, we must not constrain our solution to a single B-Tree. The fact that past revisions are immutable in our approach coincides with requirment **[R2]**, so we decided to split the data along the time axis, resulting in a series of B-Trees. Each tree is contained in one file, which we refer to as a *chunk file*. An accompanying *meta file* specifies the time range which is covered by the chunk file. The usual policy of ChronoDB is to maximize sharing of unchanged data as much as possible. Here, we deliberately introduce data duplication in order to ensure that the initial version in each chunk is complete. This allows us to answer *get* queries within the boundaries of a single chunk, without having to navigate to the previous one. As each access to another chunk has CPU and

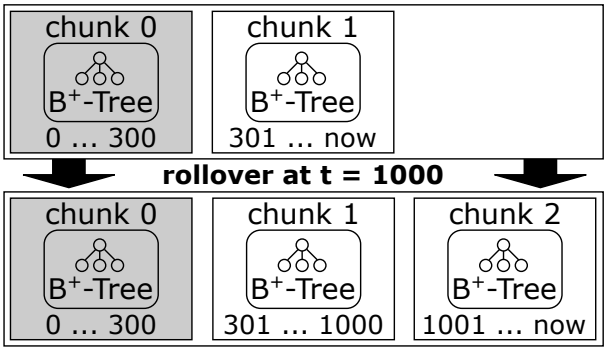


Fig. 2. The temporal rollover process by example

I/O overhead, we cannot afford to access more than one chunk to answer a basic query. Without duplication, accessing a key that has not changed for a long time could potentially lead to a linear serach through the chunk chain, violating requirement **[R3]**.

The algorithm for the "rollover" procedure outlined in Figure 2 works as follows.

Algorithm 1: Temporal Rollover

Data: The data chunk containing the "head" revision
Result: An archive chunk and a new "head" chunk

```
1 time ←getLastTimestamp(headChunk);
2 head ←getHeadRevisionFromChunk(headChunk);
3 setValidTo(headChunk, time);
4 newHeadChunk ←createEmptyChunk(time + 1);
5 insertEntriesIntoChunk(head, newHeadChunk);
6 updateTimeRangeLookup();
```

In Line 1 of Algorithm 1, we fetch the latest timestamp where a commit has occurred in our current head revision chunk. Next, we calculate the full head version of the data in Line 2. With the preparation steps complete, we set the end of the validity time range to the last commit timestamp in Line 3. This only affects the metadata, not the chunk itself. We now create a new, empty chunk in Line 4, and set the start of its validity range to the split timestamp plus one (as chunk validity ranges must not overlap). The upper bound of the new validity range is infinity. In Line 5 we copy the head version of the data into the new chunk. Finally, we update our internal lookup table in Line 6. This entire procedure only modifies the last chunk and does not touch older chunks, as indicated by the grayed-out boxes in Figure 2.

The lookup table that is being updated in Algorithm 1 is a basic tree map which is created at startup by reading the metadata files. For each encountered chunk, it contains an entry that maps its validity period to its chunk file. The periods are sorted in ascending order by their lower bounds, which is sufficient because overlaps in the validity ranges are not permitted. For example, after the rollover depicted in Figure 2, the time range lookup would contain the entries shown in Table II.

TABLE II
TIME RANGE LOOKUP

Time Range	Chunk Number
$[0 \dots 300]$	0
$[301 \dots 1000]$	1
$[1001 \dots \infty]$	2

We employ a tree map specifically in our implementation for Table II, because the purpose of this lookup is to quickly identify the correct chunk to address for an incoming request. Incoming requests have a timestamp attached, and this timestamp may occur exactly at a split, or anywhere between split timestamps. As this process is triggered very often in practice and the time range lookup map may grow quite large over time, we are considering to implement a cache based on the least-recently-used principle that contains the concrete resolved timestamp-to-chunk mappings in order to cover the common case where one particular timestamp is requested more than once in quick succession.

With this algorithm, we can support all three of our previously identified requirements (see Section III). We achieve support for a virtually unlimited number of versions **[R1]** because new chunks always only contain the head revision of the previous ones, and we are always free to roll over once more should the history within the chunk become too large. We furthermore do not perform writes on old chunk files anymore, because our history is immutable **[R2]**. Regardless, thanks to our time range lookup, we have close to $O(\log n)$ access complexity to any chunk **[R3]**, where n is the number of chunks, with the additional possibility of caching frequently used timestamp-to-chunk mappings.

This algorithm is a trade-off between disk space and scalability. We introduce data duplication on disk in order to provide support for large histories. The key question that remains is *when* this process happens. We need a metric that indicates the amount of data in the current chunk that belongs to the history (as opposed to the head revision) and thus can be archived if necessary by performing a rollover. We introduce the *Head–History–Ratio* (HHR) as the primary metric for this task, which we defined as follows:

$$HHR(e,h) = \begin{cases} e, & \text{if } e = h \\ \frac{h}{e-h}, & \text{otherwise} \end{cases}$$

... where e is the total number of entries in the chunk, and h is the size of the subset of entries that belong to the head revision (excluding entries that represent deletions). By dividing the number of entries in the head revision by the number of entries that belong to the history, we get a proportional notion of how much history is contained in the chunk that works for datasets of any size. It expresses how many entries we will "archive" when a rollover is executed. When new commits add new elements to the head revision, this value increases. When a commit updates existing elements in the head revision or deletes them, this value decreases. We can employ a threshold as a lower bound on this value

to determine when a rollover is necessary. For example, we may choose to perform a rollover when a chunk has an HHR value of 0.2 or less. This threshold will work independently of the absolute size of the head revision. The only case where the HHR threshold is never reached is when exclusively new (i.e. never seen before) keys are added, steadily increasing the size of the head revision. However, in this case we would not gain anything by performing a rollover, as we would have to duplicate all of those entries into the new chunk to produce a complete initial version, therefore the HHR metric is properly capturing this case by never reaching the threshold, thus never indicating the need for a rollover.

V. Discussion & Related Work

Even though data versioning has been recognized as an important problem in academic literature for many years, the sustainability of the versioning process itself is rarely discussed in related work. For example, in the SQL-based *ImmortalDB* system [4], every time an entry overrides another one, a pointer is created from the current to the previous version, forming a history chain. The resulting system offers good performance on the head revision and recent versions, but the access times increase linearly as the request timestamp is moved further back in time, because the history chains need to be traversed linearly.

Felber et al. propose a similar solution [3] for key-value stores. Here, each key refers to a list of value versions. Again, the search for older entries is linear. This approach furthermore suffers from the loss of coherence information - the second entry in the list for key a may have existed at the same time as the tenth entry for key b, but this information is lost.

The solution proposed by Ramaswamy [2] does not suffer from any of these drawbacks and is conceptually much closer to our solution. Ramaswamy proposes to have validity intervals for key-value pairs and perform the search over these intervals. In the paper, the drawback of having to update the upper bounds of the validity ranges is clearly explained, which is the main difference to our versioning approach. However, Ramaswamy does not propose a concrete implementation, and therefore does not touch upon the issue of sustainability.

Our solution offers an access time to any version of any key in $O(\log c + \log e)$ accesses, where c is the number of chunks and e is the number of entries in the target chunk. If we assume a cache with $O(1)$ access time for the most frequently used request timestamps, the complexity is reduced to $O(\log e)$. We would like to emphasize that e is the number of entries in a single chunk, and therefore only represents a fraction of the overall history. Even without such a cache, with a reasonable choice for the HHR threshold, the number of chunks will always be significantly smaller than the total number of entries, and the resulting time range lookup table is small enough to be held in-memory, which further decreases the access times. In order to achieve these advantages, our approach relies on data duplication on disk, which increases the footprint of the database. In our case this duplication does not introduce synchronization issues, because only existing versions are duplicated, which are immutable in our system. The challenging part is the implementation itself, because the rollover process has to be implemented in a way that is safe against crashes and immediate program termination. For example, when a rollover is taking place and the program is terminated by an external signal, then at restart the database needs to be able to infer the correct steps for reaching a consistent state again. This recovery process in turn needs to be safe against crashes and needs to be repeatable an arbitrary number of times without ever leading to a non-recoverable state.

VI. Outlook & Future Work

Using the versioning concepts presented in this paper, we are implementing a versioned graph database compatible with the Apache TinkerPop API[3] which will soon be officially announced. Our final goal is to make use of this versioned graph database in order to implement a modern, efficient and feature–rich repository for EMF[4] models.

VII. Summary & Conclusion

In this paper, we presented our approach for a sustainable data versioning process. Based on our previous work [6] we employed a key-value format in order to keep the complexity of the versioning problem to a minimum. We presented a summary of our existing technology, and provided an overview of our efforts to transition the prototype into a software tool that can run for several years in a server backend and deal with the resulting large amounts of data. The method is based upon a *rollover* process, which takes the current (immutable) head revision and copies it into a new, empty data structure, where it will continue to evolve as new changes are applied. We also introduced a metric called *Head–History–Ratio* (HHR) which provides an estimate when a rollover should be executed based on the contents of the database. Finally, we compared our approach to existing related work and provided a discussion on the individual advantages, disadvantages and challenges.

References

[1] R. T. Snodgrass, "Temporal databases," *IEEE Computer*, vol. 19, pp. 35–42, 1986.

[2] S. Ramaswamy, "Efficient indexing for constraint and temporal databases," in *Database Theory-ICDT'97*. Springer, 1997, pp. 419–431.

[3] P. Felber, M. Pasin *et al.*, "On the Support of Versioning in Distributed Key-Value Stores," in *SRDS 2014, Nara, Japan, October 6-9, 2014*, 2014, pp. 95–104.

[4] D. Lomet, R. Barga *et al.*, "Immortal DB: Transaction Time Support for SQL Server," in *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '05. New York, NY, USA: ACM, 2005, pp. 939–941. [Online]. Available: http://doi.acm.org/10.1145/1066157.1066295

[5] ISO, "SQL Standard 2011 (ISO/IEC 9075:2011)," 2011.

[6] M. Haeusler, "Scalable versioning for key-value stores," in *DATA 2016 - Proceedings of 5th International Conference on Data Management Technologies and Applications, Lisbon, Portugal, 24-26 July, 2016.*, 2016, pp. 79–86. [Online]. Available: http://dx.doi.org/10.5220/0005938700790086

[7] B. Salzberg, *File Structures: An Analytic Approach*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988.

[3] https://tinkerpop.apache.org/

[4] https://www.eclipse.org/modeling/emf/

User-Defined Sandbox Behavior for Dynamic Symbolic Execution

Dávid Honfi, Zoltán Micskei
Department of Measurement and Information Systems,
Budapest University of Technology and Economics,
Budapest, Hungary
Email: {honfi,micskeiz}@mit.bme.hu

***Abstract*—Code-based generation of unit tests is an active topic of research today. One of the techniques is dynamic symbolic execution (DSE) that combines concrete executions with symbolic ones. The mature state of research in DSE allowed the technique to be transferred for industrial use. However, the complexity of software used in industrial practice have shown that DSE has to be improved in numerous areas. This includes handling dependencies to the environment of the unit under test. In this paper, we present a user-oriented approach that enables users to *provide input constraints and effects* for isolated environment dependencies in a parameterized sandbox. This sandbox collaborates with DSE, thus the isolated dependencies do not produce unrealistic or false behaviors.**

I. Introduction

Today, the demand for high quality software is greatly increasing. There are several techniques to improve quality, one of them is unit testing. A unit is a small portion of software, which can be a function, a class or even a set of classes. Most software projects use unit testing throughout the development lifecycle. Due to this fact, significant amount of time and effort is invested into unit testing.

Numerous ideas and techniques have already addressed to reduce this time and effort. Symbolic execution [1] is one of these techniques, as it is able to generate tests based on source code by using symbolic variables instead of concrete ones. During the symbolic execution process, each statement is interpreted and constraints are formed from the symbolic variables. When the execution has finished, these constraints can be solved using special solvers to obtain concrete input values. These inputs are used to execute the path on which the satisfied constraints were collected (path condition – PC). An advanced variant of this technique is dynamic symbolic execution (DSE) that combines concrete executions with symbolic ones in order to improve coverage [2], [3], [4]. The development of DSE has been progressing in the recent years in such a way that its industrial adoption became feasible. However, the technique faces several issues when it is used on large-scale programs with diverse behaviors [5], [6].

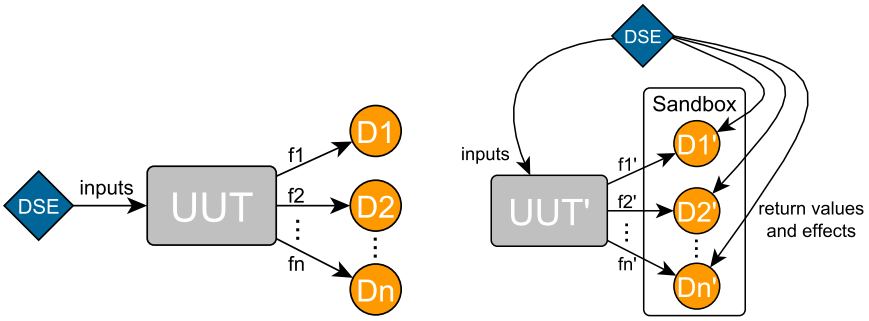
One of the most hindering issues is the isolation of dependencies. Without any isolation, – due to the concrete executions – the DSE test generation process would access the dependencies of the unit under test, which could cause undesirable events during test generation like accessing the file system, databases, or reaching parts of the software that are outside of the testing scope. To alleviate this, isolation (mocking) frameworks are commonly used in unit testing to handle the dependencies. Most of these frameworks however are not designed to be used with DSE-based test generation as the built-in low-level techniques in both may conflict with each other. To overcome this, we introduced a technique (based on a previous idea [7]) that is able to automatically handle the dependencies on the source code level by using transformations [8], [9]. These transformations replace invocations to the dependencies with calls to a parameterized sandbox generated with respect to the signatures of the methods being invoked. Then, the parameterized sandbox is used by DSE to provide return values and effects to objects resulting in the increase of coverage in the unit under test.

As the parameterized sandbox is filled with values by dynamic symbolic execution, the behavior depends on the underlying algorithm only. In most cases, DSE uses values that are relevant only to the code of the unit under test. This may cause important behaviors to be omitted for various reasons like 1) when the solver of the DSE is not able to obtain concrete values, or 2) when a combined behavior of two dependencies yields new behavior in the unit under test, or even 3) when the dependencies have some form of specification, which restricts the possible values. Also, the unrestricted behavior of the sandbox may result in false and unwanted behaviors for the unit under test.

In this paper, we address this issue by presenting an approach that builds on partial behaviors defined by developers to enhance the generated sandbox resulting in covering more relevant scenarios in the unit under test. The rest of the paper is organized as follows. Section II gives a detailed overview of the problem regarding the generated sandbox, along with presenting related concepts. Section III presents the approach with a detailed description of its process. In Section IV, a complex example is presented to provide better understanding of the approach. Section V summarizes our contribution.

II. Overview

Our approach for supporting DSE-based test generation with automated isolation consists of two main steps [9]: 1) transforming the unit under test and 2) generating a parameterized



(a) DSE without isolation.

(b) DSE with parameterized sandbox.

Fig. 1. The essence of the automatically generated parameterized sandbox approach.

sandbox. In step one, the abstract syntax tree of the unit is transformed using various rules, including the replacement invocations to dependencies with calls to the sandbox. Then, in step two, the sandbox is generated for the transformed invocations. The methods defined inside the parameterized sandbox receive the same *input* values as the original dependency would. Also, the generated methods may have *effects* on the passed objects, and may also *return* values given by DSE through the parameters of the sandbox. See Figure 1 and [9] for more details of the approach. The unit under test denoted with UUT uses n dependencies (D_k) through functions f_k. These dependencies also exist in the sandbox in a parameterized form, where the return values and effects are provided by the DSE.

An issue with the parameterized sandbox – that may hinder DSE from generating relevant cases – is that its behavior is uncontrolled, thus completely relies on the values generated by DSE. However, these values, in most of the cases are only relevant for covering more statements: their behaviors are not depending e.g., on 1) the state of the unit under test, 2) the state of other called dependencies, 3) their own state.

In order to overcome this issue, our approach employs inputs defined by the users of DSE, i.e. developers or testers. For the precise definition of input types we employed existing concepts (defined below) from related works on compositional symbolic execution [10].

Partition-effects pair [11]: A partition-effects pair (i, e) has 1) an input constraint i that is an expression over constants and symbolic variables, 2) an effects constraint e, which denotes expressions of symbolic variables bounded to values. A set of partition-effects pairs is used to describe the behavior of a method. A constraint i defines a partition of the input space and an effect e denotes the effects when the given argument values are found in the partition i.

Symbolic summary [12]: A symbolic summary of method m is a set of partition-effects pairs $m_{sum} = \{(i_1, e_1), (i_2, e_2), ..., (i_k, e_k)\}$, where $i_1, i_2, ..., i_k$ are disjoint.

An important aspect of compositional symbolic execution is to execute each unit separately and create method summaries to other units that use the currently analyzed one. The concepts introduced before can be employed for both static and dynamic compositional symbolic executions, although that requires the ability to execute the dependencies.

However in terms of DSE, this may not be possible for various reasons (e.g., no code/binary is available, DSE fails with traversal, dependencies are environments like file systems or databases). The previously presented automatically generated sandbox may be able to handle these cases, yet the behavior of the sandbox must be given somehow as no compositional execution is performed. Our approach is to involve developers and testers to provide those behaviors for the sandbox.

III. Approach

The basic idea of our approach is to obtain behaviors for the generated sandbox incrementally from developers and testers (the *users* of DSE-based test generation). These behaviors are defined in an easily readable format then transformed to symbolic method summaries with a preliminary logical check. When the check passes, source code is generated, which can be used by DSE as summaries for the corresponding execution paths.

Several other related works have already addressed the problematic area of isolation in symbolic execution-based test generation (e.g., [13], [14], [15]). These approaches also rely on user input at some point, however these employ various forms of user inputs differently than ours. Also, our approach uses incremental refinement by allowing the definition of partial behaviors as well.

This section uses an example to provide better understanding of our approach. The method `M(object o, bool b):int` used is a dependency of the unit under test. The real behavior of the method is out of scope for this example. Note that `object o` is passed by reference, thus its state can be affected in `M`.

Incremental refinement. Our approach uses incremental refinement of behavior (see Figure 2). At first, only the automated isolation transformation and sandbox generation are performed (0th iteration). After this step, the behavior of the sandbox depends only on DSE. The generated effects, input and output values for each dependency are presented to the user. Based on these, the user can decide to alter the behavior of each called dependency (1st iteration). The change of behavior is defined in a table structure (presented in Table I

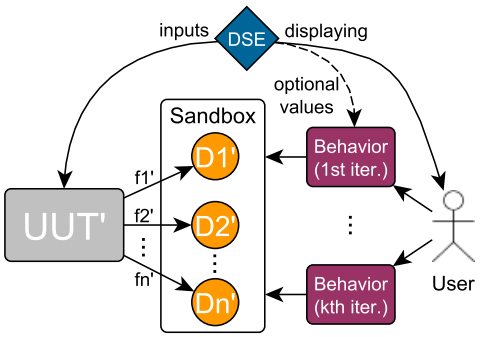


Fig. 2. The approach for user-defined sandbox behavior.

TABLE I
TABLE STRUCTURE FOR PROVIDING BEHAVIOR WITH TWO EXAMPLES.

Input (I)		Effects (E)	Return
o	b	o	
not null	true	new object()	>10
null	true	-	-1

for the example dependency method `M`). This table describes the constraints for each parameter, effect and for the return value as well. Also note that during the incremental usage, the user would see the values provided by DSE in each column.

In Table I, method `M` has two behaviors defined by the user overriding the arbitrary default. The first defines when parameter `o` is not null and parameter `b` is set to `true`, then `o` is assigned to a new object, and the method returns more than 10. The second states that when `o` is null and `b` is `true`, then the method returns constant -1. Note that in this example, the user provided only an under-approximation of the behavior as not all of the input partitions were covered (omitted where parameter `b` is `false`). For unspecified cases, our approach employs an `otherwise` feature, where both the effects and return value are set by the DSE.

Before the next execution, the automated isolation technique transforms the behavior (provided by the user) to source code into the sandbox. However, a validity check is performed before the code generation as the input partitions must be disjoint. In order to check this, our approach transforms the user-defined behaviors to symbolic summaries first, and then generates code from the summaries. The transformation from the table-structured user-defined behavior to the symbolic summaries can be defined as follows.

Transforming behaviors to symbolic summaries. A user-defined behavior b for method `m` consists of an input constraint $i \in I_m$, an effect constraint $e \in E_m$ and a return value r_m. Also, the behavior of method m can be described using multiple $b_k \in B_m$. First, it is checked that if $\bigwedge i_{b_k}$ for every k is unsatisfiable, i.e. checking if the input partitions defined by the user are disjoint, thus there are no conflicts between them. If the check passes, then from all $b_k \in B_m$ the

$$\bigvee_k (i_{b_k}, e_{b_k} \wedge r_{b_k})$$

symbolic summary is created.

Consider the example in Table I. The symbolic summary to be created here – based on the previous description – can be formulated as follows: $(o \neq null \wedge b == true, o == new\,object() \wedge return > 10) \vee (o == null \wedge b == true, return == -1)$.

The code generation for the sandbox is performed along predefined rules. These are designed to produce code that appends the symbolic summary to the path conditions of DSE. The input partition constraints are transformed to `if` statements and conditions, while the effects (as assignment statements) and return constraints are appended to the body of the sandbox method. Consider the example in Table I, the generated code for the sandbox method of method `M(object o, bool b):int` would be as follows.

Listing 1. Sandbox code for method m(object o, bool b):int.

```
public int FakeM(object o, bool b) {
  if(o != null && b == true) {
    o = new object();
    return DSE.ChooseFromRange<int>(11, int.Max);
  }
  if(o == null && b == true) {
    return -1;
  }
  return DSE.Choose<int>();
}
```

After the code generation has finished, the user is ready to re-execute the DSE test generation process to obtain new tests (this time with the user-defined sandbox behavior). In the next step of the refinement, the user gets the values generated by DSE again. The user can refine the behavior again (Figure 2 – nth iteration) by filling, appending or modifying the table previously introduced (Table I). This process can be repeated until the user finds the behaviors acceptable.

IV. COMPLEX EXAMPLE

The following example introduces an advanced scenario, where the user-defined behavior of the parameterized sandbox helps reducing the number of tests that trigger false behavior. The example is based on a backend service of an e-commerce site.

The current example focuses on a module of the service, which is responsible for handling advertisements. More precisely, the method under test (Listing 2) receives a product identifier and gets the advertisements (`Ads`) for that product. In its current implementation, the ads are consisting of related products only. The query of the related products is performed through database using the data access layer of the backend (`DB`). This layer fills in a set of integer identifiers of the related products. If there are related products, then the query returns `true`, while returns `false` if the set remained empty. The method under test adds the set of identifiers to the advertisement object to be returned. The addition causes an error if the set being passed is empty as it cannot occur in real use with respect to the specification.

In this example, the `DB` module is a dependency and has to be isolated during dynamic symbolic execution-based test generation in order to avoid unwanted accesses to the database.

Listing 2. Source code for complex example method.

```
public Ads GetAdsForProduct(int id) {
  if(id == -1) throw new NoProductExists();
  Set<int> ids = new Set<int>();
  bool success = DB.GetRelatedProducts(id, ids);
  if(success) {
    Ads ads = new Ads();
    ads.AddRelatedProducts(ids);
    return ads;
  }
  return null;
}
```

The first step of the incremental refinement is transforming the unit to use the parameterized sandbox. In the 0th iteration, the sandbox is only controlled by dynamic symbolic execution. The generated sandbox code would be as found in Listing 3.

Listing 3. Partial source code of the generated sandbox for advanced example.

```
bool FakeGetRelatedProducts(int id, Set<int> ids) {
  ids = DSE.Choose<Set<int>>();
  return DSE.Choose<bool>();
}
```

The issue with this unrestricted sandbox behavior is that there is no connection between the effects and the return value of the method. This causes a false behavior: the sandbox method can return `true` even when the set of identifiers remains empty causing the method under test to unintentionally crash. To tackle this issue, users can define the partial behavior as found in Table II.

TABLE II
TABLE STRUCTURE FOR PROVIDING BEHAVIOR WITH TWO EXAMPLES.

Input (I)		Effects (E)	Return
id	ids	id	
-	not null	not empty	true
-	null	-	false

Listing 4. Partial source code of the generated sandbox for advanced example

```
bool FakeGetRelatedProducts(int id, Set<int> ids) {
  if(ids != null) {
    for(int i = 0; i <= 10; i++) {
      ids.Add(DSE.Choose<int>(i));
    }
    return true;
  } else if(ids == null) {
    return false;
  }
}
```

The generated code for these restrictions modifies the parameterized sandbox as found in Listing 4. The code contains a loop with 10 iterations (predefined with sandbox loop settings) to fill in the set with identifiers and return `true` if the set was not null. Otherwise, the code returns `false`. There is no need for other branches as the input space is partitioned into two parts by the user-defined behaviors. Using this parameterized sandbox, the DSE will not generate test cases, where the set is empty, and the method returns `true`, thus avoids the problematic false behavior that would cause a known error. Note that this example can be continued with subsequent iterations of the incremental process to refine the behavior of the sandbox.

V. CONCLUSIONS

In this paper, we presented a user-oriented approach that extends our previous work [9]. Our results showed that an automatically generated parameterized isolation sandbox may improve DSE-based test generation. However, we were also able to identify that a fully parameterized sandbox – relying on values from DSE only – may 1) omit important behaviors or 2) cause false behavior to occur. To tackle this issue, we presented the idea of incremental user-defined behaviors in the parameterized sandbox. The users of DSE-based test generation are able to incrementally refine their behaviors defined for each dependency method in the isolated sandbox. In each refinement iteration, a corresponding code snippet is generated from the user-defined behaviors based on existing ideas from compositional symbolic execution. These generated code parts are used by DSE during test generation, thus are able to influence and steer the DSE process. This may reduce the false behaviors in the unit under test triggered by the parameterized sandbox.

REFERENCES

[1] J. C. King, "Symbolic execution and program testing," *Communications of the ACM*, vol. 19, no. 7, pp. 385–394, 1976.

[2] T. Chen, X. song Zhang, S. ze Guo, H. yuan Li, and Y. Wu, "State of the art: Dynamic symbolic execution for automated test generation," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1758 – 1773, 2013.

[3] K. Sen, "Concolic testing," in *Proc. of the Int. Conf. on Automated Software Engineering*. ACM, 2007, pp. 571–572.

[4] N. Tillmann and J. de Halleux, "Pex–White Box Test Generation for .NET," in *TAP*, ser. LNCS, B. Beckert and R. Hähnle, Eds. Springer, 2008, vol. 4966, pp. 134–153.

[5] C. Cadar and K. Sen, "Symbolic execution for software testing: three decades later," *Comm. of the ACM*, vol. 56, no. 2, pp. 82–90, 2013.

[6] X. Qu and B. Robinson, "A case study of concolic testing tools and their limitations," in *Proc. of Int. Symp. on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2011, pp. 117–126.

[7] N. Tillmann and W. Schulte, "Mock-Object Generation with Behavior," in *Proc. of Int. Conf. on Automated Software Engineering (ASE)*. ACM, 2006, pp. 365–366.

[8] D. Honfi and Z. Micskei, "Generating unit isolation environment using symbolic execution," in *Proc. of the 23rd PhD Mini-Symposium*. BUTE-DMIS, 2016.

[9] D. Honfi and Z. Micskei, "Supporting Unit Test Generation via Automated Isolation," *Periodica Polytechnica, Electrical Engineering and Computer Science*, 2017, Accepted. In press.

[10] S. Anand, P. Godefroid, and N. Tillmann, "Demand-Driven Compositional Symbolic Execution," in *TACAS*, ser. LNCS. Springer Berlin Heidelberg, 2008, vol. 4963, pp. 367–381.

[11] S. Person, M. B. Dwyer, S. Elbaum, and C. S. Păsăreanu, "Differential Symbolic Execution," in *Proc. of the 16th Int. Symp. on Foundations of Software Engineering*, 2008, pp. 226–237.

[12] P. Godefroid, "Compositional Dynamic Test Generation," in *Proc. of the 34th Symp. on Principles of Programming Languages*, 2007, pp. 47–54.

[13] S. J. Galler, A. Maller, and F. Wotawa, "Automatically Extracting Mock Object Behavior from Design by Contract Specification for Test Data Generation," in *Proc. of the Workshop on Automation of Software Test (AST)*. ACM, 2010, pp. 43–50.

[14] M. Islam and C. Csallner, "Dsc+Mock: A Test Case + Mock Class Generator in Support of Coding against Interfaces," in *Proc. of the 8th Int. Workshop on Dynamic Analysis*. ACM, 2010, pp. 26–31.

[15] B. Pasternak, S. Tyszberowicz, and A. Yehudai, "GenUTest: a Unit Test and Mock Aspect Generation Tool," *Int. Journal on STTT*, vol. 11, no. 4, pp. 273–290, 2009.

Towards Language Independent (Dynamic) Symbolic Execution

Stefan Klikovits*†, Manuel Gonzalez-Berges† and Didier Buchs*
*University of Geneva, Software Modeling and Verification Group
Geneva, Switzerland, Email: {firstname.lastname}@unige.ch
†European Organization for Nuclear Research (CERN), Beams Department
Geneva, Switzerland, Email: {stklikov,mgonzale}@cern.ch

***Abstract*—Symbolic execution is well-known for its capability to produce high-coverage test suites for software source code. So far, most tools are created to support a specific language. This paper elaborates on performing language independent symbolic execution and three ways to achieve it. We describe the use of one approach to perform dynamic symbolic execution using translation of a proprietary language and show the results of the tool execution on a real-life codebase.**

I. Introduction and Background

Software testing is a popular technique amongst software developers. The recent advance of computing power increased the usability of dynamic symbolic execution (DSE) [1] to a point where it is now even included in commercial tools [2]. DSE implementations are mostly designed to support one specific language (e.g. Java, C), or their underlying low-level language (e.g. x86 byte code). While the advantages of having such tools are indisputable, software developers working with proprietary or less popular languages often cannot benefit. Our research focuses on proposing a solution for languages that do not have dedicated (D)SE tools. The drive for our explorations comes from a practical application at CERN, which uses a proprietary scripting language to control and monitor large installations.

A. Motivation

The European Organization for Nuclear Research (CERN) uses a proprietary software called Simatic WinCC Open Architecture (WinCC OA) to control its particle accelerators and installations (e.g. the electrical power grid). To support the design of such systems the BE-ICS group maintains and develops the *Joint COntrols Project* (JCOP), as set of guidelines and software components to streamline developments. JCOP is based upon the WinCC OA SCADA platform which is scriptable via *Control* (CTRL), a proprietary programming language inspired by ANSI C.

Until very recently CTRL code did not have a dedicated unit test framework. The development of such a testing library filled this need, but after over a decade of development the CTRL code base of JCOP sizes some 500,000 lines of code (LOC). This code has to be manually (re-)tested during the frequent changes of operating system versions, patching or for framework releases. Over the decades-long lifetime of CERN's installations, this testing is repeatedly (often annually) required and involves a major overhead. To overcome this issue, the use of automatic test case generation (ATCG) was decided.

B. Symbolic Execution

Symbolic execution (SE) is a whitebox ATCG methodology that analyses source code's behaviour. The approach works by constructing the code's execution tree. The program's input parameters are replaced with symbolic variables, their interactions recorded as operations thereon. The conjunction of all symbolic constraints of an execution tree branch, provides a *path constraint*. Finding a solution for it, e.g. using a satisfiability modulo theories (SMT) solver, provides inputs that will lead the program to follow this branch. SE experiences shortcomings when it comes to uninstrumentable libraries, impossible/infeasible constraints (modulo, hashes) or too complex constraints. To overcome these limitations, dynamic symbolic execution has been introduced. DSE switches to concrete value execution in cases where SE reaches its limits. We refer the reader to [1] for an overview of SE and DSE.

This paper is organised as follows: Sect. II introduces language independent SE. Sect. III describes an implementation to bring SE to CTRL. Sect. IV presents results of the implemented solution. Sect. V explores related work and Sect. VI concludes.

II. Language Independent Symbolic Execution

Most SE and DSE tools operate on low-level (LL) representations (LLVM, x86, .NET IL) of popular high-level languages (Java, C, C#). The reason is that LL representations are simpler, leading to fewer operations types that have to be treated by the symbolic execution engine. This means that only source code which compiles into these LL representations is supported. Other languages miss out on the functionality. Another disadvantage of using a specific low-level representation are implicit assumptions on data types. Only data types supported by the operating language are supported. This voids the possibility to use own or modified data types and languages.

For this reason we propose the use of language independent symbolic execution. Apart from the development of a dedicated (D)SE engine for a programming language, there

exist three possibilities for language independent symbolic execution.

First, a SE tool that operates on a generic abstract syntax tree (AST). SE already performs operations on symbolic variables, which are either subtypes of or proxies for real data types. Hence, a tool operating on a model of the source code, i.e. an AST, could perform truly language independent SE. This approach comes with two difficulties: a) specifying an AST generic enough for all targeted languages and their particular features and differences, b) deterministically translating a language parser's representation into the generic AST.

The second approach is a symbolic execution engine based on callback functions. The source code under test is parsed using a modified version of its native parser. The parser triggers actions in the symbolic execution tool when it comes across the variable interactions. The advantage of this approach is that existing parsers for the source language can be adapted to issue the required calls, leading to an easily implemented SE framework. The caveat is a large number of messages being issued by the parser, potentially leading to low performance.

Lastly, a translation into the operating language of an existing tool. This solution, while similar to the first, is distinct in that usually the target language and the tool cannot be modified. The target language has to offer similar concepts as the source, otherwise only a subset of the sources can be supported without major overhead. It is also necessary to re-implement any standard-library or built-in functionality that the source language relies on. A difficulty arises when the target language has different semantics or data types, requiring trade-offs and leading to unsupported features.

A. Semantics

The semantics of a language play an important role. In all three approaches, it is necessary to precisely capture the meaning of each statement. Failure to do so would lead to divergences and hence wrongly produced constraints and input data. As an example one can look at differences of zero- and one-based list/array indices. This minor change in the language semantics can lead to severe errors such as *out of bounds*-errors, changed loop behaviour, and similar.

It is also important that the source language's data types are supported by the symbolic execution tool. Some existing (D)SE tools such as Microsoft Pex [3] support the use of own data types (*classes*). However, these class data types are treated differently from native types (they are `nullable`) and sometimes replaced by stubs. Additionally, existing tools do not allow for the modification of their native data types, leading to unsupported features. An example would be that some languages (e.g. JavaScript) support native implicit casting between `int` and `string` values. This behaviour cannot be reproduced in C# and is hence not supported by Pex.

A solution would be to define all necessary *sorts* (data types) for the underlying SMT solvers (e.g. Z3, CVC4). Current tools adjust the SMT solver configuration based on their operation language. An SE framework that supports the specification and use of user-defined sorts can provide a solution to these problems. Alternatively, it is possible to abandon SMT solvers and explore other ways of solving constraints. Term rewriting systems are well known for their capability to express semantics and offer constraint solving capabilities. This solution, while offering less performance, provides more flexibility and support for native data types.

Using term rewriting systems, it would also be possible to perform SE/DSE on generic models, opening the door to many different kinds of analyses.

III. Implementation

Since no tool exists that natively supports CTRL code, CERN is faced with the choice between three solutions: 1) develop an ATCG tool specifically for CTRL; 2) develop a language independent ATCG tool; 3) translate the CTRL code into the operating language of an existing ATCG tool.

Given CERN's practical need, the last option was chosen. A short evaluation led to Microsoft Research's Pex tool [3], a program that performs test case generation through DSE. In order to use Pex for CTRL code, we developed a tool called *Iterative TEst Case* system (ITEC). ITEC translates CTRL to C#, in order to execute Pex and obtain input values for automatically generated CTRL test cases. This tool helped to build up regression tests that can then be reused on the evolving system to ensure its quality. ITEC works on the assumption that the current system reached a stable state after 13 years of continuous development and use.

A. Architecture

CERN's focus mainly lies in the test case creation for CTRL code. Hence, in a first step, the re-use of existing software is preferred over the creation a generic ATCG framework. ITEC relies heavily on an existing CTRL parser, which has been implemented in Xtext [4] during a previous project at CERN. ITEC's workflow is separated into six consecutive steps: 1) code under test (CUT) selection[1] 2) semi-purification 3) C# translation 4) test input generation 5) test case creation 6) test case execution.

ck] (start) – (one);

In the initial task, the CUT selection, the user or the tool automatically (in bulk execution mode) chooses which code is to be tested. ITEC analyses the sources for dependencies (global variables, database values, subroutines) which are to be replaced, if necessary.

Listing 1. SP doubles: Before

```
get15OrMore(x){
  a = dependency(x)
  return a > 15 ? a : 15
}
dependency(x){
  r = randomValue() + 5
  return r * x
}
```

Listing 2. SP doubles: After

```
get15OrMore(x, b){
  a = dependency(x, b)
  return a > 15 ? a : 15
}
dependency(x, b){
  observe(x)
  return b
}
```

[1]we use to *code* under test instead of *system* under test, as we focus on individual functions or code segments rather than systems

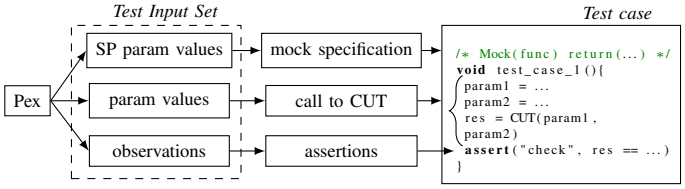


Fig. 1. Test case generation from Pex output

Following the CUT identification, a process called *semi-purification* (SP) [5] is applied. Semi-purification replaces a CUT's dependencies with additional input parameters. SP is required as CTRL, similar to most other procedural languages, does not support mocking. The result of SP is a modified version of the CUT, where the function is only dependent on its input parameters. The benefit is that it is possible to use any form of ATCG (random testing, combinatorial approaches), independent of them being white- or black-box techniques. Replacement of subroutines with *semi-purification doubles* (an adapted form of test stubs [6]) allows for the specification of additional observation points. The observations can later be used for the definition of assertions in test cases. Listing 1 shows a short program that returns a random value ≥ 15. During the SP process the dependency is replaced with a SP double of same name. The resulting code is displayed in Listing 2. The additional parameter `b` was added to replace the return value in the SP double and simulate the CUT behaviour. Additionally an observation point was inserted[2].

ITEC uses Microsoft's Pex tool to generate test input. As Pex operates on the .NET Intermediate Language, the semi-purified CTRL code has to be translated to a .NET language. C# was chosen since it is syntactically similar to CTRL.

After the translation, the CUT is added to other artefacts and compiled into a `.dll` file. The required resources are:

- **PUT:** Parameterized unit tests (PUTs) are entry points for Pex' exploration. They also specify observation points and expectations towards parameter values.
- **Pex factories:** Factories are manually created, annotated methods that serve as blueprints for data types. They help Pex generate input values. ITEC uses factories to teach Pex how to generate certain CTRL data types.
- **Data types:** Many CTRL data types are not natively present in C# , e.g. `time`, `anytype` or CTRL's dynamic list-types. They were re-implemented in C#.
- **Built-in functions:** CTRL's standard library provides functions for various actions (e.g. `string` operations). They had to be re-implemented to ensure the source code's compatibility.
- **Other:** Some additional libraries were developed to support the generation. One example is a *Serializer* for generated values.

Following the compilation, Pex is triggered on the resulting executable.

CTRL test cases are created from the results of Pex's exploration. Each set of values generated by Pex represents a test case for the semi-purified CUT. These values can be classified into three different categories:

1) **Parameter values:** The parameter values for the original CUT are used as arguments for the test's function call to the CUT.
2) **SP parameter values:** Additional parameters introduced by the SP process are used to specify test doubles for the test case execution. The values for semi-purified global variables are assigned before the call to the CUT.
3) **Observations:** Observations are transformed into assertion points. Additionally to `return` and reference parameter values there is the possibility to assert database writes and similar commands.

Figure 1 visualises the split of this information and shows how the values are used in the test cases.

The last step is the test case execution. To run the tests it is necessary to wrap them inside a construct of functions that will permit the observation of success or failure. Note that in our case, success means that the observations during the CTRL execution match the observations made by Pex. Additionally, test doubles are generated from their specifications and the code is modified to call the stubs instead of the original dependencies.

B. Challenges & Lessons Learned

There are several challenges we faced during the creation of ITEC. One challenge is the translation of CTRL code to C#. Small changes in semantics have big impacts on the generated values. For example, list indices in C# are zero-based, while in CTRL they start at one. This means, that these lists had to be re-implemented, adding to constraint complexity, as Pex is optimised to native types. Additionally, C# is incapable of dealing with index or casting-expressions as reference parameters. These statements had to be extracted and placed before the function call, the resulting values written back after. There are numerous similar differences, leading to re-implementation of data types and functionality, while increasing the complexity of path constraints.

The validation of the translation is an important challenge. In [7] we give one proposal to solve this problem. The full list of challenges has been described in more detail in [8].

The lesson learned during implementation of the translator is that re-implementation of data types can be time-consuming and difficult. Often an increase in applicability and translation validity comes with a drop in performance. Finding a solution to these issues is one of the big challenges of our approach.

Despite the effort of implementing a translator, SP engine and TC generator, we believe that the implementation of a DSE tool for CTRL would be more costly.

IV. Current Results

We executed the tool on 1521 functions in the CTRL framework. For 52.0 % (791) of the functions ITEC was able to execute Pex. The other 48.0 % failed due to one of the following reasons:

[2]The observation is not required here, but added to show the functionality

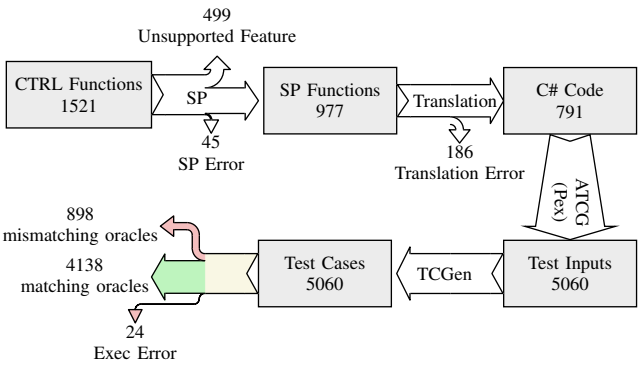


Fig. 2. Sankey diagram displaying the test generation and execution for the JCOP framework

- use of unsupported features, data types and functionality (32.8%): We explicitly excluded some functionality due to their complexity. Example: user interface interactions.
- errors due to unresolvable resource links (3.0%): The CERN-developed CTRL IDE occasionally has problems linking function invocations to the correct definitions.
- compilation errors of the translated C# (12.2%): The translator actively does not account for some language differences as they can be seen as "bad coding practice". ITEC serves as a motivation to avoid/alter these parts of the code base. Other concepts, such as the casting of native C# data types as explained above, cannot be translated.

For the 791 former functions, ITEC generated between 0 and 104 test cases (mean: 6.37; median: 4; first and third quartile: 2 and 7). Figure 2 displays the process and numbers.

The execution of these test cases lets us look at the line coverage data, as produced by WinCC OA's CTRL interpreter. The coverage shows a distribution as follows: 76% of the functions are fully covered, 9.9% have a coverage higher than 75%, for 7.2% of the functions the coverage is above 50%, the rest has either a coverage under 50% or no recorded data due to errors during the execution.

One result we observed during our analysis is that the coverage drops for long functions. While routines with less than 40 LOC are covered to a large extent (over 75 % line coverage), more than half of the functions longer than 40 LOC achieve less coverage. This suggests that it is harder to generate covering test suites for long functions, due to higher complexity in the path constraints. This theory is supported by the fact, that in general longer functions produce fewer test cases and that long routines with smaller test suites have bad coverage metrics.

We refer the reader to the technical document [8] for a more detailed breakdown of the test case generation and a thorough analysis of the test case execution results.

V. Related Work

Test case generation through symbolic execution has been researched by others before us. Cseppentő et al. compared different SE tools in a benchmark [9], supporting our choice of Pex. [10] shows an approach to isolate units from their dependencies, similar to semi-purification. Bucur et al. [11] perform SE on interpreted languages by adapting the interpreter instead of writing a new engine. Bruni et al. show their concept of lightweight SE for Python, by using Python's operator overloading capabilities in [12]. The testing of database applications via mock objects was presented in [13].

VI. Summary and Future Work

This paper shows our considerations for automated test case generation for CTRL, a proprietary, ANSI C-like language. We describe different approaches for language independent symbolic execution, before we explain our particular approach. Our solution involves the translation from CTRL to C#, which is supported by Microsoft's Pex dynamic symbolic execution tool. We explain the tool's general workflow and present lessons learned and results from the tool execution on our main codebase.

In future, we aim to extend the tool's applicability by lowering the number of unsupported features and executing it on additional parts of CERN's codebase. We also aim to enhance our analysis by comparing coverage to code complexity measures. Orthogonal to this effort our efforts will evaluate other (D)SE tools and approaches such as the ones described in the first part of this paper.

References

[1] C. Cadar and K. Sen, "Symbolic Execution for Software Testing: Three Decades Later," *Commun. ACM*, vol. 56, no. 2, pp. 82–90, Feb. 2013.

[2] N. Tillmann, J. de Halleux, and T. Xie, "Transferring an Automated Test Generation Tool to Practice: From Pex to Fakes and Code Digger," in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*. ACM, 2014, pp. 385–396.

[3] Microsoft Research, "Pex, Automated White box Testing for .NET," http://research.microsoft.com/en-us/projects/pex/.

[4] L. Bettini, *Implementing Domain-Specific Languages with Xtext and Xtend*. Packt Publishing, 2013.

[5] S. Klikovits, D. P. Y. Lawrence, M. Gonzalez-Berges, and D. Buchs, "Considering Execution Environment Resilience: A White-Box Approach," in *Software Engineering for Resilient Systems*, vol. 9274. Springer LNCS, 2015, pp. 46–61.

[6] G. Meszaros, "Test Doubles," in *XUnit Test Patterns: Refactoring Test Code*. Addison Wesley, 2011.

[7] S. Klikovits, D. P. Y. Lawrence, M. Gonzalez-Berges, and D. Buchs, "Automated Test Case Generation for the CTRL Programming Language Using Pex: Lessons Learned," vol. 9823. Springer LNCS, 2016, pp. 117–132.

[8] S. Klikovits, P. Burkimsher, M. Gonzalez-Berges, and D. Buchs, "Automated Test Case Generation for CTRL," Report EDMS 1743711, 2016. [Online]. Available: https://edms.cern.ch/document/1743711

[9] L. Cseppentő and Z. Micskei, "Evaluating Symbolic Execution-based Test Tools," in *Proceedings of the IEEE Int. Conf. on Software Testing, Verification and Validation (ICST)*. IEEE, 2015.

[10] D. Honfi and Z. Micskei, "Generating unit isolation environment using symbolic execution," in *Proceedings of the 23rd PhD Mini-Symposium*. IEEE, 2016.

[11] S. Bucur, J. Kinder, and G. Candea, "Prototyping symbolic execution engines for interpreted languages," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2014, pp. 239–254.

[12] A. D. Bruni, T. Disney, and C. Flanagan, "A Peer Architecture for Lightweight Symbolic Execution," Tech. Rep., 2011. [Online]. Available: https://hoheinzollern.files.wordpress.com/2008/04/seer1.pdf

[13] K. Taneja, Y. Zhang, and T. Xie, "MODA: Automated test generation for database applications via mock objects," in *Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering*, 2010, pp. 289–292.

Constraint Programming with Multi-valued Decision Diagrams: A Saturation Approach

Vince Molnár, István Majzik
Budapest University of Technologies and Economics, Department of Measurement and Information Systems
Email: {molnarv, majzik}@mit.bme.hu

***Abstract*—Constraint programming is a declarative way of modeling and solving optimization and satisfiability problems over finite domains. Traditional solvers use search-based strategies enhanced with various optimizations to reduce the search space. One of such techniques involves multi-valued decision diagrams (MDD) to maintain a superset of potential solutions, gradually discarding combinations of values that fail to satisfy some constraint. Instead of the relaxed MDDs representing a superset, we propose to use exact MDDs to compute the set of solutions directly without search, compactly encoding all the solutions instead of enumerating them. Our solution relies on the main idea of the saturation algorithm used in model checking to reduce the required computational cost. Preliminary results show that this strategy can keep the size of intermediate MDDs small during the computation.**

I. Introduction

Many problems in computer science such as operations research, test generation or error propagation analysis can be reduced to finding at least one (optimal) assignment for a set of variables that satisfies a set of constraints, a problem called *constraint programming* (CP) [9]. CP solvers usually use a search-based strategy to find an appropriate solution, enhanced with various heuristics to reduce the search space.

Multi-valued decision diagrams (MDD) are graph-based representations of functions over tuples of variables with a finite domain [7]. As such, they can be used to compactly represent sets of tuples by encoding their membership function. Set operations computed on MDDs then have a polynomial time complexity in the size of the diagram instead of the encoded elements [4].

One of the heuristics proposed for CP solvers use MDDs to maintain a superset of potential solutions, gradually shrinking the set by discarding tuples failing to satisfy some constraint [1]. These approaches limit the size of MDDs to sacrifice precision for computational cost, which is compensated for by the search strategy. Using an exact representation of solution sets seems to be neglected by the community, except in [6] where special decision diagrams are used to achieve this.

This paper proposes to revisit the idea of exact MDD-based CP solvers, applying a strategy well-known in the model checking community: the saturation algorithm [5]. An efficient implementation of the idea could overcome a common limitation of search-based approaches, i. e., the complexity of computing *every* solution. As opposed to traditional search-based solvers, such a tool could natively compute the MDD representation of all the solutions instead of enumerating them one by one.

This work has been partially supported by the CECRIS project, FP7–Marie Curie (IAPP) number 324334. Special thanks to Prof. András Pataricza and Imre Kocsis for their motivation and support.

II. Preliminaries

A. Multi-valued Decision Diagram

Multi-valued decision diagrams (MDD) offer a compact representation for functions in the form of $\mathbb{N}^K \to \{0,1\}$ [7]. MDDs can be regarded as the extension of binary decision diagrams first introduced in [4]. By interpreting MDDs as membership functions, they can be used to efficiently store and manipulate sets of tuples. Definition of MDDs (based on [8]) and common variants are as follows.

Definition 1 (Multi-valued Decision Diagram) A *multi-valued decision diagram* (MDD) encoding the function $f(x_1, x_2, \ldots, x_K)$ (where the domain of each x_i is $D_i \subset \mathcal{N}$) is a tuple $MDD = \langle N, r, \mathit{level}, \mathit{children}, \mathit{value}\rangle$, where:

- $N = \bigcup_{i=0}^{K} N_i$ is a finite set of *nodes*, where items of N_0 are *terminal* nodes, the rest ($N_{>0} = N \setminus N_0$) are *nonterminal* nodes;
- $\mathit{level} : N \to \{0, 1, \ldots, K\}$ is a function assigning non-negative *level numbers* to each node ($N_i = \{n \in N \mid \mathit{level}(n) = i\}$);
- $r \in N$ is the *root node* of the MDD ($\mathit{level}(r) = K$);
- $\mathit{children}$: $N_i \times D_i \to N$ is a function defining edges between nodes labeled by elements of D_i, denoted by $n_k[i]$ (i. e., $\mathit{children}(n_k, i) = n_k[i]$);
- $(N, \mathit{children})$ as a directed graph is acyclic (DAG);
- $\mathit{value} : N_T \to \{0,1\}$ is a function assigning a *binary value* to each terminal node (therefore $N_0 = \{\mathbf{0}, \mathbf{1}\}$, where $\mathbf{0}$ is the terminal zero node ($value(\mathbf{0}) = 0$) and $\mathbf{1}$ is the terminal one node ($value(\mathbf{1}) = 1$).

An MDD is *ordered* iff for each node $n \in N$ and value $i \in D_{level(n)}$: $level(n) > level(n[i])$. An ordered MDD is *quasi-reduced* (QROMDD) iff the following holds: if $n \in N$ and $m \in N$ are on the same level and all their outgoing edges are the same, then $n = m$. An ordered MDD is *fully reduced* (ROMDD) if it is quasi-reduced and there is no node $n \in N$ such that every children of n are the same node.

The width of an MDD is the maximum number of nodes belonging to the same level: $w(MDD) = max_{1 \le i \le K}(|N_i|)$.

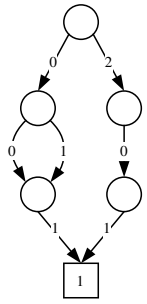


Fig. 1: An ordered MDD.

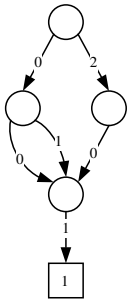


Fig. 2: A quasi-reduced MDD.

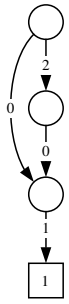


Fig. 3: A fully-reduced MDD.

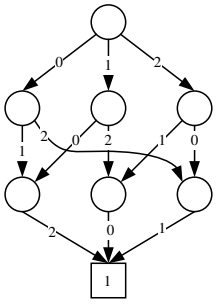


Fig. 4: MDD representation of ALLDIFFERENT(x_1, x_2, x_3).

The height of an MDD is the highest level to which any node belongs: $h(MDD) = max_{n \in N}(level)$. Note that only ROMDDs can have a lower height than K.

The semantics of a *quasi-reduced* MDD rooted in node r in terms of the encoded function f is the following: $f(v_1, v_2, \dots v_K) = \mathit{value}((((r[v_K])[v_{K-1}]) \cdots)[v_1])$. When interpreted as a set, the set of all tuples encoded in an MDD rooted in node r is $S(r) = \{\mathbf{v} \mid \mathit{value}((((r[v_K])[v_{K-1}]) \cdots)[v_1]) = \mathbf{1}\}$.

In case of ROMDDs, a reduced node is assumed to have all edges connected to the target of its incoming edge.

Figures 1–3 illustrate an ordered but not reduced, a quasi-reduced and an ROMDD respectively, both encoding the set of tuples $\{(0,0,0),(0,1,0),(0,0,1)\}$ over the domain $\{0,1,2\} \times \{0,1\} \times \{0,1\}$. For the sake of simplicity, the terminal **0** node is omitted in figures.

An advantage of decision diagrams is the ability to compute set operations such as union and intersection with polynomial time complexity in the number of nodes in the operands [4].

B. Constraint Programming

Constraint programming (CP) is a framework for modeling and solving continuous or discrete optimization or satisfiability problems over finite domains [9]. The main advantage of CP over the similar SAT or ILP problems is that it can handle arbitrary finite domains and virtually any type of constraints: they can be any relation over the set of defined variables. The subset of CP problems addressed in this paper is *constraint satisfaction problems* (CSP), where the goal is to find at least one tuple that satisfies all the defined constraints.

Definition 2 (Constraint satisfaction problem) A *constraint satisfaction problem* (CSP) is defined over a set of *variables* $X = \{x_1, \dots, x_K\}$ with finite domains $D = \{D(x_1), \dots, D(x_K)\}$ and set of arbitrary relations over the variables (called *constraints*) $C = \{c_i \mid c_i \in D(x_{i_1}) \times \cdots \times D(x_{i_k})\}$, where $S(c_i) = \{x_{i_1}, \dots, x_{i_k}\} \subseteq X$ is the support of the constraint, i. e., the variables over which the relation is defined. The question is whether there exists a tuple $\mathbf{v} \in D(x_1) \times \cdots \times D(x_K)$ that satisfies all constraints in C.

Due to the rich modeling opportunities, solvers cannot exploit any uniform semantics of the constraints. CP solvers therefore employ a systematic search, supported by *propagation strategies* that transfer the knowledge inferred in a constraint to other constraints to reduce the search space [9]. *Constraint propagation* is the process of discarding as many potential solutions as possible before stepping in the search. One extremity is the explicit enumeration of every possible tuple, gradually shrinking the set by discarding those that violate some constraint. In this case, a search in the traditional sense is not necessary, as after the restrictions, every tuple is a valid solution to the problem. Since this approach is generally considered infeasible or not scalable, the CP literature proposed various relaxations that are still useful to reduce the search space with an acceptable cost. Note however, that most of these solutions are therefore limited when it comes to computing *every* solution of a problem.

1) Domain-based Constraint Propagation: The traditional constraint propagation approach for CSP solving is built around *domain stores* [9]. Domain stores maintain the current domain for every variable of the problem, propagating inferred knowledge by the means of *domain consistency* [9].

Definition 3 (Domain consistency) A constraint C is *domain consistent* with the current variable domains D if for every value $v \in D(x_i)$ of every variable $x_i \in X$, there exists a tuple $\mathbf{v}$ with $\mathbf{v}[i] = v$ that satisfies C.

A constraint C can be made domain consistent by discarding values from the domains that cannot be extended to a tuple that satisfies C. Constraint propagation then consists of making constraints domain consistent until every constraint is domain consistent. If any domain becomes empty, the problem is unsatisfiable. If every domain contains a single value only, a solution is found and returned. In any other case, the search strategy binds the value of a variable and repeats the process.

2) MDD-based Constraint Propagation: One weakness of domain-based constraint propagation approaches is the lack of interaction between variables, i. e., every subset of the Cartesian product of the domains is domain consistent. For example, in the case of the ALLDIFFERENT constraint, which demands that all the variables in the tuple should assume different values, domain consistency fails to express the connection

between values of the variables.

To address this, [1] introduced the notion of *MDD consistency* and enhanced the domain store with an *MDD store*. MDDs can efficiently encode the various interactions between variables (see Figure 4 for the MDD representation of the ALLDIFFERENT constraint for three variables). One or more MDDs can then be used to communicate the restrictions between different constraints.

Definition 4 (MDD consistency) A constraint C is *MDD consistent* with an MDD rooted in node r if every edge in $children$ belongs to at least one path leading from r to the terminal $\mathbf{1}$ representing a tuple $\mathbf{v} \in S(r)$ that satisfies C.

MDD-based constraint propagation approaches usually use limited-width MDDs to reduce the complexity of MDD operations at the cost of losing some information. As previously noted, the spurious solutions introduced by the relaxation are eliminated with a search strategy that will eventually concretize solutions to obtain an exact result.

Note that in this form, MDDs are used as a relaxed set of potential solutions, a superset of the actual solutions. Domain stores can be regarded as the special case when the width of the MDD is fixed in one [2]. In this case, every domain is represented by the edges starting from the node on the corresponding level and the MDD encodes the Cartesian product of the domains.

III. SATURATION-BASED CONSTRAINT PROPAGATION

As presented in Section II, the CP community have embraced limited-width MDDs as a means to enhance the traditional domain store for more efficient constraint propagation. However, the literature rarely mentions the possibility of using explicit MDD representations (with unlimited width) and MDD operations to compute the set of solutions directly without relaxations and searching. As it seems, researchers of the community consider this approach infeasible or not scalable, which can explain the lack of corresponding results.

This paper proposes to revisit the idea mentioned as an extremity in Section II-B, that is, enumerating every potential solution and discarding those that fail to satisfy some of the constraints. In this setting, *fully reduced* MDDs provide an efficient encoding as only the levels corresponding to variables in the support of a constraint will contain nodes. The set of all tuples, for example, is encoded simply by the terminal $\mathbf{1}$ node, as none of the variables are bound by any constraint.

Discarding invalid solutions is then equivalent to computing the intersection of the current set of potential solutions with the set of solutions of the next constraint c_i. The set of all actual solutions is therefore obtained by taking the intersection of all the MDDs representing every constraint in the problem:

$$\mathcal{S} = \bigcap_{c_i \in C} \mathcal{S}(c_i) \tag{1}$$

The usual pitfall in MDD-based set operations is that the decision diagrams tend to grow very large during computation. Such computations usually aim to reach a fixed point, thus the number of encoded tuples constantly rises or falls during the computation. *Denser* or *sparser* sets usually have a more compact MDD representation than those encoding around half of all the possible tuples, thus the final size of the decision diagram is usually in an acceptable range. Intermediate results, however, can be exponentially larger (multiple orders of magnitude in practice as shown in Section IV).

The symbolic model checking community uses an efficient strategy to combat this phenomenon: *saturation* [5].

A. The Saturation Approach

Originally, saturation has been proposed as an iteration strategy tailored to work with decision diagrams to perform least fixed point computation with the transition relation of state-based behavioral models for state space exploration [5]. In other words, its original purpose is to efficiently compute the reflexive transitive closure of a relation on a set of initial values (initial states), minimizing the size of intermediate decision diagrams during the computation.

The essential idea of saturation is to keep the intermediate decision diagrams as dense as possible by applying relations affecting only the lower levels first. Relations are therefore applied in the order of the highest level that they affect.

In the CSP setting, transition relations are replaced with constraints and instead of the reflexive transitive closure, the intersection of all constraints must be computed. Nevertheless, the idea of ordering the constraints by the "highest" variable they affect (highest in terms of the decision diagram level that encodes the variable) is applicable. Moreover, it should carry the same benefits since the number of currently encoded tuples is again monotonic during the computation – this time converging towards the empty set.

To formally describe the proposed solution, we have to assume that a variable ordering is given, i. e., there is a bijective function $l : X \to \{1, \ldots, K\}$. This function describes the relationship between the variables and the encoding MDD as well: every variable x_i is represented by the level $l(x_i)$. W.l.o.g., we will assume that $l(x_i) = i$, unless otherwise noted.

Definition 5 (Level of constraint) The *level* of a constraint c_i is $Top(c_i) = max\{l(x_i) \mid x_i \in S(c_i)\}$, i. e., the largest level number assigned to the variables in the support of the constraint. Let $C_i = \{c_j \mid Top(c_j) = i\}$ then denote the set of constraint belonging to level i.

Ordering the constraints by the assigned *Top* level modifies Equation 1 as follows:

$$\mathcal{S} = \left(\left(\left(\underbrace{c_1^1 \cap \ldots}_{c_j^1 \in C_1} \right) \cap \underbrace{c_1^2 \cap \ldots}_{c_j^2 \in C_2} \right) \cap \ldots \right) \underbrace{c_1^K \cap \ldots}_{c_j^K \in C_K} \tag{2}$$

B. Discussion

The strategy of the proposed saturation approach can be characterized by two goals:

- In every step, minimize the height of the resulting (fully-reduced) MDD.

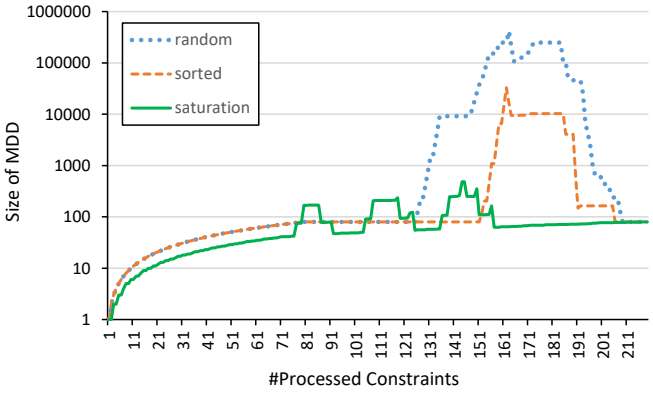


Fig. 5: MDD size during computation.

- If a new node n is introduced on a new level, minimize the number of tuples n encodes.

Both of these goals are accomplished with the ordering of constraints. The following lemmas provide the rationale.

Lemma 1 The width of an ROMDD cannot be higher than the number of tuples it encodes.

Proof *Assume there is an ROMDD with width w encoding $w-1$ tuples. Having a width w means there is at least one level with w nodes. Every node in any ROMDD has to be on a path leading from the root node to the terminal* **1***, so the MDD must encode at least w tuples, as opposed to $w-1$.*

Lemma 2 The number of nodes in an ROMDD cannot be higher than $w(MDD) \cdot h(MDD)$.

From the lemmas we can conclude that the saturation strategy aims to minimize the size of the resulting MDD in every step. Note, however, that the number of encoded tuples is not in a direct relationship with the width of the MDD. As stated before, the two extremities are the empty set and the universe, but in between the size can grow exponentially. The saturation strategy can be therefore considered as a "best effort" heuristic rather than an optimal algorithm.

IV. Results

The proposed approach has been implemented in Java as a CSP solver processing problems given in the XCSP3 format [3]. As a proof of concept, a small experiment has been carried out where a simple model encoding error propagation in a railway system has been solved by three different strategies.

The first "strategy" applied the constraints in the order of declaration in the problem definition (which can be considered more-or-less random). The second one orders the constraints by the number of variables supporting the constraint, while the third one is the proposed saturation approach. Figure 5 shows the size of the solution MDD after the processing of each constraint for the three strategies on a logarithmic scale.

The experiment demonstrates the potential benefits of using the saturation approach in an explicit MDD-based CSP solver. Compared to the "random" strategy, the peak size of the MDD was almost three orders of magnitude smaller with the saturation approach, at most 6 times more than the final size. Ordering the constraints by the number of supporting variables yields a better result than the random strategy, but it is still far worse than the saturation approach. The remaining peaks correspond to the inclusion of complex constraints when a new variable is processed and the size of the MDD usually normalizes before processing the next variable.

V. Conclusion and Future Work

This paper proposed to revisit a seemingly undiscussed topic of applying exact MDD-based methods to compute the solution set of finite-domain constraint programming problems. Although the approach of compiling the MDD representation of constraints and computing their intersection may seem "brute-force", it is worth exploring the solutions employed in related research areas such as symbolic model checking.

In this spirit, we have applied the strategy of the saturation algorithm well-known in the model checking community. Saturation orders the relations by the highest level assigned to one of their supporting variables, keeping the intermediate MDD relatively compact compared to other approaches.

We have demonstrated the benefits of the strategy in a small experiment, which provided promising results. Once a larger set of benchmark models are available after the first XCSP3 competition[1], we plan to systematically evaluate and fine-tune our solution to see if it can match the performance of traditional tools employing the domain- and MDD-store approaches. Regardless of the performance, it is noteworthy that the proposed solution can natively compute the set of *all solutions* in a compact representation, which poses a great challenge to traditional tools that can only enumerate the solutions one by one.

References

[1] H. R. Andersen, T. Hadzic, J. N. Hooker, and P. Tiedemann. A constraint store based on multivalued decision diagrams. In *Proc. of the 13th Internation Conference on Principles and Practice of Constraint Programming*, pages 118–132. Springer, 2007.

[2] D. Bergman, A. A. Cire, W. J. van Hoeve, and J. Hooker. *MDD-Based Constraint Programming*, pages 157–181. Springer, 2016.

[3] F. Boussemart, C. Lecoutre, and C. Piette. XCSP3: an integrated format for benchmarking combinatorial constrained problems. *CoRR*, abs/1611.03398, 2016.

[4] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

[5] G. Ciardo, R. Marmorstein, and R. Siminiceanu. The saturation algorithm for symbolic state-space exploration. *International Journal on Software Tools for Technology Transfer*, 8(1):4–25, 2006.

[6] R. Mateescu and R. Dechter. Compiling constraint networks into AND/OR multi-valued decision diagrams (AOMDDs). In *Proc. of the 12th Int. Conf. on Principles and Practice of Constraint Programming*, pages 329–343, 2006.

[7] D. M. Miller and R. Drechsler. Implementing a multiple-valued decision diagram package. In *Proc. of the 28th IEEE Int. Symp. on Multiple-Valued Logic*, pages 52–57, 1998.

[8] V. Molnár, A. Vörös, D. Darvas, T. Bartha, and I. Majzik. Component-wise incremental LTL model checking. *Formal Aspects of Computing*, 28(3):345–379, 2016.

[9] F. Rossi, P. van Beek, and T. Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier, 2006.

[1] http://xcsp.org/competition

Effects of Graph Transformation Rules to Design Space Exploration Problems

András Szabolcs Nagy, Dániel Varró
Budapest University of Technology and Economics, Department of Measurement and Information Systems, Hungary
MTA-BME Lendület Research Group on Cyber-Physical Systems, Hungary
Email: {nagya, varro}@mit.bme.hu

***Abstract*—Design space exploration (DSE) aims to explore different design candidates that satisfies multiple criteria and is optimal with respect to different quality properties. The strength of rule-based DSE is that the exploration rules can be tailored to a specific problem, shaping the design space into a more concise form than traditional approaches. However, experts may have several choices to define the exploration rules and choosing the good one may increase exploration performance significantly. In this paper, we present several ways to define the exploration rules of a rule-based DSE problem and investigate the impact of these rules.**

I. Introduction

As a challenging branch of search based software engineering (SBSE), design space exploration (DSE) aims at searching through different design candidates to fulfill a set of constraints and then proposing optimal designs with respect to certain (multiple) objectives. It frequently supports activities like configuration design of avionics and automotive systems or dynamic reconfiguration of systems with high availability at runtime. Many of such traditional DSE problems can be solved by using advanced search and optimization algorithms or constraint satisfaction programming techniques [1], [2].

In model-driven engineering (MDE), rule-based DSE [3], [4], [5] aims to find instance models of a domain that are 1) reachable from an initial model by applying a sequence of exploration rules, while 2) constraints simultaneously include complex structural and numerical restrictions. A solution of such a problem is a sequence of rule applications which transforms the initial model to a desired model. Multi-objective rule-based DSE (MODSE) may also include multiple optimization objectives [4] which help to distinguish between solutions.

One of the major characteristics of rule-based DSE against traditional techniques is that the domain expert can define the atomic steps the exploration process can use to modify a candidate solution. These exploration steps have a high impact on the actual design space that an algorithm has to explore and thus it will affect the overall performance of the exploration. However, the exploration steps can be defined in multiple ways even for a relatively small problem and affecting the design space and thus the performance differently.

This paper is partially supported by the MTA-BME Lendület 2015 Research Group on Cyber-Physical Systems.

The objective of this paper is to give an insight to newcomer DSE users through an example how the exploration rules can impact the performance of the exploration and to help decide on defining the exploration rules.

The paper is structured as follows: Sec. II briefly presents a motivating example and introduces the most important concepts of rule-based DSE, Sec. III provides an insight of the effects of transformation rules and Sec. V concludes the paper.

II. Rule-Based Design Space Exploration

Case study: The motivating example of this paper is the class responsibility assignment problem from the 9th Transformation Tool Contest (TTC16) [6].

The exploration task is taken from a reengineering challenge of object-oriented programs: create classes for a set of initially given methods and attributes (features) where methods can depend on other methods and attributes, in such a way that the resulting class model is optimal with respect to a software metric called CRA-Index (a metric derived from cohesion and coupling).

Besides the CRA-Index, there are two important constraints that the resulting class model has to satisfy: 1) there should be no empty classes in the resulting model and 2) all the features have to be assigned.

Domain model and instance model: Model-driven system design (MDSD) aims to lift the abstraction level of a problem allowing a better overview of it. For this purpose a **domain model** is created which describes the possible elements of the problem, their properties and their relations. For example the domain model of the CRA problem defines classes, features (methods and attributes) and relations between them. Domain models are also called **metamodels** as a metamodel describes the possible components of a semantic model, also called an **instance model**. In a metamodel *types* (or *classes*) describe objects from the domain which can have *attributes*, while *references* specify the relations between types.

Graph patterns and matches: A common task is to obtain data from *instance models* using queries. For this, **graph patterns** provide a good formalism which can be seen as a small, deficient instance model that we search for as part of the actual instance model. A graph pattern can capture elements, relations, negative or positive conditions on attributes and

multiplicity. A graph pattern can have multiple **match**es on an instance model similarly as a database query can return multiple rows.

Graph transformation rules: Modifications to an instance model are often described as **graph transformation rules**. A rule consists of a precondition or a left hand side (LHS), which is captured by a graph pattern and a right hand side (RHS), which declaratively defines the effects of the operation. A rule is applied on a model by 1) finding a match of graph pattern LHS (also called an **activation** of the rule), then 2) removing elements from the model which have an image in $LHS \setminus RHS$, then 3) changing the value of attributes which are reassigned in RHS and finally 4) creating new elements $LHS \setminus RHS$. A rule can have multiple activations or non at all as expected.

Rule-based design space exploration problem: The aim of rule-based DSE is to evolve a system model along transformation rules and constraints to find an appropriate system design. The state space of design candidates is potentially infinite but usually it also has a dense solution space.

The input of a **rule-based DSE problem** consists of three elements $RDSE = (M_0, G, R)$: 1) an initial model M_0, 2) a set G of goals given by graph patterns, which should be satisfied by the solution model M_{Si} and 3) a set R of transformation rules $(r_1, r_2, \dots r_r)$ which define how the initial model M_0 can be manipulated to reach a solution model M_{Si}. As a result it produces several solutions $(M_{S0}, M_{S1} \dots M_{Sn})$ satisfying all of the goals and each of them is described by a sequence of rule applications (or **trajectories**) $(r_{\vec{S}0}, r_{\vec{S}1} \dots r_{\vec{S}n})$ on the initial model M_0.

Furthermore, there are two optional inputs: **global constraints** GC and **objectives** O. Global constraints have to be satisfied on the model along each valid execution path (*i.e.,* trajectory) and are usually defined by graph patterns. An objective defines a function over either the model or the trajectory to derive a fitness value (*e.g.,* cost, response time, reliability) which can be used to distinguish between solutions in quality.

Design space: To solve an RDSE problem a search of the design space has to be conducted in accordance with an exploration strategy. The design space is a directed graph where the nodes represent the different states of the model and edges represent the rule applications (activations). There is one initial node that represents the state of the initial model and usually there are multiple goal states that satisfy the goal constraints. Depending on the RDSE problem, a model state can be reached in multiple trajectories (*e.g.,* there are independent rule activations) and the design space can have cycles in it (*e.g.,* one of the rule applications creates an element and the other deletes it).

III. Effects of Exploration Rules

In this section, we show six approaches to define the graph transformation rules for the CRA problem and discuss the properties of these approaches. We find this research relevant as it is a recurring problem in rule-based DSE to create new elements (classes) and to connect them (assign features to classes) and it is not trivial if either approach is better or worse than the other. With this research, we help newcomers to identify different approaches and to decide between them with the help of this evaluation.

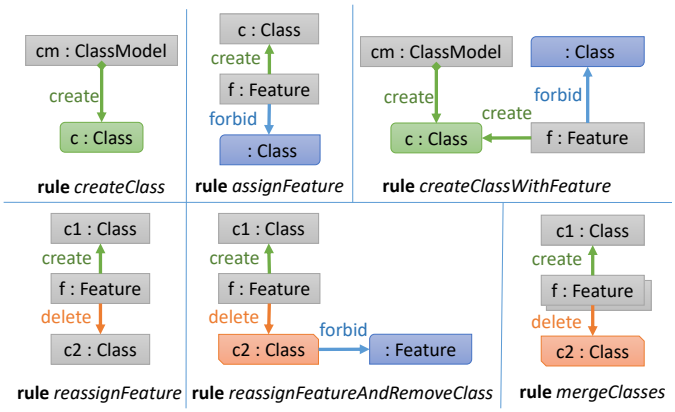


Fig. 1: Graph transformation rules used to solve the CRA problem

In Fig. 1, we present the different graph transformation rules. First, we describe them and then we will refer to them in the next paragraphs.

Rule 1 *createClass*: creates a class and inserts it to the model.

Rule 2 *assignFeature*: assigns an unassigned feature to an existing class.

Rule 3 *createClassWithFeature*: creates a class for a feature that is not assigned yet.

Rule 4 *reassignFeature*: reassigns a feature from a class to an other existing class.

Rule 5 *reassignFeatureAndRemoveClass*: reassign a feature and remove the class if it has no features.

Rule 6 *mergeClasses*: reassign all the features to a target class and remove the source class.

a) Atomic Modifications with Bounds: The idea of the first approach is to create classes and assign features separately (*createClass* rule and *assignFeature* rule). Using such atomic modifications, one can easily see that all the possible solutions are reachable (it is complete) and probably this is the first to think of when defining a DSE problem as these are the simplest rules. However, having a transformation rule that can create objects (*e.g.,* classes) without upper bound can make the exploration strategy to create too many of them and in different order.

To overcome this issue, we incorporate an upper bound for creating classes to the condition of the *createClass* rule. In our example, this bound is the number of features available in the model as creating more classes is unnecessary.

Unused objects (*e.g.,* empty classes) is an other problem of this approach and can be handled in two ways (without modifying the exploration rules). Besides the essential goal constraints (all features are assigned) additional goal constraints (no empty classes) can be added that forbids unused objects in solutions. However, the exploration may fail to

remove these unused objects and thus fail to return with a valid solution. Alternatively, these objects can be easily removed after the exploration has finished but in this case the found solutions may vary only in the number of unused objects.

b) Atomic Modifications with Maximum One Bound: This approach is the very same as the previous one, except that we incorporate a stronger bound to the *createClass* rule: maximum one unused class can be present in the model and a second one cannot be created.

c) The Generative Approach: A similar approach to the previous ones is to design the rules in such a way to never have an unused object (*e.g.,* classes). Using the *createClassWithFeature* rule instead of *createClass* rule, a newly created class will instantly have an assigned feature resulting in two advantages: 1) there is an upper bound for creating classes (number of features) and 2) the "no empty class" constraint is fulfilled automatically.

An interesting property of this approach is that the *createClassWithFeature* rule has more activations (number of unassigned features) as opposed to the first approach (maximum one activation) and it can affect the performance and the results of the exploration depending on the algorithm.

d) The Preprocess Approach: Another approach is to initially create the maximum number of required objects (*e.g.,* one class for each feature) in a preprocess phase and then use rules that connects the elements (*assignFeature* rule).

While in overall, there will be less transitions in the search space than the first approach as the classes are already created, there will be many activations available in the same state on average as any feature can assigned to any class. Also a post process is required to remove empty classes.

e) The Initial Solution Approach: The fifth approach is to create a valid initial model and use transformations that keeps the model valid throughout the exploration. In our example, this means to initially create and assign a class for each feature and use the *reassignFeatureAndRemoveClass* rule (*InitialSolution*).

f) Initial Solution with mergeClasses rule: Alternatively to the *reassignFeatureAndRemoveClass* rule, the *mergeClasses* rule can be used instead, which allows to merge two classes that have more than one features assigned (*InitialSolutionMergeClasses*).

IV. Evaluation

We carried out measurements for each approaches by 1) traversing the full search space with a depth-first search algorithm for initial models containing 3-7 features to understand the characteristics of the search space and by 2) searching for optimal solutions with the NSGA-II [4] algorithm for an initial model containing 18 features (initial model B introduced in the TTC case). The NSGA-II algorithm was configured with population size of 20 and with a mutation rate of 0.8.

The measurements were executed on a Intel® Core™ i5-2450M CPU @2.5 GHz desktop computer with 8 GB memory. The approaches were implemented in the open-source VIATRA-DSE framework [7].

Fig. 2 shows the first set of measurements with the number of states, transitions and the transitions-states ratio of the search space by model size (number of features). The measurements show that using atomic modifications without significant bound of creating objects results in a huge state space compared to other approaches. In the other hand, preprocessing the initial model and using carefully crafted rules (*e.g., mergeClasses* rule) may shrink the state space significantly.

The results of running an NSGA-II algorithm with different exploration rules shows interesting results. Fig. 3 depicts the median and the maximum found CRA-Index and the median time taken by different approaches where each point is an aggregation of ten separate runs. Points are missing where there were at least one run that couldn't return a valid solution, *i.e.,* there were unassigned features remaining. The horizontal axis shows the number of allowed fitness evaluations during exploration (250, 500, 750, 1000, 2000, ..., 7000).

While the *InitialSolutionMergeClasses* approach has the smallest state space, after 2000 evaluations it could not increase the CRA-Index significantly and was slower than most of the approaches. On the contrary, the *AtomicModificationsMaxOne* could improve consistently by passage of time and was the fastest among the others.

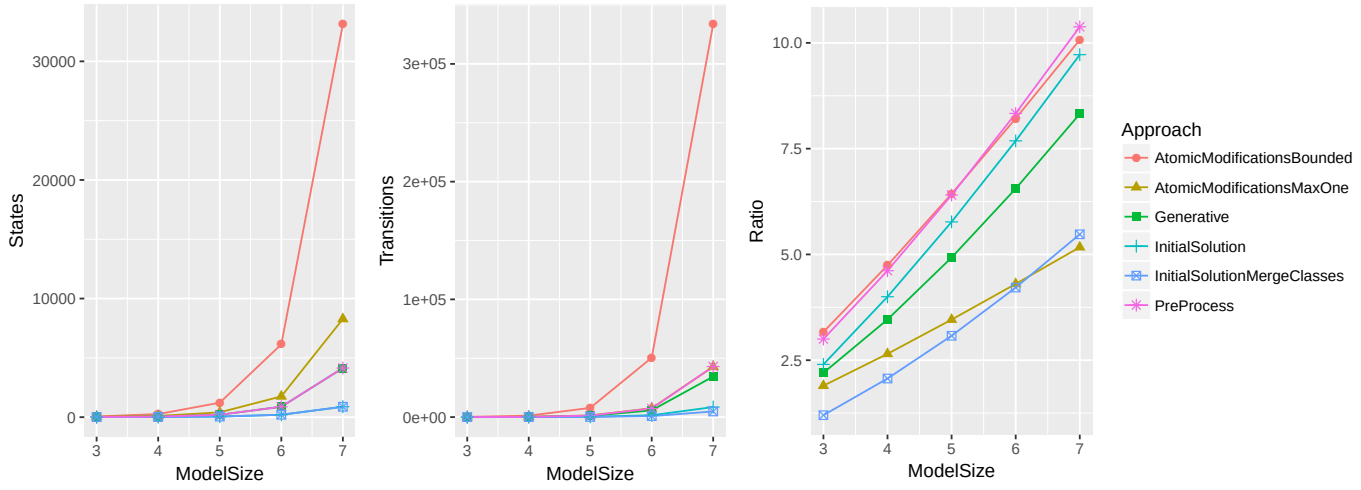
It is clear that *AtomicModificationsBounded* was the least effective approach of all and *Preprocess* was the best on average, however the best solutions are found by the slowest approach: *InitialSolution*. The *Generative* approach has the best trade-off: it found good solutions in reasonable time.

The *InitialSolution* approach was probably slow because the nature of the activations: there are relatively lot of them in each state and a good portion of them changes (disappears or appears). The used VIATRA-DSE framework evaluates a solution by transforming the initial solution and then backtracking to reuse the model. While the framework stores the activations (and activation codes) incrementally, it still has a lot of work maintaining them.

V. Conclusion

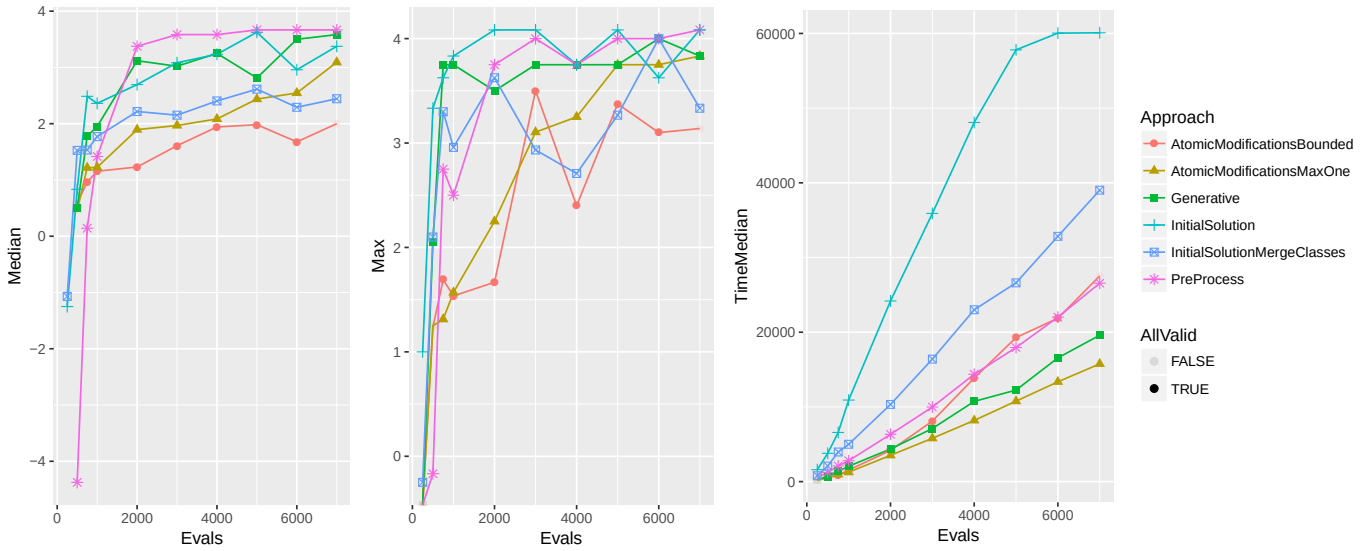
In this paper, we presented several ways to define the exploration rules of a rule-based DSE problem and investigated the impact of these rules. Based on the class responsibility assignment case, our observation is that using atomic modifications (creating single elements and connecting them separately) as exploration rules without any adjustments can be the worst choice. Instead, preprocessing the initial model and defining exploration rules that maintains the correctness of the model (and with that reducing the state space) may have significant positive impact on the exploration process. When using a genetic algorithm the best approach was either to create objects one at a time, maybe connecting them immediately to other elements, or creating the required amount of objects ahead of the exploration.

However, we investigated this observation only with a simple example and other problems may behave differently when modifying the exploration rules. Still, experimenting



(a) Explored number of states (b) Explored number of transitions (c) Ratio of explored number of transitions and states

Fig. 2: Comparison of approaches by model size using exhaustive search



(a) Median of reached CRA-Index (b) Maximum reached CRA-Index (c) Time taken to finish the exploration

Fig. 3: Comparison of approaches using NSGA-II by allowed number of evaluations. Each point is an aggregation of ten runs.

with different approaches is highly recommended for rule-based DSE problems to find the best system designs.

As for future work, more rule-based DSE problems could be investigated to derive a set of best practices for defining exploration rules when faced with a certain type of problem. Furthermore, based on the goal constraints and preconditions on the initial model, different set of exploration rules (approaches) could be generated automatically along with corresponding measurement skeletons.

References

[1] T. Basten, E. van Benthum *et al.*, "Model-driven design-space exploration for embedded systems: The Octopus toolset," in *Leveraging Applications of Formal Methods, Verification, and Validation*, ser. LNCS, 2010, vol. 6415, pp. 90–105.

[2] J. Denil, M. Jukšs, C. Verbrugge, and H. Vangheluwe, "Search-based model optimization using model transformations," McGill University, Canada, Tech. Rep., 2014.

[3] Á. Hegedüs, Á. Horváth, and D. Varró, "A model-driven framework for guided design space exploration," *Automated Software Engineering*, pp. 1–38, 08/2014 2014.

[4] H. Abdeen, D. Varró, H. Sahraoui, A. S. Nagy, Á. Hegedüs, Á. Horváth, and C. Debreceni, "Multi-objective optimization in rule-based design space exploration," in *29th IEEE/ACM International Conference on Automated Software Engineering (ASE 2014)*.

[5] M. Fleck, J. Troya, and M. Wimmer, "Search-based model transformations with momot," in *Proceedings of ICMT 2016*. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-42064-6_6

[6] ——, "The class responsibility assignment case," in *9th Transformation Tool Contest (TTC 2016)*, 2016.

[7] "The VIATRA-DSE framework," https://wiki.eclipse.org/VIATRA/DSE, Accessed: 2017-01-23.

Enhanced Spectral Estimation Using FFT in Case of Data Loss

András Palkó, László Sujbert
Budapest University of Technology and Economics,
Department of Measurement and Information Systems,
Budapest, Hungary
Email: palkoandras96@gmail.com, sujbert@mit.bme.hu

***Abstract*—Spectral estimation plays a significant role in engineering practice. Along with the spreading of sensor networks, more and more data are transmitted through unreliable channels which may lead to lost data. The most common method of spectral estimation uses FFT, but this requires the whole record without any data loss. This paper presents a new FFT-based method for the problem which can be used for coherent sampling. Its efficiency and accuracy is demonstrated via theoretical analysis, simulation and measurement results.**

Keywords—spectral estimation, data loss, FFT

I. Introduction

Data loss is usually caused by communication problems. For example, in sensor networks data are often transmitted via radio channel, which is known to be faulty if interference or noise occurs. Data loss can mean missing samples, invalid samples (e.g. ADC overdrive) or synchronization issues.

In engineering practice, spectral estimation plays an important role. If this is the measurement task and some of the samples are lost, then data loss becomes a serious problem. The spectral estimate of sampled signals can be calculated via DFT:

$$X(k) = \sum_{n=0}^{N-1} x_n e^{-j\frac{2\pi}{N}nk} \, (n, k = 0, 1, \dots, N-1). \qquad (1)$$

The DFT can effectively be evaluated by the Fast Fourier Transform (FFT). In order to get the value of any point of the DFT, the whole record is needed, without data loss. An obvious solution is to wait for a complete record, but the number of samples which are needed can be the multiple of the DFT record size, which is unacceptable in most applications, where linear or exponential averaging is applied to reduce the measurement noise.

There are available methods which can be used, e.g. Lomb-Sclarge [5][6] or autoregressive analysis [7]. For our research, computationally effective and robust methods are preferred, two of them will be presented briefly. The first one is the extension [1] of the resonator-based observer (RBO) [2]. The second one [3] utilizes the FFT because of its particular efficiency in spectral estimation. This modifies the FFT blocks by zero padding them (here: replacing the samples with zeros) from the first lost sample.

A question arises why we don't replace only the lost samples with zeros. Replacing lost samples results in an additive noise, which can make difficult or impossible to find low magnitude spectral components. The aim of the methods is to reduce the power of this noise.

When data loss arises, time-domain interpolation (e.g., linear from nearest neighbors or Lagrange) is one of the first ideas to consider. However, interpolation methods cause a linear distortion in the spectrum (e.g., linear interpolation distorts the original spectrum with a sinc-squared function), in spite the additive noise of "replacement with zeros" method. Every method which uses surrounding samples gives a kind of memory. This makes the spectrum variable (even if it was constant) which we want to avoid.

In the paper a new method is presented which can be used effectively if the sampling is coherent. It provides the same accuracy as the RBO, but with much less computational complexity.

II. Preliminaries

A. Mathematical Description of Data Loss

1) Indicator Function

Data loss can be modeled with an availability indicator function:

$$K_n = \begin{cases} 1 & \text{, if sample is available at n} \\ 0 & \text{, if sample is lost at n} \end{cases} \qquad (2)$$

Available samples will also be termed as processed samples. Using this we can define the data loss rate:

$$\gamma = \mathrm{P}\{K_n = 0\} \qquad (3)$$

where $\mathrm{P}\{\cdot\}$ is the probability operator. We can describe a signal with lost samples as

$$x_n = K_n x_{0,n}, \qquad (4)$$

where $x_{0,n}$ is the original signal (without data loss).

2) Data Loss Models

There are different data loss models with different indicator functions. Random independent data loss is the simplest. It can be defined as

$$K_n = \begin{cases} 1 & \text{with } \mu = 1 - \gamma \text{ probability} \\ 0 & \text{with } \gamma \text{ probability} \end{cases} \text{ for } \forall n. \qquad (5)$$

In random block-based data loss, a block is formed from each M samples. The same applies to the blocks as in the random, independent case:

$$\alpha_k = \begin{cases} 1 & \text{with } \mu = 1-\gamma \text{ probability} \\ 0 & \text{with } \gamma \text{ probability} \end{cases} \text{ for } \forall k$$
$$K_{kM+n} = \alpha_k \text{ for } \forall k\ \forall (n \in \{0, 1, \dots, M-1\}). \quad (6)$$

In Markov-chain based data loss, the indicator function is generated as the state of a Markov-chain, see Fig. 1.

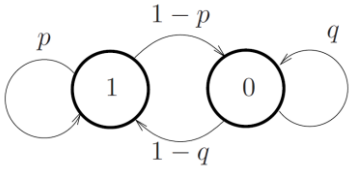


Fig. 1. Two-state Markov-chain. State 1: $K_n = 1$, state 0: $K_n = 0$

B. *Spectral Estimation Using the Resonator-Based Observer [2]*

The resonator-based observer was designed to follow the state variables of the conceptual signal model [2] which generates signals according to their Fourier-series. This way, the observed state variables can be the Fourier coefficients or their rotating versions.

This structure has been modified to be able to handle data loss [1]. The main idea is to modify the conceptual signal model to generate signals with lost samples, then design a state observer for this system. The conceptual signal model can be described as:

$$\mathbf{x}_{n+1} = \mathbf{A}\mathbf{x}_n$$
$$y_n = K_n \mathbf{c}\mathbf{x}_n \quad (7)$$
$$\mathbf{A} = \langle z_i \rangle \quad \mathbf{c} = [1, 1, \dots, 1] \quad z_i = \mathrm{e}^{\mathrm{j}2\pi f_i}$$

where $\langle \cdot \rangle$ is the diagonal matrix formation operator, $\mathbf{x}_n$ is the state vector in the time step n, y_n is the output signal and f_i is the relative frequency of the ith component. Fourier-coefficients can be extracted from the state vector as:

$$\langle z_i^{-n} \rangle \mathbf{x}_n = X_i \quad (8)$$

where X_i is the column vector of the Fourier-coefficients. The equation of the observer is the following:

$$\hat{\mathbf{x}}_{n+1} = \mathbf{A}\hat{\mathbf{x}}_n + \mathbf{g}K_n(y_n - \hat{y}_n) = \mathbf{A}\hat{\mathbf{x}}_n + \mathbf{g}K_n e_n$$
$$\hat{y}_n = \mathbf{c}\hat{\mathbf{x}}_n \quad (9)$$

where $\hat{\mathbf{x}}_n$ is the estimated state vector, $\mathbf{g}$ is the feedback vector, $\hat{y}$ is the estimated signal and e_n is the estimation error. Fig. 2. shows the RBO for signals with lost samples.

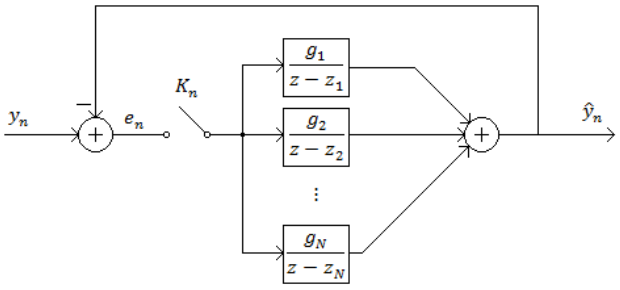


Fig. 2. Resonator-based observer for signals with lost samples.

It is worth noting that the estimation error is multiplied by the indicator function which can be interpreted in two ways. First, if the sample is lost, measurement update isn't performed. Second, at the lost samples the structure acts as its estimate was accurate.

RBO can be used for coherent and incoherent sampling if the frequencies of the signal components are known. The characteristic polynomial can be set arbitrarily with the feedback vector $\mathbf{g}$, which implies, for example, exponential averaging can be done without extra computation. The structure can be applied in real-time and offers fairly precise spectral estimation even at high data loss rate. The main drawback is the complexity: RBO is a quadratic algorithm, while the complexity of FFT-based algorithms is linearithmic ($O(N \log N)$).

C. *Spectral Estimation Using FFT with Zero Padding [3]*

The procedure of spectral estimation using FFT with zero padding for a record is the following:

1. x_n and K_n $(n = 0, 1, \dots, N-1)$ input FFT record and indicator function are given. N is the size of FFT.
2. $L = \min\{N;\ n, \text{where } K_n = 0\}$ is the position of the first lost sample in the block.
3. If $L \leq N_{min}$, the block is discarded. ($N_{min} \geq \frac{N}{4}$ is recommended.) Else, a new indicator function is generated:
$$K'_n = \begin{cases} 1 & \text{, if } n \leq L \\ 0 & \text{, if } n > L \end{cases} \quad (10)$$
4. The signal is multiplied by $\frac{N}{L}$ and the new indicator function is applied:
$$y_n = \frac{N}{L} x_n K'_n \quad (11)$$
5. DFT of y_n is computed, with a window function for L samples, the result is the spectral estimate of the record.

Fig. 3. shows the procedure graphically.

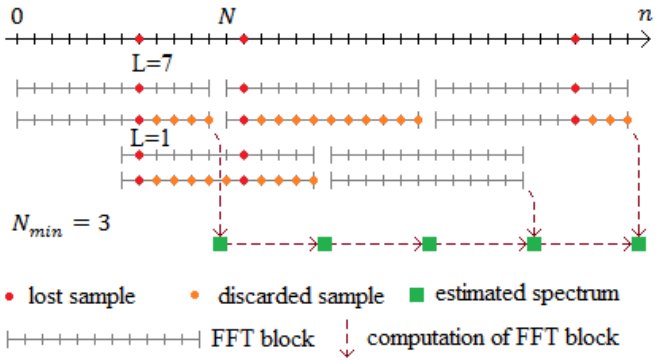


Fig. 3. Spectral estimation with zero padding FFT method

Here overlapping FFT-blocks are used. The aim of this is to make estimate converge faster. Based on [4], the maximal recommended overlap ratio is 75%.

The final spectral estimate is computed from the FFTs of the blocks with an averaging procedure. Averaging can be done both with magnitude spectra and complex spectra. Magnitude spectra can be averaged both for coherently and

incoherently sampled signals, but complex spectra can only be averaged in the case of coherent sampling.

The main benefit of this method is its linearithmic complexity, which makes it easy to apply in real-time measurements. It can be used effectively for searching dominant components even with high data loss rates. If we don't need to know the phase information, this method can be used for both coherently and incoherently sampled signals.

III. Proposed Method: Replacement FFT

A. Description of the Method

The idea is to use FFT for spectral estimation, and to try to achieve the same behavior at lost samples as RBO has. This means that at positions with lost samples, the method needs to behave as its estimate was accurate. This can be done by computing a replacement value for each lost sample via IDFT. The procedure is the following:

1. Wait for the first N samples (FFT block), substitute lost samples with zeros and compute the DFT of the block, the result is FFT_1.
2. Wait for the next FFT block.
3. For the positions of lost samples, compute the replacement value with IDFT from FFT_1.
4. Compute the DFT of the new block, the result is FFT_2.
5. After applying the appropriate phase shift on FFT_2, compute the new spectral estimate from FFT_1 and FFT_2 via exponential averaging, and store it in FFT_1.
6. If the measurement is not over, continue from the second step.

Fig. 4. shows the method graphically.

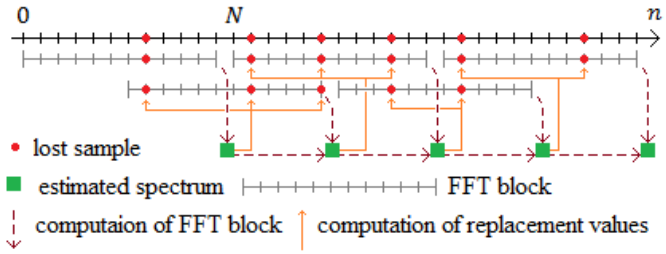


Fig. 4. Spectral estimation with replacement FFT method

It should be noted that in the first FFT block the lost samples are substituted with zeros, thus a noise is added to the initial estimate. It depends on the actual application if it is a problem or not, because exponential averaging reduces this noise over time. If it is a problem, we can wait for the first block without lost samples.

With the phase shift on FFT_2 the phase of the fundamental harmonic (in the DFT base functions) is made equal in FFT_1 and FFT_2. Similar phase shift is necessary at the computation of the replacement values. These can be done only for coherently sampled signals. That's why this method is not applicable for incoherently sampled signals. The phase shifts can be done automatically by implementing the method with a circular buffer.

B. Computation of the Replacement Values

The computation of a single replacement value needs $O(N)$ operations using IDFT. In an N samples long block there are on average γN lost samples, so the replacement values can be computed with $O(\gamma N^2)$ operations. If the data loss rate isn't small enough, these operations make the method complexity quadratic. In this case, IFFT can be used to compute a whole replacement block and use only the necessary positions of it. Of course, this solution needs more memory and at low data loss rates it is slower than individual computation.

It can be easily suspected that there is a data loss rate, where the two procedures have the same computational requirement, this is called critical data loss rate (γ_{crit}). Below it, IDFT, above it, IFFT needs less operations.

An individual replacement value can be computed with $4N$ real operations using IDFT, for the whole block we need $4\gamma N^2$ steps. Assuming the usage of radix-2 IFFT, the replacement block can be computed with $\frac{N}{2}\log_2 N$ complex multiplications and $N\log_2 N$ complex additions, which means $5N\log_2 N$ real operations. We also need to check every position if there was data loss (N operations) and replace the lost samples with the computed values (γN operations). In total, IFFT-based replacement has $5N\log_2 N + N + \gamma N$ steps.

At the critical data loss rate, the two procedures have the same number of operations, from which we obtain

$$\gamma_{crit} = \frac{5\log_2 N + 1}{4N - 1}. \tag{12}$$

Considering that this is only an estimate (e.g. different operations need different number of machine cycles, SIMD instruction execution, etc.), we can rewrite (12) as

$$\gamma_{crit} \approx \frac{5\log_2 N}{4N}. \tag{13}$$

The evaluation of (13) for different N values can be found in Table 1.

TABLE I. Critical Data Loss Rates

N	γ_{crit}	$N\gamma_{crit}$	N	γ_{crit}	$N\gamma_{crit}$
16	31,250%	5	4096	0,366%	15
64	11,719%	7,5	16384	0,107%	17,5
256	3,906%	10	65536	0,031%	20
1024	1,221%	12,5	262144	0,009%	22,5

The first and fourth column show the size of the FFT, the second and fifth ones show the critical data loss rate and the third and last ones show the critical number of lost samples in a block. Based on this, if the data loss rate and the FFT size are known in advance, we can decide which replacement procedure is faster. If data loss rate varies within a large interval which contains γ_{crit} and speed is critical, it should be taken into consideration to count the number of lost samples in each block and use the appropriate method. If this means too much overhead, IFFT-based replacement should be used.

IV. Simulations and Measurement Results

The proposed method was examined and compared with RBO and zero padding FFT via simulations and

measurements. Some results are presented to demonstrate the features of the proposed method.

A. Simulation Results

Simulation parameters: N=256 FFT size, N_{min}=N/4 minimal valid block size for zero padding FFT, 75% overlap ratio, exponential averaging for all three methods with the same time constant (1000), square wave with 1/64 relative frequency input signal with additive white noise (SNR=20 dB), L=50*N simulation time, random independent data loss with 0.1% data loss rate.

Fig. 5. shows the settling of the spectral estimation. The error was formed as the Euclidean (L_2) norm of the difference of the original and the estimated magnitude spectra. It must be noted that the original spectrum was calculated without windowing for RBO and replacement FFT but with Hanning window for zero padding FFT. The reason of this asymmetry is that in zero padding FFT windowing should be used, but it's problematic to use a window function with RBO. Replacement FFT behaves similarly to RBO and provides accurate estimate only with coherent sampling, that's why it doesn't need windowing. The Euclidean norm of the noise magnitude spectrum (Noise FFT) is displayed for comparison. In the bottom, the data availability (K) is shown: high level means available, low level means lost samples.

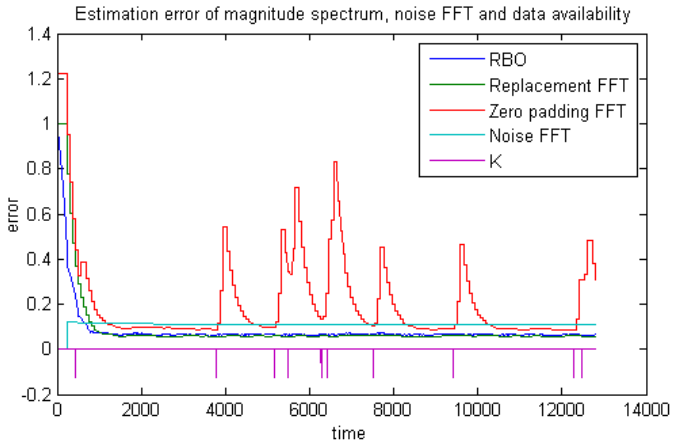


Fig. 5. Simulation results of coherently sampled square wave with 0.1% data loss rate

It can be seen that when there are complete blocks, all the three methods give fairly accurate results. The precision of the replacement FFT is the same as that of the RBO. Data loss leads to a peak in the error of zero padding FFT, but the estimates of replacement FFT and RBO are unaffected.

B. Measurement Results

Measurements were conducted with a Sharc ADSP-21364 Ez-kit Lite DSP board. A noise generator has been used to independently control the data availability.

Fig. 6. shows the measurement results of the processing of a square wave sampled in a special way: the sampling is incoherent for the first harmonic, but coherent for the third harmonic. That's why the third harmonic and its higher harmonics are measured correctly with all the methods. In this measurement, FFT methods were compared with 0.1% data loss rate and N=4096.

The other components are measured incorrectly with the replacement FFT even with applying a window function, because the problem arises from averaging complex spectra. Examining the results of zero padding FFT, it can be stated that windowing should be used with the method.

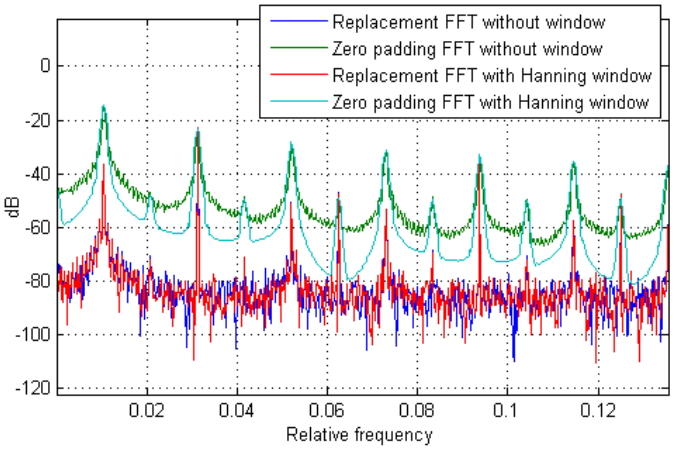


Fig. 6. Measurement results of incoherently sampled square wave with 0.1% data loss (zoomed)

Other measurements have shown that RBO and replacement FFT yield the same results.

V. Conclusion

In this paper, a new method of spectral estimation in the case of data loss was introduced and examined. This method calculates a replacement value for each lost sample from the latest estimate. The replacement FFT can be used effectively for coherently sampled signals, even at high data loss rates and provides distortion-free spectral estimate. This method has linearithmic complexity which makes it beneficial for real-time applications.

References

[1] Gy Orosz, L. Sujbert, G. Péceli, „Analysis of Resonator-Based Harmonic Estimation in Case of Data Loss", IEEE Transactions on Instrumentation and Measurement 62:(2) p. 510-518., 2013

[2] G .Péceli, „A Common Structure for Recursive Discrete Transforms", IEEE Transactions on Circuits and Systems 33:(10) p. 1035-1036., 1986

[3] L. Sujbert, Gy. Orosz, „FFT-based Spectrum Analysis in the Case of Data Loss," IEEE Transaction on Instrumentation and Measurement,vol. 65, no. 5, pp. 968–976, May 2016.

[4] F. Harris, "On the use of Windows for Harmonic Analysis with the Discrete Fourier Transform", In Proc. IEEE, vol. 66, no. 1, pp. 51-83, Jan. 1978.

[5] N. R. Lomb, „Least-Squares Frequency Analysis of Unequally paced Data", Astrophysics And Space Science 39 (1976), pp. 447-462

[6] J. D. Sclarge, „Studies in Astronomical Time Series Analysis. III. Fourier Transforms, Autocorrelation Functions, and Cross-Correlation Functions of Unevenly Spaced Data", Theoretical Studies Branch, Space Science Division, NASA-Ames Research Center, vol. 343 pp.874-887, No.2, 1989

[7] P. M. T. Broersen, S. de Waele, R. Bos, „Application of Autoregressive Spectral Analysis to Missing Data Problems", IEEE Transactions On Instrumentation And Measurement, vol. 53., no. 4, August 2004

Boosting Software Verification with Compiler Optimizations

Gyula Sallai* and Tamás Tóth†
Budapest University of Technology and Economics,
Department of Measurement and Information Systems,
Fault Tolerant Systems Research Group
Email: *salla@sch.bme.hu, †totht@mit.bme.hu

***Abstract*—Unlike testing, formal verification can not only prove the presence of errors, but their absence as well, thus making it suitable for verifying safety-critical systems. Formal verification may be performed by transforming the already implemented source code to a formal model and querying the resulting model on reachability of an erroneous state. Sadly, transformations from source code to a formal model often yield large and complex models, which may result in extremely high computational effort for a verifier algorithm. This paper describes a workflow that provides formal verification for C programs, aided by optimization techniques usually used in compiler design in order to reduce the size and complexity of a program and thus improve the performance of the verifier.**

I. Introduction

As our reliance upon safety-critical embedded software systems grows, so does our need for the ability to prove their fault-free behavior. Formal verification techniques offer reliable proofs of a system's correctness. These algorithms operate on formal models which describe the semantic behavior of the system under verification and are able to answer queries on its properties. However, model-driven design can be rather difficult and the financial and time constraints of a project often do not make it a viable choice for development.

Many projects start right at the implementation phase without sufficient planning and modeling. In the domain of embedded systems, implementation is usually done in C. Although there are many tools that can be used to generate C code from a formal model (model-to-source), the reverse transformation (source-to-model) is far less supported.

Another difficulty with this process is the size of the state space of the model generated from the source code. As most verification algorithms have a rather demanding computational complexity (usually operating in exponential time and beyond), the resulting model may not admit efficient verification. A way to resolve this issue is to reduce the size of the generated model during source-to-model transformation.

The project presented in this paper proposes a *transformation workflow* from C programs to a formal model, known as control flow automaton. The workflow enhances this transformation procedure by applying some common *optimization transformations* used in compiler design [1]. Their application results in a simpler model, which is then split into several smaller, more easily verifiable chunks using the so-called *program slicing* technique [2]. This allows the verification algorithm to handle multiple small problems instead of a single large one. At the the end of the workflow, the resulting simplified slices are verified using *bounded model checking* [3] and *k-induction* [4].

Measurements show that the applied transformations reduced the models' size considerably, making this technique a promising choice for efficient validation. Benchmarks on the execution time of the verifier algorithm suggest that breaking up a larger program into several smaller slices may also speed up the verification process.

†This work was partially supported by Gedeon Richter's Talentum Foundation (Gyömrői út 19-21, 1103 Budapest, Hungary).

II. Background and Notations

There are several program representations with formal semantics suitable for verification. In this paper we shall focus on the one called *control flow automaton* (CFA) [5]. A CFA is a 4-tuple (L, E, ℓ_0, ℓ_e), where

- $L = \{\ell_0, \ell_1, \ldots, \ell_n\}$ is a set of locations representing program counter values,
- $E \subseteq L \times Ops \times L$ is a set of edges, representing possible control flow steps labeled with the operations performed when a particular path is taken,
- $\ell_0 \in L$ is the distinguished entry location, and
- $\ell_e \in L$ is the special error location.

During verification we will attempt to prove that there is no feasible execution path which may reach ℓ_e, thus proving that the input program is not faulty. Let π be a path in a CFA (L, E, ℓ_0, ℓ_e). We say that π is an *error path* iff its last location is ℓ_e. π is an *initial path* iff its first location is ℓ_0. The path π in a CFA is an *initial error path* iff its both an initial path and an error path.

The verification algorithms used in our work are *bounded model checking* and *k-induction*. A bounded model checker [3] (BMC) searchers for initial error paths with the length of k (the *bound*) and reduces them to SMT formulas. If the resulting formula is satisfiable, then its solution will serve as a counterexample to correctness. If no satisfiable formula was found for a length of k, then the algorithm increases the bound to $k + 1$. It repeats this process until it finds a counterexample or reaches a given maximum bound. Bounded model checking is not complete, and can only be used for

finding counterexamples in erroneous programs, as the BMC algorithm always runs into timeout for safe programs.

For proving safety, we can use k-induction [4]. A k-induction model checker applies inductive reasoning on the length of program paths. For a given k, k-induction first proves that all paths from ℓ_0 with the length less than k are safe, using bounded model checking. If the BMC algorithm finds no counterexamples then the algorithm performs an induction step, attempting to prove that a safe path with the length of $k-1$ can only be extended to a safe path with the length of k. This is done by searching for error paths with the length of k and proving that their respective SMT formula is unsatisfiable. If all error paths with the length of k are unsatisfiable then all initial error paths will be unsatisfiable, thus the safety of the system is proved. If a feasible error path exists then it is a counterexample to the induction and the safety of the program cannot be proved of refuted with the bound k.

In order to reduce the resulting model's size, we shall use *optimization transformations* usually known from compiler theory. Compiler optimizations transform an input program into another semantically equivalent program while attempting to reduce its execution time, size, power consumption, etc [1]. In our work we used these transformations to reduce the resulting model's size and complexity. Many of these optimization algorithms are present in most modern compilers. The project presented in this paper focuses on the following algorithms:

- constant folding,
- constant propagation,
- dead branch elimination,
- function inlining.

Constant folding evaluates expressions having a constant argument at compile-time. Constant propagation substitutes constants in place of variables with a value known at compile-time. Both algorithms operate on local and global constants. In many cases, these two algorithms are able to replace one or more branching criteria with the boolean literals **true** or **false**. Dead branch elimination examines these branch decisions and deletes inviable execution paths (e.g. the **true** path of a branch decision always evaluating to **false**). Function inlining is the procedure of replacing a function call with the callee's body. In this work we shall use function inlining to support simple inter-procedural analysis, as an inlined function offers more information of its behavior than a mere function definition.

This work also makes use of an efficient and precise program size reduction technique known as *program slicing*. Weiser [2] suggested that programmers, while debugging a complex program, often dispose code pieces irrelevant to the problem being debugged. This means that programmers usually mentally extract a subset from the entire program relevant to some criteria. He called these subsets *program slices*. Attempting to formalize this practice, Weiser defined a program slice P' as an *executable subset* of a program P, which provides the same output and assigns the same values to a set of variables V as P at some given statement S. This statement S and the variable set V is often put together into a pair which will serve as the *slicing criterion*. By using slicing with multiple criteria, it is possible divide a larger program into several smaller executable slices.

III. Contribution

The project presented in this paper implements a verification compiler, that is a compiler built to support verification. This is done by using a complex workflow which transforms C source code to control flow automata, applying optimization transformations and program slicing during the process. The resulting model(s) can then be verified using an arbitrary verification algorithm. An overview of the workflow can be seen in Figure 1.

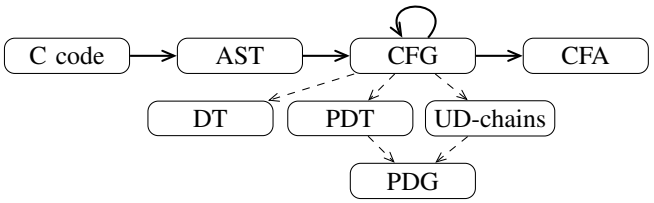


Fig. 1. Transformation workflow.

As an input, the compiler takes a C source code file, which is then parsed and transformed into an *abstract syntax tree* (AST), representing the syntactic structure of the program. This AST is then transformed into a *control flow graph* (CFG), representing the instructions and control flow paths of the program. Optimization algorithms and program slicing are performed on the CFG, resulting in multiple smaller CFG slices of the program. These slices then are transformed into control flow automata. Currently the slicer criteria are the assertion instructions in the control flow graph (which are calls to the `assert` function in C), meaning that each assertion gets its own CFA slice. In the resulting CFA, the error location represents a failing assertion.

Several helper structures are required for these transformations, such as *call graphs* (for function inlining), *use-definition chains* for data dependency information, *dominator trees* (DT) and *post-dominator trees* (PDT) for control structure recognition [1]. The program slicing algorithm requires the construction of a *program dependence graph* (PDG), which is a program representation that explicitly shows data and control dependency relations between two nodes in a control flow graph. The control dependencies show if a branch decision in a node affects whether another instruction gets executed or not. Data dependencies tell which computations must be done in order to have all required arguments of an instruction. Slicing is done by finding the criteria instruction S in the PDG and finding all instructions which S (transitively) depends on [6].

IV. Implementation and Evaluation

The implemented system has three main components: the parser, the optimizer and the verifier. The parser component handles C source parsing and the control flow graph construction. The optimizer module performs the optimization transformations and program slicing and is also responsible for

building the control flow automata from the CFG slices. The verifier component (implemented as a bounded model checker with k-induction) performs the verification on its input model.

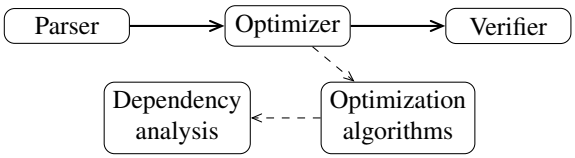


Fig. 2. Architecture of the implemented program.

All components are implemented in Java 8, with dependencies on certain Eclipse[1] libraries. The program also makes use of the theta formal verification framework, developed at the Fault Tolerant Systems Research Group of Budapest University of Technology and Economics. It defines several formal tools (mathematical languages, formal models) and algorithms. It also provides a set of utilities for convenience, such as expression representations and interfaces to SAT/SMT solvers, which are used in the project's implementation. The work discussed here extends this framework with an interface and toolset for C code verification.

The parser module utilizes the parsing library of the Eclipse C/C++ Development Tools plug-in (CDT). The CDT library performs lexing and parsing and returns an abstract syntax tree, which is then transformed into a control flow graph. Currently only a small subset of the C language is supported. The current implementation only allows the usage of control structures (such as **if-then-else**, **do-while**, **switch**, **while-do**, **break**, **continue**, **goto**) and non-recursive functions. Types are only restricted to integers and booleans. Arrays and pointers are not supported at the moment.

The optimizer module handles optimization transformations and program slicing. The implemented transformation algorithms are constant folding, constant propagation, dead branch elimination, function inlining and program slicing. After finishing with the optimization and transformation passes, the optimizer generates a list of control flow automata from each extracted slice. These smaller slices then later will be used as the verifier's input.

Currently the verifier is implemented as a simple bounded model checker extended with a k-induction algorithm. The verifier operates on a collection of control flow automata, with each automaton being a slice extracted from the input program. If a CFA was deemed faulty, then the whole program is reported as erroneous. Currently the verifier may report one of the following statuses: `FAILED` for erroneous programs, `PASSED` for correct programs and `TIMEOUT` if it was not able to produce an answer in a given time limit.

To evaluate the effects of the optimizations mentioned previously, we shall use two types of measurements: the size of the control flow automata used as the verifier input and the results of a benchmarking session on the verifier execution time. The size of an automaton is currently measured by two factors: the number of its locations and edges. The performance benchmarking was performed by measuring the execution time of the verifier on every input CFA. Due to the slicing operation, a single input model may get split into several smaller slices, which then can be verified independently.

The verification task sets are divided into three categories, two of them are taken from the annual *Competition on Software Verification* (SV-COMP) [7]. The first task set, **trivial**, contains trivially verifiable tasks, such as primitive locking mechanisms and greatest common divisor algorithms. The task sets used from the SV-COMP repertoire are the ones called **locks** and **eca**. The **locks** category consists of programs describing locking mechanisms with integer variables and simple **if-then-else** statements.The **eca** (short for *event-condition-action*) task set contains programs implementing event-driven reactive systems. The events are represented by nondeterministic integer variables, the conditions are simple **if-then-else** statements.

The results are shown with two different optimization levels. The first level only uses function inlining, as it is needed for verifying some interprocedural tasks. The second level utilizes all optimizing transformations presented in this paper, including function inlining and program slicing.

The measurement results for each optimization level are shown in different tables. The first column always contains the task name, while the other columns contain the measurement and benchmarking data for a given slice. The legend of column labels is shown in Table I.

TABLE I
COLUMN LABELS AND THEIR ASSOCIATED MEANINGS.

Label	Description
L	CFA location count
E	CFA edge count
R	Verification result (**F**AILED/**P**ASSED/**T**IMEOUT)
ER	Expected verification result (**F**/**P**)
T	Verification execution time (average of 10 instances, in ms)
S	Slice count (for sliced programs)
SL	Average location count (for sliced programs)
SE	Average edge count (for sliced programs)

All models were checked with the timeout of 5 minutes on a x86_64 GNU/Linux (*Arch Linux with Linux Kernel 4.7.6-1*) system with an Intel i7-3632QM 2.20 GHz processor and 16 GB RAM.

TABLE II
BENCHMARK RESULTS WITH INLINING ONLY.

Task	L	E	R	ER	T
triv-lock	8	8	F	F	5
triv-gcd0	11	11	F	F	4
triv-gcd1	9	9	F	F	18
locks05	62	86	T	P	-
locks06	53	73	T	P	-
locks10	98	138	T	P	-
locks14	136	194	F	F	33
locks15	145	207	F	F	36
eca0-label00	391	459	T	F	-
eca0-label20	391	459	T	F	-
eca0-label21	391	459	T	F	-

[1] http://www.eclipse.org/

The benchmark results without any optimization algorithms (except inlining) are summarized in Table II. As it can be seen, the erroneous tasks can usually be verified rather fast, except for the **eca** task set. This set contains models with large if-else constructs inside loops. This yields an exponential number of possible error paths. The bounded model checking algorithm is rather ineffective for such problems and thus, it cannot handle these models in a reasonable amount of time, resulting in a timeout in all cases. It is also worth noting that the k-induction algorithm could not prove the non-faulty models' correctness within the given time frame.

TABLE III
BENCHMARK RESULTS WITH FULL OPTIMIZATION.

Task	S	SL	SE	R	ER	T
triv-lock	1	8	8	F	F	6
triv-gcd0	1	11	11	F	F	4
triv-gcd1	1	9	9	F	F	20
locks05	6	18	23	P	P	9
locks06	5	17	21	P	P	9
locks10	10	22	29	P	P	14
locks14	16	33	45	F	F	25
locks15	17	33	47	F	F	27
eca0-label00	1	309	377	T	F	-
eca0-label20	1	309	377	T	F	-
eca0-label21	1	309	377	T	F	-

Benchmark results with optimization are listed in Table III, which shows the number of produced slices, their average location and edge count (rounded to the nearest integer) and also the verifier running time. As it is sufficient to find one failing assertion among all slices for reporting that the input program is faulty, the running time for erroneous programs is the time until the verifier found the first failing slice. For correct programs, the running time is equals to the sum of the running time for all slices.

Table III shows that while the optimization transformation have little to no effect on trivial programs, it reduces the size of larger programs considerably. While the non-faulty programs of the **locks** category have all ran into timeout without optimization, their verification finished almost instantly after the optimization transformations. Due to the small running time, the other running time measurement differences are within the margin of error.

The tasks of the **eca** set were also reduced considerably, location count is reduced by 21%, edge count is reduced by 17%. Sadly, the verifier algorithm was not able to cope even with the reduced programs of this set, still timing out during verification. As the verification method is completely replaceable and this bounded model checking was merely implemented for the workchains completeness, this is not a large issue. However, further investigation is required for execution time evaluation.

V. CONCLUSIONS AND FUTURE WORK

In this paper we described a transformation workflow for generating multiple smaller optimized formal models from a single C program. To achieve this, the workflow uses optimization algorithms known from compiler theory and the program slicing technique.

The resulting models are then verified using a simple bounded model checking and k-induction algorithm. The developed project was built as modular components, therefore any module can be replaced for further improvement.

The evaluation of the above methods showed that program slicing is promising technique for program size reduction especially for verification. It is also worth noting that splitting a larger problem into multiple ones may allow efficient parallelization of the verification algorithm. As the runtime evaluation proved to be difficult because of the implemented verifiers performance, further evaluation is in order with other, more effective verification algorithms.

The project has several opportunities for improvements and feature additions. Some of them are listed below.

- Extending the support for more features of the C language. Such features could be arrays, pointers, structs.
- Introducing other optimization algorithms into the workflow, such as interprocedural slicing, or more aggressive slicing methods such as value slicing [8].
- Currently the counterexample is only shown in the verified formal model. The additions of traceability information would allow showing the counterexample in the original source code.
- The LLVM compiler infrastructure framework[2] provides a language-agnostic intermediate representation (*LLVM IR*) for several programming languages. Adding support for the LLVM IR would extend the range of supported languages and would also implicitly add multiple fine-tuned optimizations into the workflow.

REFERENCES

[1] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1986.

[2] M. Weiser, "Program slicing," in *Proceedings of the 5th International Conference on Software Engineering*, ser. ICSE '81. IEEE Press, 1981, pp. 439–449.

[3] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science. Springer, 1999, vol. 1579, pp. 193–207.

[4] M. Sheeran, S. Singh, and G. Stålmarck, "Checking safety properties using induction and a sat-solver," in *Proceedings of the Third International Conference on Formal Methods in Computer-Aided Design*, ser. FMCAD '00. Springer-Verlag, 2000, pp. 108–125.

[5] D. Beyer, A. Cimatti, A. Griggio, M. E. Keremoglu, and R. Sebastiani, "Software model checking via large-block encoding," in *Formal Methods in Computer-Aided Design*. IEEE, 2009, pp. 25–32.

[6] J. Ferrante, K. J. Ottenstein, and J. D. Warren, "The program dependence graph and its use in optimization," *ACM Trans. Program. Lang. Syst.*, vol. 9, no. 3, pp. 319–349, Jul. 1987.

[7] D. Beyer, "Reliable and reproducible competition results with benchexec and witnesses (report on SV-COMP 2016)," ser. Lecture Notes in Computer Science, M. Chechik and J.-F. Raskin, Eds. Springer Berlin Heidelberg, 2016, vol. 9636, pp. 887–904.

[8] S. Kumar, A. Sanyal, and U. P. Khedker, "Value slice: A new slicing concept for scalable property checking," in *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems - Volume 9035*. Springer-Verlag New York, Inc., 2015, pp. 101–115.

[2] http://llvm.org/

Securing Critical Systems through Continuous User Authentication and Non-repudiation

Enrico Schiavone
Department of Mathematics and Informatics, University of Florence
Viale Morgagni 65, 50134, Florence, Italy
enrico.schiavone@unifi.it

***Abstract*— Providing a mechanism for authenticating a user's access to resources is very important, especially for systems that can be considered critical for the data stored and the functionalities offered. In those environments, traditional authentication mechanisms can be ineffective to face intrusions: they usually verify user's identity only at login, and even repeating this step, frequently asking for passwords or PIN would reduce system's usability. Biometric continuous authentication, instead, is emerging as viable alternative approach that can guarantee accurate and transparent verification for the entire session: the traits can be repeatedly acquired avoiding disturbing the user's activity. Another important property that critical systems may need to be guaranteed is non-repudiation, which means protection against the denial of having used the system or executed some specific commands with it. The paper focuses on biometric continuous authentication and non-repudiation, and it briefly presents a preliminary solution based on a specific case study. This work presents the current research direction of the author and describes some challenges that the student aims to address in the next years.**

Keywords—authenticity; non-repudiation; continuous authentication; biometrics; security;

I. INTRODUCTION

In the last decades, the constant growth and diffusion of Information and Communications Technology (ICT) contributed to make people's life easier. Today, users and operators can exploit technologies to share confidential data from a long distance or to execute critical commands in real-time. However, the need for security services has gone hand in hand with the technological progress.

Especially when some operation is considered highly critical, preventing unauthorized access can avoid undesirable consequences or even catastrophes. The system in charge to execute an operation has to verify that the involved users are really who they claim to be, before giving them the permission to accomplish the action.

Authentication is the process of providing assurance in the claimed identity of an entity. The identity verification is obtained exploiting a piece of information and/or a process called *authentication factor* that belongs to one of the following categories: knowledge (e.g. password, PIN); possession (e.g. passport, private key); inherence (biometric characteristics, physiological or behavioral, e.g. fingerprint or keystroke).

Traditionally this verification is based on pairs of username and password and performed as a single-occurrence process, only at login phase. No checks are executed during sessions, which are terminated by an explicit logout or expire after an idle activity period of the user. Instead, if the operation covers a long period, it may be necessary to repeat the authentication procedure; however, asking for passwords and secrets several times requires users' active participation, and it may disturb their main activity. In order to design an effective continuous authentication mechanism for critical systems, together with security, also usability has to be taken into account.

To prevent unauthorized access of ICT systems, solutions based on *biometric continuous authentication* have been studied in literature. They modify user identity verification from a single-occurrence to a continuous process [1], [2]. To enhance security, authentication can also exploit multiple traits, being multimodal; in fact it has been verified that using various biometric traits, properly combined, can improve the performance of the identity verification process. In addition, with appropriate sensors, some biometric traits can be acquired transparently.

However, most of the existing solutions suffer from high computational overhead or their usability has not been adequately substantiated. Our goal is to design a multi-biometric continuous authentication system that is usable, incurs in little system overhead and permits to easily manage the trade-off between security and usability through configuration parameters.

Besides authentication, being able to demonstrate user involvement in the usage of a system or application can also be useful. In fact, when a dispute arises or a disaster happens people may try to deny their involvement and to repudiate their behavior.

Repudiation can be defined as the denial of having participated in all or part of an action by one of the entities involved. Consequently *non-repudiation* is the ability to protect against denial by one of the entities involved in an action of having participated in all or part the action.

A non-repudiation mechanism should guarantee the establishment of the facts even in front of a court of law. Therefore, a non-repudiation service can be useful both as a

mean to obtain accountability as well as a deterrent for deliberate misbehaviors.

This paper presents the research plan of a second year Ph.D. student and it follows [13] and [14]. The objective of the research direction identified is to study, define, and possibly test, mechanisms that can offer authentication and non-repudiation, with the aim to provide trustworthy security services for ICT systems.

The present work is focused on biometric continuous authentication and describes a case study regarding control room workstations, in which traditional mechanisms -i.e. password-based authentication- are not sufficient for the expected requirements. In addition, it addresses the issue of repudiation and study scenarios, and possible solutions in which a biometric-based non-repudiation service can help solving disputes between entities.

The paper proceeds as follows: Section II presents our contribution in providing continuous authentication, briefly describing the approach we followed, some results regarding its usability and the ongoing work related to risk assessment; Section III concentrates on non-repudiation, its connection with biometrics and introduces some scenarios.

II. Biometric Continuous Authentication of Control Room Operators

A. Context and Requirements

Control room operators are a category of users that can access potentially sensitive information to issue critical commands for the entire working session. They are also directly responsible for such commands and for the data accessed, modified and deleted.

For instance, transportation (e.g. airways, railways), electric power generation, military or aerospace operations are some contexts in which control rooms are often adopted. Operators are in charge of analyzing and interpreting situations that describe the current status of events and activities. They are also able to command intervention teams on field, or to dispatch instructions in a target area. It is required to protect the control rooms and their workstations from unauthorized people, both *intruders* and *insiders*, that may want to acquire privacy-sensitive data, disrupt the operations, disseminate false information, or simply commit errors which will be ascribed to the operator in charge of the workstation.

Consequently, in order to protect the workstations, we need to guarantee *authenticity* and *non-repudiation* of the commands/functions executed, meaning that the identity of the worker which sends the commands from a workspace should be properly verified and they cannot deny that action.

In addition, the workspace should be usable for the legitimate worker: the security mechanism should not disturb or excessively slow down the working activity of the operator. For that reason the verification process should be transparent.

B. The Proposed Continuous Authentication Protocol

To comply with the above requirements we defined a client-server multimodal continuous authentication protocol (for further details on requirements please refer to [3]). The overall architecture of the biometric system is composed of the operator workstation and the connected sensors required for acquiring the biometric data. It is based on three biometric subsystems, for face recognition, fingerprint recognition and keystroke recognition.

The protocol is shown in the sequence diagram of Fig. 1. and is divided in two phases: the initial phase and the maintenance phase.

Initial phase. It is composed of the following steps:

- The user logs in with a strong authentication or a successful biometric verification executed with all the three subsystems in a short time interval.
- Biometric data is acquired by the workstation and transmitted to the authentication server.
- The authentication server matches the operator's templates with the traits stored in a database and verifies his/her identity.
- In case of successful verification, the Critical System establishes a session and allows all restricted functions expected for the operator's role.
- The authentication server computes and updates a *trust level* that decreases as time passes; the session expires when such level becomes lower than a threshold.

Maintenance phase.

- The authentication server waits for fresh biometric data, from any of the three subsystems.
- When new biometric data is available, the authentication server verifies the identity claimed by the operator and, depending on the matching results of each subsystem, updates the trust level.
- When the trust level is close to the threshold, the authentication server may send a notification to the operator, to signal that the session will expire soon.
- When the trust level is below the threshold, the Critical System disables the restricted functions,

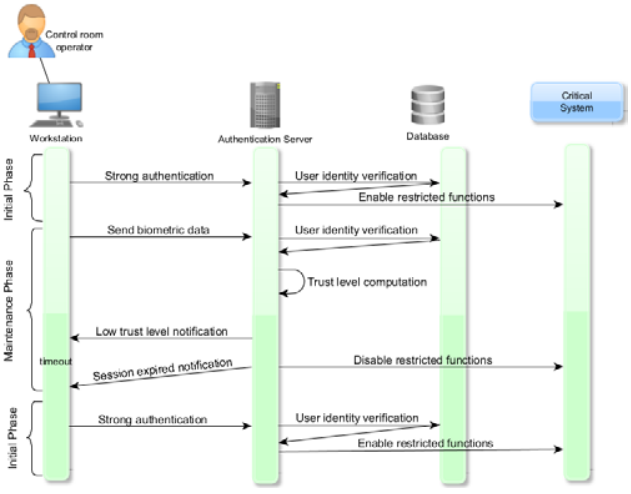


Fig. 1 Sequence Diagram of the Protocol

which will be available again only when the operator restarts from the initial phase.

No active participation of the operator is necessary, which only needs to use the mouse – that should incorporate a fingerprint scanner at the place where users would normally place their thumb-, a keyboard, or to be positioned in front of a webcam.

More details about the protocol, the algorithm for trust level computation the prototype realized and the software implemented can be found in [3].

C. Usability and Risk Assessment

To investigate the usability of the system, we are conducting an experiment (which in part is a replication of [10]) involving a wide group of participants, asking them to complete four tasks on a workstation provided with our continuous authentication application running in background. First, we want to measure the *effectiveness* of our solution, calculating the FAR (False Acceptance Rate), and the FRR (False Rejection Rate) for each of the biometric subsystem, and for the main biometric continuous authentication system.

Then, we are going to measure the *efficiency* of the system, tracking the time interval between the initial authentication and any unexpected termination (meaning that the trust level has fallen below the threshold). Similarly, we are interested in the time necessary to the authentication system to reject an impostor. The user *satisfaction* will be measured with a questionnaire. In addition to usability testing, we want to clarify if the *overhead* introduced by the continuous authentication system can slow down the workstation and consequently increase the users' required effort. Another main goal is to perform the specified measurements with different *parameters configuration*, e.g. varying the trust threshold (the minimum trust level allowed to remain authenticated).

Preliminary results are in [11]. They show that the system appears to be secure and usable, and there is every chance to increase its usability integrating three highly accurate recognition algorithms. In fact, 75% of the users completed the tests without unexpected expirations, and this result is interesting if compared with the previous studies. As expected, modifying parameters we were able to obtain a highly usable configuration, with which the users remained authenticated for the whole duration of the session. In terms of resources utilization, Biometric Continuous Authentication System did not have any significant impact on task performance, and its overhead was negligible.

We are also conducting a NIST-compliant qualitative risk assessment for the biometric continuous authentication protocol [15]. The activity focuses on both threats related to transmission and specific for the biometric system level. The goal is to establish its strengths, weaknesses and consequently understand the countermeasures needed in order to improve the security of our authentication solution.

The proposed protocol addressed the problem of non-repudiation exploiting the biometric nature of the credentials, which are supposed to provide it inherently. However, this is still under discussion, as described in Section IV; for this reason we are working on improvements that should fully guarantee non-repudiation.

III. NON-REPUDIATION

Explanatory tests show that with our solution for continuous authentication, the *authenticity* of control room operators is guaranteed. However, although with this solution it appears very hard for the user to deny having accessed the system, the deniability is related to error rates: is an intruder still able to repudiate actions?

Trying to directly address this problem, we aim to discuss if a continuous authentication mechanism, based on the usage of biometric traits, provides sufficient undeniable evidence of user's participation in an action.

A. Biometrics Can Guarantee Non-Repudiation?

According to the author of [12], unlike passwords and tokens, biometrics - because of their strong binding to specific persons- is the only authentication factor capable of guaranteeing that authentication cannot subsequently be refused by a user.

In [4] the author claims that for authentication mechanisms, non-repudiation depends on: (i) The ability of the authentication mechanism to discriminate between individuals; (ii) The strength of binding between the authentication data and the individual in question; (iii) Technical and procedural vulnerabilities that could undermine the intrinsic strength of the binding; (iv) Informed consent of the individual at the time the authentication is given.

In addition, the discrimination capabilities of biometrics depend on the technology used and on other application-related factors, that are quantified in terms of error rates (FAR and FRR) [4]. Despite biometric traits are sometimes presented in the computer security literature as an authentication factor that may solve the repudiation problem [12], [4], other works like [5], [6] draw completely different conclusions. Analyzing the state of the art, we can state that the answers to this question are contradictory.

However, the situation changes if biometric authentication is coupled with another security mechanism like digital signature, which is commonly considered as the standard approach to achieve non-repudiation. In fact, public key infrastructure, or PKI, and biometrics can well complement each other in many security applications [7].

Apart from biometrics, our opinion is that a non-repudiation service should be capable of:

- Reliably (and if necessary continuously) verifying the user's identity. In other words, we think that non-repudiation is impossible without authentication.
- Generating an undeniable and unforgeable evidence of the action and bind it with the user's identity.

B. Further Non-repudiation Scenarios

There are many actions that an individual or an entity may want to deny, e.g. for economic reasons, to fraud

someone or to hide a malpractice. The most studied non-repudiation protocols in the state of the art regard the transactions and exchange of messages scenario [8], [9].

Usually a basic transaction is defined as the transferring of a message M from user A to user B, and the following are the typical disputes that may arise:

- A claims that it has sent M to B, while B denies having received it;
- B claims that it received M from A, while A denies sending it; and
- A claims that it sent M before a deadline T, while B denies receiving it before T.

Transactions, especially in the e-commerce field, are often denied by consumers. According to The New York Times, 0.05% of MasterCard transactions worldwide are subjects of disputes, that probably means around 15 million questionable charges per year. Analysts, in general, estimate that 20% of disputes involve fraud. Providing a non-repudiation service for this scenario and solving those disputes, would probably make issuers save a lot of money.
Non-repudiation services can cover other kind of actions, not only transactions. In fact, there are many scenarios in the field of information exchange that may be better protected with proper authentication and non-repudiation services. Changing the mean of communication, the nature of exchanged data or the kind of information flow (i.e. one-time occurrence or continuous), we can distinguish several issues to address and related solutions. For instance, e-mails, instant messaging, VoIP communications or accessing files stored in a private area on a server are some of the possible scenarios. In general, what the service should generate is undeniable evidence that can be used if a dispute arises. Evidence is a crucial object, and sometimes has to be processed by a Trusted Third Party (TTP) [8].

IV. Conclusions and Future Works

Security is a fundamental property in the ICT field, especially for critical systems and applications in which confidential data are managed and where unauthorized accesses and behaviors can cause undesirable consequences or even catastrophes. In this context, *authentication* and *non-repudiation* are common requirements. The aim of our research is to study approaches to guarantee them. First, we are planning to integrate an existing biometric continuous authentication mechanism [2] with a non-repudiation service and our solution will probably combine biometric continuous authentication with digital signature.

Finally, another ongoing activity is investigating if biometrics-based solutions permit to obtain irrefutable evidence of user identity: for different scenarios we will study which biometric trait –single or combined- can be appropriate, also considering the error rates that may be admissible, the technological or environmental limitations and the user acceptability. A strategy can be searching a set of the most accurate biometric verification algorithms in literature (e.g. exploiting initiatives like [16]), and trying to evaluate the probability of successful non-repudiation for a user of a continuous authentication system based on a combination of those algorithms.

Acknowledgments

The author would like to thank his supervisor Prof A. Bondavalli and co-supervisors, Dr. A. Ceccarelli and Prof. A. Carvalho that are assisting this research providing insight and expertise. This work has been supported by the European FP7-IRSES project DEVASSES and by the Joint Program Initiative (JPI) Urban Europe via the IRENE project.

References

[1] S. Kumar, T. Sim, R. Janakiraman, and S. Zhang, “Using continuous biometric verification to protect interactive login sessions,” In: 21st Annual Computer Security Applications Conference (ACSAC), pp. 441-450, 2005.

[2] A. Ceccarelli, L. Montecchi, F. Brancati, P. Lollini, A. Marguglio, and A. Bondavalli, “Continuous and transparent user identity verification for secure internet services,” In: IEEE Transactions on Dependable and Secure Computing, 12(3), pp. 270-283, 2015.

[3] E. Schiavone, A. Ceccarelli, and A. Bondavalli, “Continuous user identity verification for trusted operators in control rooms,” Proc. of ICA3PP, pp. 187-200, 2015.

[4] H. Bidgoli, “Handbook of information security, Threats, Vulnerabilities, Prevention, Detection, and Management”. vol. 3. 2006.

[5] D.R. Kuhn, V.C. Hu, W.T. Polk, and S.J. Chang, “Introduction to public key technology and the federal PKI infrastructure”. National Inst of Standards and Technology Gaithersburg MD, 2001.

[6] A. Kholmatov, and Y., Berrin, "Biometric cryptosystem using online signatures." *Computer and Information Sciences–ISCIS 2006*. Springer Berlin Heidelberg, 2006. 981-990.

[7] H. Feng, and C. Choong Wah, "Private key generation from on-line handwritten signatures." *Information Management & Computer Security* 10.4 (2002): 159-164.

[8] J.A. Onieva, J. Zhou, and J. Lopez. "Multiparty nonrepudiation: A survey." *ACM Computing Surveys (CSUR)* 41.1 (2009): 5.

[9] S. Kremer, O. Markowitch, and J. Zhou, “An intensive survey of fair non-repudiation protocols”, 2002.

[10] G. Kwang, R.H. Yap, T. Sim, and R. Ramnath, “An usability study of continuous biometrics authentication”. In *Advances in Biometrics* (pp. 828-837). Springer Berlin Heidelberg, 2009.

[11] E. Schiavone, A. Ceccarelli, A. Bondavalli, and A. Carvalho “Usability Assessment in a Multi-biometric Continuous Authentication System” Proc. of Dependable Computing (LADC), 2016 Seventh Latin-American Symposium on, 43-50, 2016.

[12] Li, Stan Z. “Encyclopedia of biometrics”: I-Z. Vol. 1. Springer Science & Business Media, 2009.

[13] E. Schiavone “Providing Continuous Authentication and Non-Repudiation Security Services” Student Forum of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, 2016

[14] E. Schiavone, A. Ceccarelli, and A. Bondavalli “Continuous Authentication and Non-repudiation for the Security of Critical Systems, "Reliable Distributed Systems (SRDS), 2016 IEEE 35th Symposium on", 207-208, 2016.

[15] E. Schiavone, A. Ceccarelli, and A. Bondavalli “Risk Assessment of a Biometric Continuous Authentication Protocol for Internet Services” to appear in Proc. of ITASEC17 The Italian Conference on cybersecurity, 2017.

[16] Kemelmacher-Shlizerman, Ira, et al. "The megaface benchmark: 1 million faces for recognition at scale." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016.

Block-Oriented Identification using the Best Linear Approximation: Benefits and Drawbacks

Maarten Schoukens
Dept. ELEC, Vrije Universiteit Brussel, Belgium
Email: maarten.schoukens@vub.ac.be

***Abstract*—Due to their simplicity and structured nature, block-oriented models are popular in nonlinear modeling applications. A wide range of block-oriented identification algorithms were developed over the years. One class of these approaches uses the so-called best linear approximation to initialize the identification algorithm. The best linear approximation framework allows the user to extract important information about the system, it guides the user in selecting good candidate model structures and orders, and it proves to be a good starting point for nonlinear system identification algorithms. This paper gives an overview of the benefits and drawbacks of using identification algorithms based on the best linear approximation.**

I. Introduction

Nonlinear models are often used these days to obtain a better insight in the behavior of the system under test, to compensate for a potential nonlinear behavior using predistortion techniques, or to improve plant control performance. Popular nonlinear model structures are, amongst others, nonlinear state-space models [1], NARMAX models [2], and block-oriented models [3]. This paper focuses on the block-oriented class of models.

Many different types of block-oriented identification algorithms exist [3], where the linear-approximation based algorithms are amongst the more popular. One particular method to obtain such a linear approximation is the Best Linear Approximation (BLA) framework. This paper discusses the benefits and drawbacks of using BLA-based block-oriented system identification algorithms.

II. Block-Oriented Systems

Block-oriented models are constructed starting from two basic building blocks: a linear time-invariant (LTI) block and a static nonlinear block. Due to the separation of the nonlinear dynamic behavior into linear time invariant dynamics and the static nonlinearities, block-oriented nonlinear models are quite simple to understand and easy to use. They can be combined in many different ways. Series, parallel and feedback connections are considered in this paper, resulting in a wide variety of block-oriented structures as is depicted in Figure 1. These block-oriented models are only a selection of the many different possibilities that one could think of. For instance the generalized Hammerstein-Wiener structure that is discussed in [4] is not considered in this paper.

The LTI blocks and the static nonlinear blocks can be represented in many different ways. The LTI blocks are most often represented as a rational transfer function. The model order selection of the order of the numerator and denominator of the different blocks is a challenging problem. The static nonlinear block can again be represented in a nonparametric way using, for instance, kernel-based methods, or in a parametric way using, for instance, a linear-in-the-parameters basis function expansion (polynomial, piecewise linear, radial basis function network, ...), neural networks, or other dedicated parametrizations for static nonlinear functions. Again, in the parametric case, the nonlinear function complexity (number of basis functions, neurons, ...) needs to be selected by the user.

Another issue of block-oriented models is the uniqueness of the model parametrization. Gain exchanges, delay exchanges and equivalence transformations are present in many block-oriented structures [5]. This results in many different models with the same input-output behavior, but with a different parametrization.

It is assumed throughout this paper that a Gaussian additive, colored zero-mean noise source $n_y(t)$ with a finite variance σ^2 is present at the output of the system only:

$$y(t) = y_0(t) + n_y(t). \tag{1}$$

This noise $n_y(t)$ is assumed to be independent of the known input $u(t)$. The signal $y(t)$ is the actual output signal and a subscript 0 denotes the exact (unknown) value.

III. Best Linear Approximation

A linear model often explains a significant part of the behavior of a (weakly) nonlinear system. This approximative linear model also provides the user with a better insight into the behavior of the system under test. It motivates the use of a framework that approximates the behavior of a nonlinear system by a linear time invariant model. This paper considers the Best Linear Approximation (BLA) framework [6], [7] to estimate a linear approximation of a nonlinear system..

The BLA is best in mean square sense for a fixed class of input signals $\mathbb{U}$ only, it is defined in [6], [7] as:

$$G_{bla}(q) \triangleq \arg\min_{G(q)} E\left\{|\tilde{y}(t) - G(q)\tilde{u}(t)|^2\right\}, \tag{2}$$

where $E\{.\}$ denotes the expected value operator. The expected value $E\{.\}$ is taken w.r.t. the random input $\tilde{u}(t)$. The zero-mean signals $\tilde{u}(t)$ and $\tilde{y}(t)$ are defined as:

$$\tilde{u}(t) \triangleq u(t) - E\{u(t)\}, \tag{3}$$

$$\tilde{y}(t) \triangleq y(t) - E\{y(t)\}. \tag{4}$$

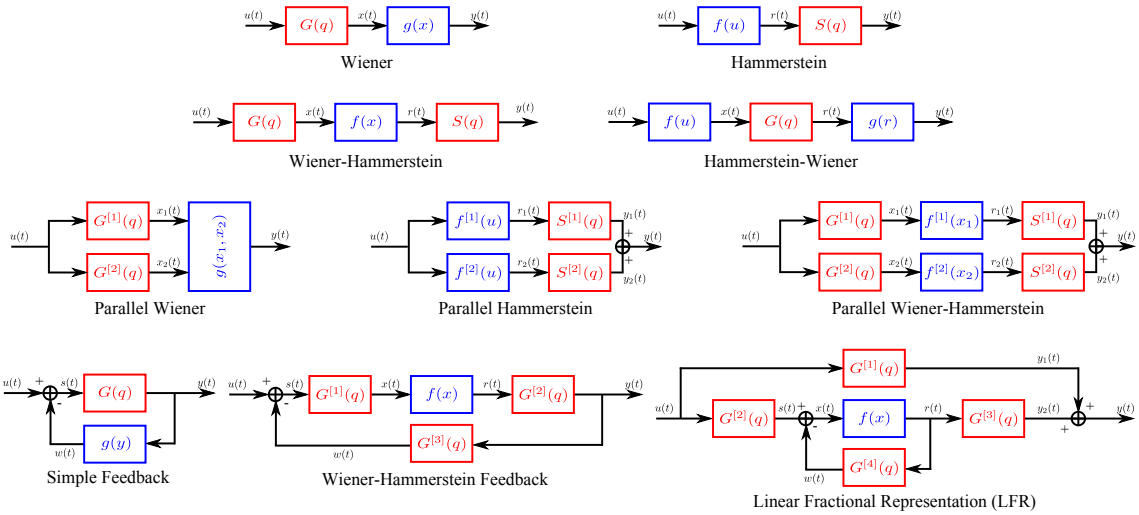


Fig. 1. An overview of possible block-oriented structures. The different structures are obtained by using series, parallel and feedback connections of LTI blocks ($G(q)$ and $S(q)$) and static nonlinear blocks ($f(\cdot)$ and $g(\cdot)$). There are three types of structure classes: single branch structures (Wiener, Hammerstein, Wiener-Hammerstein and Hammerstein-Wiener), parallel branch structures (parallel Wiener, parallel Hammerstein, parallel Wiener-Hammerstein) and feedback structures (simple feedback structure, Wiener-Hammerstein feedback and LFR).

The error in eq. (2) is minimized by [6], [7]:

$$G_{bla}(e^{j\omega T_s}) = \frac{S_{\tilde{Y}\tilde{U}}(e^{j\omega T_s})}{S_{\tilde{U}\tilde{U}}(e^{j\omega T_s})}, \tag{5}$$

where $S_{\tilde{Y}\tilde{U}}$ and $S_{\tilde{U}\tilde{U}}$ denote the crosspower of $\tilde{y}(t)$ and $\tilde{u}(t)$ and autopower of $\tilde{u}(t)$ respectively.

The BLA of a system depends on the signal class $\mathbb{U}$ that is used. This work considers $\mathbb{U}$ to be the Riemann equivalence class of asymptotically normally distributed excitation signals. The signal class $\mathbb{U}$ contains Gaussian noise sequences, but contains also periodic signal sets known as random phase multisines [7]. This class of signals will be referred to as Gaussian signals in the remainder of this paper. When stationary Gaussian excitation signals are used, the BLA of many block-oriented systems becomes a simple function of the linear dynamics that are present in that system.

A. BLA of Single Branch Structures

The BLA of Wiener, Hammerstein and Wiener-Hammerstein structures are a very simple expression of the LTI-blocks present in the block-oriented system when Gaussian excitation signals are used. Due to Bussgang's Theorem [8] one obtains the following expression for the BLA of a Wiener-Hammerstein structure [9]:

$$G_{bla}(q) = \lambda G(q)S(q), \tag{6}$$

where q^{-1} is the backwards shift operator, and λ is a gain depending on the system and considered class of input (input power, offset and power spectrum). Note that the poles and zeros of the BLA are the poles and zeros of the LTI blocks present in the system [9].

B. BLA of Parallel Branch Structures

The output of a parallel branch structures is the summation of multiple single branch system, hence, the BLA of a parallel Wiener-Hammerstein system is given by:

$$G_{bla}(q) = \sum_{i=1}^{n_{br}} \lambda_i G^{[i]}(q)S^{[i]}(q). \tag{7}$$

Note that zeros of the BLA depend on the gains λ_i, while the poles of the BLA are the poles of the LTI blocks of the parallel Wiener-Hammerstein system [10].

C. BLA of structures containing feedback

Bussgangs Theorem cannot be used anymore in the case of nonlinear feedback structures since the input of the static nonlinear block is not Gaussian anymore. Therefore, only approximate expressions of the BLA are given here. The BLA of a simple feedback structure is approximately given by [1]:

$$G_{bla}(q) \approx \frac{G(q)}{1+\lambda G(q)}. \tag{8}$$

The case of the LFR structure is more involved [11]:

$$G_{bla}(q) \approx G^{[1]}(q) + \frac{G^{[2]}(q)G^{[3]}(q)}{1+\lambda G^{[4]}(q)}. \tag{9}$$

It can be observed that the poles of the BLA of a simple feedback structure depend on the gain $lambda$. In the case of the LFR structure both the poles and the zeros depend on the gain $lambda$.

IV. Detection of Nonlinearity

Although one might know beforehand that a given system is nonlinear, it can very well turn out that, for the class of signals that will realistically act on the system, and for the frequency region of interest, and application on hand, no significant

nonlinear behavior is observed. In such a case, much modeling effort can be spared by simply estimating the BLA of the system, and using it later on for its intended task (control design, simulation, system analysis, ...).

The BLA framework allows a user to detect and quantify the level of the nonlinear distortions. Based on this analysis one can see, for a chosen class of input signals, if the effects of the nonlinearity are dominant or not, in which frequency region the nonlinearity is active, and how much can be gained by estimating a nonlinear model [7].

V. Model Order and Model Structure Selection

The model structure and model order selection problem is a though challenge in many nonlinear system identification problems. Given an input/output dataset the user has to decide what type of nonlinear model will be used (e.g. Hammerstein, Wiener, nonlinear feedback, ...) and which model orders (e.g. orders of the dynamics and degree of the static nonlinearity) are to be selected.

A. Model Order Selection

The model order selection problem in block-oriented modeling problems is much harder than the one in the LTI framework. One needs to select the model order of each LTI block separately and on top of this, also the complexity of the static nonlinearity needs to be decided on. Using the BLA framework to start the modeling of a block-oriented system allows one to extract the model orders of the combined linear dynamics present in the block-oriented structure. Indeed, the BLA is in many cases a simple function of the underlying linear blocks of the block-oriented system under test (see Section III).

An important part of the model order selection problem, the selection of the order of the dynamics of the system, is now taking place in a linear framework on the BLA, separate from the nonlinearity selection problem. This results in a problem which is much more simple and better understood by many researchers and practitioners.

The selected model orders of the BLA can be used directly in the nonlinear modeling step (Hammerstein, Wiener, Simple Feedback structure), or they are translated automatically in a second step into the model orders of each LTI block separately using either pole-zero allocation algorithms [9], [10] to split the dynamics over the front and the back (Wiener-Hammerstein, parallel Wiener-Hammerstein and Wiener-Hammerstein Feedback structure), singular value decomposition approaches [10], [12] to split the dynamics over the parallel branches (parallel Hammerstein, parallel Wiener and parallel Wiener-Hammerstein), or by solving a Riccati equation in the case of the LFR structure [11].

B. Model Structure Selection

The model structure selection problem can also be tackled in part by taking a closer look at the expression of the BLA for the different block-oriented model structures. It is discussed in [13] how the BLA behaves when it is estimated at different setpoints of the system (different input amplitudes, constant offsets or power spectra).

Single-branch system structures such as the Wiener-Hammerstein structure will only exhibit a varying gain over the different BLA setpoints, while parallel branch systems exhibit varying zeros and feedback structures exhibit varying poles over the different BLA setpoints (see Table I). Note that the LFR structure is both a parallel branch and a nonlinear feedback structure. Hence, both poles and zeros of the BLA will depend on the input power, constant offset and power spectrum. This analysis demonstrates how one can quickly detect some important structural features of the nonlinear system under test using only linear approximations of that system.

VI. Drawbacks of BLA-Based Modeling

Of course, obtaining a sufficiently high-quality estimate of the BLA (sufficiently low variance on the BLA) comes at a cost. The variance on the estimated BLA depends on how nonlinear the system under test is. If the system is very nonlinear, a significant error is introduced when the least-squares linearization is performed. This results in a high variance on the estimate. The classical approach to lower the variance on the estimated BLA is to use more input-output data. Hence, it can be the case that to model a strongly nonlinear system using the BLA, a larger dataset is required compared to some of the approaches that take the nonlinearity directly into account.

Another issue can be the presence of nonlinearities which give rise to a BLA equal to zero over all frequencies. Of course this is input dependent: this problem can be circumvented by doing measurements at different constant offset of the input signal. For example, the BLA of an even nonlinearity using a zero-mean Gaussian input is equal to zero [8]. However, when an constant offset is added to this input signal a non-zero BLA is obtained.

A last remark concerns the systems with nonlinear feedback (simple feedback structure, Wiener-Hammerstein feedback and LFR). The BLA expressions given in this paper for these systems are not exact. Although they are a good approximation of reality, more involved effects come into play due to the non-Gaussianity of the signal at the input of the static nonlinearity. However, it is observed in many practical applications that the simplification used in this paper does lead to good model estimates, e.g. [1].

Note that, aside the drawbacks listed above, many other challenges exists. The selection of the nonlinearity present in the model, the validation of the system structure over a wide range of use, dealing with model errors in a proper manner, using more involved noise frameworks, and many more are all open problems in nonlinear system identification.

VII. Example: Silverbox

The Silverbox system (an electronic realization of the duffing oscillator) is studied (see for instance [1]) here as a simple illustration of the theory explained in the previous sections.

TABLE I
MODEL STRUCTURE SELECTION USING THE BLA BY OBSERVING THE GAIN, POLES AND ZEROS OF THE BLA ESTIMATED AT MULTIPLE SETPOINTS OF THE SYSTEM (W-H STANDS FOR WIENER-HAMMERSTEIN).

	LTI	Wiener	Hammerstein	W-H	Parallel W-H	Simple Feedback	W-H Feedback	LFR
Gain	fixed	varying	varying	varying	varying	varying	varying	varying
Poles	fixed	fixed	fixed	fixed	fixed	varying	varying	varying
Zeros	fixed	fixed	fixed	fixed	varying	fixed	fixed	varying

As a first step the nonparametric BLA is estimated. The estimated FRF and the total (noise + nonlinearities) and noise distortion variance are shown in Figure 2 for two different amplitudes of the input excitation. Based on this figure, the user can observe that a nonlinear model would not offer much improvement in the low input level case. On the other hand, the nonlinear contribution are almost as large as the linear one for the high input level case.

A clear shift in the resonance frequency can be observed. This is a strong indication for a shifting pole, and hence, the presence of a nonlinear feedback in the system. The system dynamics are also clearly visible, and can easily be determined using the linear model order selection techniques, 2nd order dynamics are clearly present.

To conclude, a nonlinear feedback model structure with 2nd order linear dynamics should be a good candidate to model the behavior of the Silverbox system when it is excited by the high amplitude input level. This corresponds with the known underlying structure of the Silverbox system, it is a simple feedback structure. A linear model will be qualitative enough in the low input case.

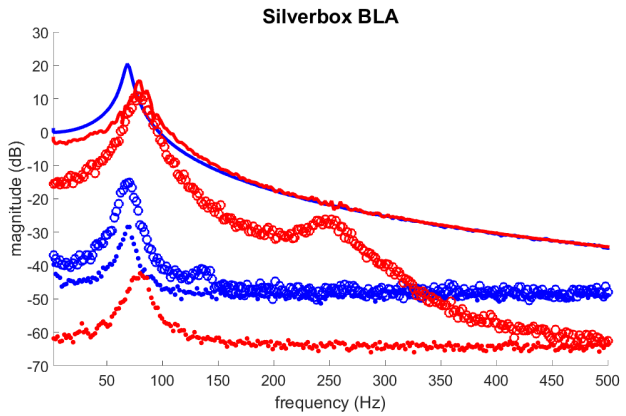


Fig. 2. The BLA of the Silverbox system for low (blue) and high (red) level of excitation. The FRF (full line), total distortion levels (circles) and noise distortion levels (dots) are shown. The size of the gap between the noise and total distortion level indicate how nonlinear the system behaves.

VIII. CONCLUSION

The paper has presented an overview on how the complexity of the (block-oriented) nonlinear modeling process can be reduced significantly using the Best Linear Approximation. The BLA framework offers answers to the questions: "Should I use a nonlinear model for the application at hand?", "What model structure should I select?", and "How can I select the model orders in a simple but efficient way?".

The main disadvantage of using the BLA framework is that it possibly requires more data than some of the other nonlinear modeling approaches which are available. Furthermore, one has to be aware that the BLA can be equal to zero in the presence of nonlinearities which are even around the setpoint of the input signal. This can, of course, easily be solved by changing the constant offset of the input signal used.

ACKNOWLEDGMENT

This work was partly funded by the Belgian Government through the Inter university Poles of Attraction IAP VII/19 DYSCO program, and the ERC advanced grant SNLSID, under contract 320378.

REFERENCES

[1] J. Paduart, "Identification of Nonlinear Systems using Polynomial Nonlinear State Space Models," Ph.D. dissertation, Vrije Universiteit Brussel, Belgium, 2008.

[2] S. Billings, *Nonlinear System Identification: NARMAX Methods in the Time, Frequency, and Spatio-Temporal Domains*. West Sussex, United Kingdom: John Wiley & Sons, Ltd., 2013.

[3] F. Giri and E. Bai, Eds., *Block-oriented Nonlinear System Identification*, ser. Lecture Notes in Control and Information Sciences. Berlin Heidelberg: Springer, 2010, vol. 404.

[4] A. Wills and B. Ninness, "Generalised Hammerstein-Wiener system estimation and a benchmark application," *Control Engineering Practice*, vol. 20, no. 11, pp. 1097–1108, 2012.

[5] M. Schoukens, R. Pintelon, and Y. Rolain, "Identification of Wiener-Hammerstein systems by a nonparametric separation of the best linear approximation," *Automatica*, vol. 50, no. 2, pp. 628–634, 2014.

[6] M. Enqvist and L. Ljung, "Linear approximations of nonlinear FIR systems for separable input processes," *Automatica*, vol. 41, no. 3, pp. 459–473, 2005.

[7] R. Pintelon and J. Schoukens, *System Identification: A Frequency Domain Approach*, 2nd ed. Hoboken, New Jersey: Wiley-IEEE Press, 2012.

[8] J. Bussgang, "Cross-correlation functions of amplitude-distorted Gaussian signals," MIT Laboratory of Electronics, Tech. Rep. 216, 1952.

[9] J. Sjöberg and J. Schoukens, "Initializing Wiener-Hammerstein models based on partitioning of the best linear approximation," *Automatica*, vol. 48, no. 2, pp. 353–359, 2012.

[10] M. Schoukens, A. Marconato, R. Pintelon, G. Vandersteen, and Y. Rolain, "Parametric identification of parallel Wiener-Hammerstein systems," *Automatica*, vol. 51, no. 1, pp. 111 – 122, 2015.

[11] L. Vanbeylen, "Nonlinear LFR Block-Oriented Model: Potential Benefits and Improved, User-Friendly Identification Method," *IEEE Transactions on Instrumentation and Measurement*, vol. 62, no. 12, pp. 3374–3383, 2013.

[12] M. Schoukens and Y. Rolain, "Parametric Identification of Parallel Wiener Systems," *IEEE Transactions on Instrumentation and Measurement*, vol. 61, no. 10, pp. 2825–2832, 2012.

[13] J. Schoukens, R. Pintelon, Y. Rolain, M. Schoukens, K. Tiels, L. Vanbeylen, A. Van Mulders, and G. Vandersteen, "Structure discrimination in block-oriented models using linear approximations: A theoretic framework," *Automatica*, vol. 53, pp. 225–234, 2015.

Distributed Runtime Verification of Cyber-Physical Systems Based on Graph Pattern Matching

Gábor Szilágyi[1], András Vörös[1,2]
[1]Budapest University of Technology and Economics,
Department of Measurement and Information Systems, Budapest, Hungary
[2]MTA-BME Lendület Cyber-Physical Systems Research Group, Budapest, Hungary
Email: szilagyi@db.bme.hu, vori@mit.bme.hu

***Abstract*—Cyber-physical systems process a huge amount of data coming from sensors and other information sources and they often have to provide real-time feedback and reaction. Cyber-physical systems are often critical, which means that their failure can lead to serious injuries or even loss of human lives. Ensuring correctness is an important issue, however traditional design-time verification approaches can not be applied due to the complex interaction with the changing environment, the distributed behavior and the intelligent/autonomous solutions.**

In this paper we present a framework for distributed runtime verification of cyber-physical systems including the solution for executing queries on a distributed model stored on multiple nodes.

I. Introduction

The rapid development of technology leads to the rise of cyber-physical systems (CPS) even in the field of safety critical systems like railway, robot and self-driving car systems. Cyber-physical systems process a huge amount of data coming from sensors and other information sources and it often has to provide real-time feedback and reaction.

Cyber-physical systems are often critical, which means that their failure can lead to serious damages or injuries. Ensuring correctness is an important issue, however traditional design-time verification approaches can not be applied due to the complex interaction with the environment, the distributed behavior and the intelligent controller solutions. These characteristics of CPS result many complex behavior, huge or even infinite number of possible states, so design-time verification is infeasible.

There are plenty of approaches for monitoring requirements [6]. Runtime analysis provides a solution where graph-based specification languages and analysis algorithms are the proper means to analyze the behavior of cyber-physical systems at runtime.

In this paper a distributed runtime verification framework is presented. It is capable of analyzing the correctness of cyber-physical systems and examining the local behavior of the components. An open-source graph query engine being able to store a model in a single machine served as a base of the work [4]. It was extended to support distributed storage and querying: in case of complex specifications, the algorithm collects the information from the various analysis components and infers the state of the system. The introduced framework was evaluated in a research project of the department and proved its usefulness.

Figure 1 shows the basic approach to runtime verification. System development is started by specifying the requirements for the system. Then it is designed, according to the specification. From the specification and the system design, a monitor is created for observing the environment. The monitoring component stores the gathered information in a live model which is updated continuously to represent the actual state of the system. The runtime requirements can be evaluated on the live model and the solution can find if a requirement is violated. Various monitoring approaches exist, some observes data dependent behavior, others can analyze temporal behavior. In this paper the focus is on the runtime analysis of data dependent behavior which can be captured by a graph based representation.

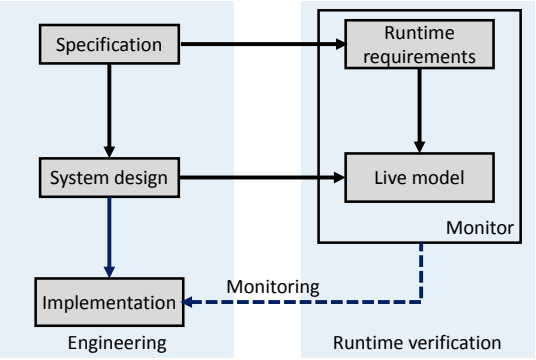


Fig. 1. Model-based runtime verification of a cyber-physical system.

II. Graphs as abstractions

To verify cyber-physical systems, we need to have information about its operation context. Various kinds of information might belong to the context such as the physical environment, computational units, configuration settings or other domain specific information. In modern cyber-physical systems, sensors provide a huge amount of data to be processed by the monitors, it is important to have a comprehensive image of the operation context which can be supported by graph-based knowledge representations.

The current snapshot of the system and its operational context can be formally captured as a *live model* which continuously gets updated to reflect relevant changes in the underlying real system [3]. This live model serves as an abstraction of the analyzed system. The framework uses graph representation to model the actual state of the system. These are directed, typed and attributed graphs. Their vertex types, edge types, and other constraints must be specified in a *metamodel*. The metamodel is also needed for the formalization of specification, since it also specifies the possible structure of the live model.

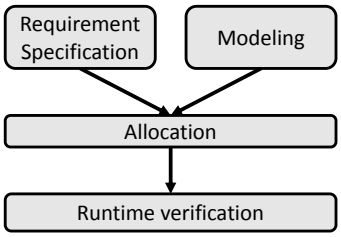


Fig. 2. The presented approach for runtime verification.

The steps of our approach to graph based runtime verification are illustrated on Figure 2. First we need to describe the metamodel which captures the domain information of the monitored system. According to the metamodel and the initial state of the system, a live model is created. This live model is used during the runtime analysis. Then requirements can be defined. After modeling, the system engineer shall specify the allocation i.e. how the elements of the live model are allocated to the computational units of the distributed system. After these tasks, the framework is able to generate the code for runtime verification of the system.

We illustrate this approach with an example of a simplified version of a train control system. First the metamodel shall be created for the system. (Figure 3). In our case, the model is composed of two types of elements: *Segment* and *Train*. Segments next to each other in the physical configuration are connected with *connectedTo* edges in the model. If a train is on a segment, the model represents it with the *onSegment* edge. An example live model of the system can be seen on Figure 4.

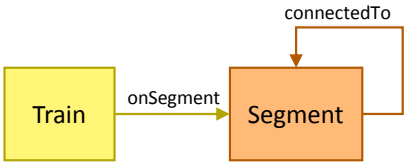


Fig. 3. The metamodel for the system

Safety requirements of the system can be described using graph patterns. A graph pattern is given by

1) a list of variables, each representing a vertex of the live model with a given type
2) a set of constraints, which must be satisfied by the variables, to match the pattern

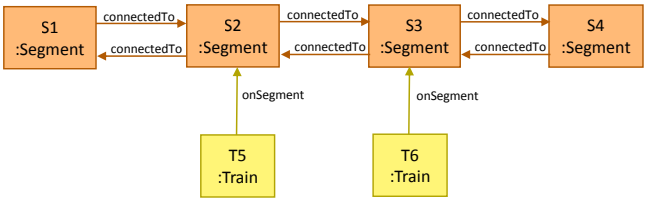


Fig. 4. An example live model for a train control system

Graph patterns in the framework are defined using the VIATRA Query Language (VQL) [1]. It has a rich expressive power capable of expressing constraints like:

- path expression – a specific reference, an attribute, or a path of references must exist between two variables.
- attribute equality – an attribute of a vertex must be a given value
- matching to a pattern – a list of given vertices must match to a pattern
- negative pattern matching – a list of given vertices must not match to another pattern
- check expression - an arbitrary expression containing attributes must be evaluated true

Graph patterns expressed as VQL expressions are evaluated on the input models. Graph pattern matching is reduced to a search for isomorphic subgraphs in the input model. The structure of the graph pattern yields the constraints during the search: the parameters of the graph pattern will finally be assigned to the corresponding graph nodes.

For example, if we want to find trains on adjacent segments, we can use the following pattern (given in VQL):

```
pattern NeighboringTrain(TA, TB)  // 1
{
  Train(TA);  // 2  TA is a train
  Train(TB);  // 3  TB is a train
  Train.currentlyOn(TA, SA);  // 4  TA is currently on SA
  Segment.connectedTo(SA, SB); // 5 SA is connected to SB
  Train.currentlyOn(TB, SB); // 6   TB is currently on SB
}
```

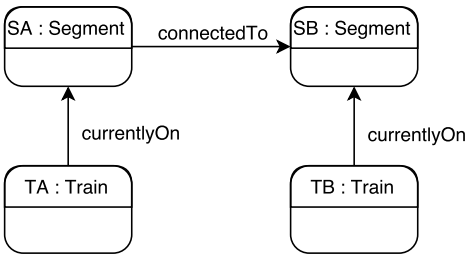


Fig. 5. Graphical visualization of the query.

The pattern's header (1) specifies its name and its parameters. Every statement in the body of the pattern is a constraint (2–6) for variables (SA, SB) and parameters (TA, TB). The visualized version of this pattern can be seen on Figure 5.

In the example model (Figure 4) there are 2 matches of this pattern. One is $\{TA = T5, TB = T6, SA = S2, SB = S3\}$ and the other is $\{TA = T6, TB = T5, SA = S3, SB = S2\}$.

After the requirements are specified, the user has to decompose the model and allocate it into computational units (see Section III). We call this the *allocation* of the live model. The computational units, the live model, and its allocation can be given in JSON format:

```
{
  "nodes" : [
    {
      "name" : "nodeA",
      "ip" : "127.0.0.1",
      "port" : 54321
    }
    ,
    ...
  ],

  "model" : [
    {":id": 0, ":node": "nodeA", ":type": "Segment",
      "connectedTo" :    [1] },
    {":id": 1, ":node": "nodeA", ":type": "Segment",
      "connectedTo" : [0, 2] },
    {":id": 2, ":node": "nodeB", ":type": "Segment",
      "connectedTo" : [1, 3] },
    ...
  ]
}
```

The allocation of a model element can be given by the *":node"* attribute. Model elements, like trains still must be assigned to a specific computational unit, although its physical place can change in time.

After the model elements are allocated to the computational units, and the framework generated the necessary artifacts, runtime verification can be started.

It works in a way depicted on Figure 6. The live model is continuously updated with the runtime information coming from sensors. Runtime requirements of the system – formalized as graph patterns – are verified on the live model continuously, as it is described in the next section, to ensure the system's correct operation.

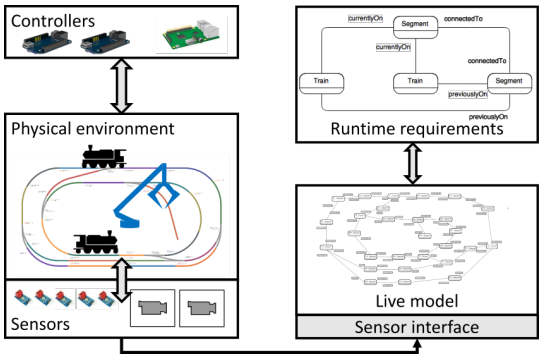


Fig. 6. Runtime verification of the system

III. Distributed Graph Queries

The distributed nature of cyber-physical systems makes runtime verification a challenging task. Various approaches exist regarding the model and query management. The main difference is the way they gather and process the information and evaluate the requirements:

- Centralized model and query management. It would require the sensor information to be transmitted to a central processing machine.
- Distributing the model to each computational unit. It would require model synchronization between nodes.
- Dividing the live model and the query processing tasks to the computational units.

Centralized approaches are not always viable due to various reasons, like the central machine can be easily overloaded, it can be a single point of failure (SPOF), which is undesirable in safety-critical systems. In the second case, model synchronization can introduce unwanted complexity, and overhead in network communication. We solve these problems by processing the sensor information on the corresponding computational units, and updating the local part of a distributed live model.

A. Distributing the storage of the model

After the metamodel is specified, which describes the types of vertices and edges, etc., an initial live model shall be created, representing the initial state of the system. As parts of the model are stored on different computational units, each vertex of the global model must be assigned to a given computational unit. References are stored where the source object for that reference is stored. Basically, the reference can only refer to a local object, i.e. a vertex assigned to the same computational unit. If the reference's destination vertex is not assigned to the same computational unit, we create a proxy object on the same computational unit. Vertices are identified with a globally unique identifier, which is portable between the computational units.

B. Distributed Query Evaluation Algorithm

The algorithm is based on the so-called local search algorithm [2]. To find matches of a given graph pattern, we start from a frame, i.e. a list of variables, unassigned at first. After that, we execute a given list of search operations (called search plan) being specific to the pattern.

To make the algorithm working in distributed systems, we examined the search operations that cannot be executed locally. There are basically two operations, that need to be handled differently from the single machine solution:

- Iterating over the instances of a given vertex type cannot be done locally, since there can be instances for that type on any of the computational units.
- Assigning a variable through a given reference cannot be done, if the source object is not present on the node.

At these operations we inserted a „virtual" search operation. It doesn't operate on the frame, but transmits the query execution to the other computational units of the system. To iterate over instances, first the query execution is distributed between units by the virtual operation, and after that, iterating over local instances can be done. In case of assigning variable via a reference the virtual search operation checks, whether the source object is present on the computational unit, then transmits it to the other units if the source object is not available.

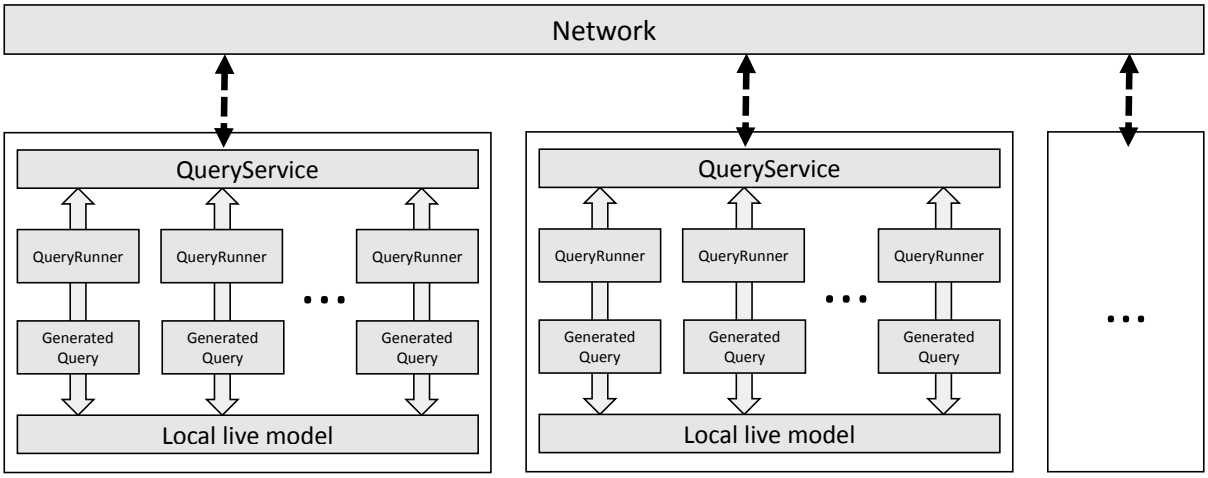


Fig. 7. Architecture of the distributed query evaluation.

C. Architecture

The architectural overview of the distributed query engine is depicted on Figure 7.

On every computational unit of the distributed system, a *QueryRunner* is set up for each generated query. Their role is to execute query tasks specific to their *Generated Query*, on the given local part of the model. An input for a query task consists of 1) a frame, containing the assigned variables, i.e. partial match, and 2) the index of the next search operation to be executed.

If an operation needs distributed execution, the QueryRunner uses the *QueryService* of the computational unit, which handles network communication and reach other computational unit. To serialize the data between different nodes, we used Protocol Buffers [5].

IV. Evaluation

The query evaluation time of the framework was measured in several configuration with the example railway control system, that was presented before, but with a more complex live model, containing 6000 elements. We split the model of the railway system into 2, 3, and 4 parts. First we ran the example query on each configuration, but every computational unit was run on the same machine. So practically, network communication had no overhead during the measurement (Figure 8).

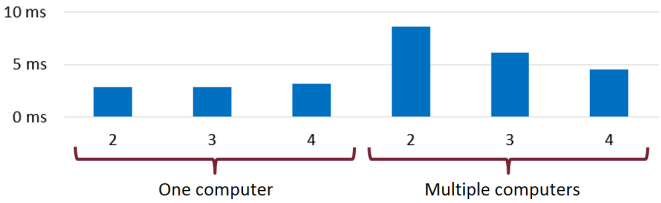


Fig. 8. Average time of query execution by computational units

After that, every computational unit of the system was run on different machines. This shows how network communication affects the speed of our implementation. We can conclude, that networking introduces overhead, but using more computational units makes the system's performance closer to single machine solution. The integration of sensor information in cyber-physical systems cause additional overhead, that can be prevented using the distributed solution.

V. Conclusion

In this paper, we presented a framework for distributed runtime verification of cyber-physical systems based on graph queries. Our approach represents the gathered information in a distributed live model and evaluates the queries as close to the informations sources as possible. A method for distributed model storage and query execution is developed based on a widely used search algorithm. In the future we plan to integrate incremental graph query algorithms to further improve the efficiency of the framework.

References

[1] G. Bergmann, I. Dávid, Á. Hegedüs, Á. Horváth, I. Ráth, Z. Ujhelyi, and D. Varró. VIATRA 3: A reactive model transformation platform. In *Theory and Practice of Model Transformations - 8th International Conference, ICMT 2015, Proceedings*, pages 101–110, 2015.

[2] M. Búr. A general purpose local search-based pattern matching framework. masters thesis. 2015.

[3] I. Dávid, I. Ráth, and D. Varró. Foundations for streaming model transformations by complex event processing. *Software & Systems Modeling*, 2016.

[4] R. Dóczi. Search-based query evaluation over object hierarchies. Master's thesis, Budapest University of Technology and Economics, 2016.

[5] Google. Protocol buffers – data interchange format. https://github.com/google/protobuf.

[6] M. Vierhauser, R. Rabiser, and P. Grünbacher. Requirements monitoring frameworks: A systematic review. *Information & Software Technology*, 80:89–109, 2016.

Timed Automata Verification using Interpolants

Tamás Tóth* and István Majzik†
Budapest University of Technology and Economics,
Department of Measurement and Information Systems,
Fault Tolerant Systems Research Group
Email: *totht@mit.bme.hu, †majzik@mit.bme.hu

***Abstract*—The behavior of safety critical systems is often constrained by real time requirements. To model time dependent behavior and enable formal verification, timed automata are widely used as a formal model of such systems. However, due to real-valued clock variables, the state space of these systems is not finite, thus model checkers for timed automata are usually based on zone abstraction. The coarseness of the abstraction profoundly impacts performance of the analysis. In this paper, we propose a lazy abstraction algorithm based on interpolation for zones to enable efficient reachability checking for timed automata. In order to efficiently handle zone abstraction, we define interpolation in terms of difference bound matrices. We extend the notion of zone interpolants to sequences of transitions of a timed automaton, thus enabling the use of zone interpolants for abstraction refinement.**

I. Introduction

The behavior of safety critical systems is often time dependent. Timed automata [1] are widely used in the formal modeling and verification of such systems. Model checkers for timed automata, e.g. UPPAAL [2], are most commonly based on zone abstraction [3], used in conjunction with an extrapolation operator to ensure termination and accelerate convergence. The quality of the abstraction obtained this way is imperative for performance, and many different techniques have been proposed, including abstractions based on *LU*-bounds [4], [5] and region closure [6]. In [7], a lazy abstraction scheme is proposed for *LU*-abstraction where the abstraction is refined only if it enables a spurious step in the abstract system. This results in an abstraction that is based not only on structural properties of the automaton (like constants appearing in guards) but the state itself.

On the other hand, in SAT/SMT based model checking, interpolation [8] is a commonly used technique [9], [10] for building safe abstractions of systems. As interpolants can be used to extract coarse explanations for the infeasibility of error paths, they can be used for abstraction refinement [11] in a way relevant for the property to be proved. Algorithms for interpolant generation have been proposed for several first order theories, including linear arithmetic ($\mathcal{LA}(\mathbb{Q})$) and difference logic ($\mathcal{DL}(\mathbb{Q})$) over the rationals [12]. To better support certain verification tasks, classical interpolation has been generalized in various ways, e.g. to sequence interpolants [10].

In this paper, we propose a *lazy abstraction* algorithm similar to [7] for reachability checking of timed automata based on interpolation. In order to efficiently handle zone abstraction, we define *interpolation for zones* represented in terms of difference bound matrices. Moreover, we extend the notion of zone interpolants to *sequences of transitions of a timed automaton*, thus enabling the use of zone interpolants for abstraction refinement.

The rest of the paper is organized as follows. In Section II, we define the notations used throughout the paper, and present the theoretical background of our work. In Section III we present our work: we give an interpolation algorithm for zones represented as DBMs (Section III-A), describe how inductive sequences of zones can be computed for timed automata using zone interpolation (Section III-B), and outline how such sequences can be used to speed up convergence for reachability checking of timed automata (Section III-C). Section IV describes the implementation and a preliminary evaluation of the proposed algorithm. Finally, conclusions are given in Section V.

II. Background and Notations

In this section, we define the notations used throughout the paper, and outline the theoretical background of our work.

A. Timed Automata

Timed automata [1] is a widely used formalism for modeling real-time systems. A *timed automaton* (TA) is a tuple $(Loc, Clock, \hookrightarrow, Inv, \ell_0)$ where

- Loc is a finite set of locations,
- $Clock$ is a finite set of clock variables,
- $\hookrightarrow \subseteq Loc \times ClockConstr \times \mathcal{P}(Clock) \times Loc$ is a set of transitions where for $(\ell, g, R, \ell') \in \hookrightarrow$, g is a guard and R is a set containing clocks to be reset,
- $Inv : Loc \to ClockConstr$ is a function that maps to each location an invariant condition over clocks, and
- $\ell_0 \in Loc$ is the initial location.

Here, $ClockConstr$ denotes the set of *clock constraints*, that is, difference logic formulas of the form

$$\varphi \quad ::= \quad \mathbf{true} \mid x_i \prec c \mid x_i - x_j \prec c \mid \varphi \wedge \varphi$$

where $x_i, x_j \in Clock$, $\prec \in \{<, \leq, \doteq\}$ and $c \in \mathbb{Z}$.

The operational semantics of a TA can be defined as a labeled transition system $(S, Act, \to, I)$ where

- $S = Loc \times Eval$ is the set of states,
- $I = \{(\ell_0, \eta_0)\}$ where $\eta_0(x) = 0$ for all $x \in Clock$ is the set of initial states,

*This work was partially supported by Gedeon Richter's Talentum Foundation (Gyömrői út 19-21, 1103 Budapest, Hungary).

- $Act = \mathbb{R}_{\geq 0} \cup \{\alpha\}$, where α denotes discrete transitions,
- and a transition $t \in \rightarrow$ of the transition relation $\rightarrow \subseteq S \times Act \times S$ is either a delay transition that increases all clocks by a value $\delta \geq 0$:

$$\frac{\ell \in Loc \quad \delta \geq 0 \quad \eta' = Delay_\delta(\eta) \quad \eta' \models Inv(\ell)}{(\ell, \eta) \xrightarrow{\delta} (\ell, \eta')}$$

or a discrete transition:

$$\frac{\ell \xrightarrow{g,R} \ell' \quad \eta \models g \quad \eta' = Reset_R(\eta) \quad \eta' \models Inv(\ell')}{(\ell, \eta) \xrightarrow{\alpha} (\ell', \eta')}$$

Here, $Eval$ denotes the set of clock valuations, that is, mappings of clocks to real values. For a real number $\delta \geq 0$ the function $Delay_\delta : Eval \to Eval$ assigns to a valuation $\eta \in Eval$ a valuation $Delay_\delta(\eta)$ such that for all $x \in Clock$

$$Delay_\delta(\eta)(x) = \eta(x) + \delta$$

Moreover, $Reset_R(\eta)$ models the effect of resetting all clocks in R to 0 for a valuation $\eta \in Eval$:

$$Reset_\eta(R)(x) = \begin{cases} 0 & \text{if } x \in R \\ \eta(x) & \text{otherwise} \end{cases}$$

For these functions, we define images and preimages in the usual way.

As the concrete semantics of a timed automaton is infinite due to real valued clock variables, model checkers are often based on a symbolic semantics defined in terms of zones. A *zone* Z is the solution set of a clock constraint φ, that is $Z = [\![\varphi]\!] = \{\eta \mid \eta \models \varphi\}$. The following functions are operations over zones:

- *Conjunction*: $Z \cap g = Z \cap [\![g]\!]$,
- *Future*: $Z^\uparrow = \bigcup_{\delta \geq 0} Delay_\delta(Z)$,
- *Past*: $Z_\downarrow = \bigcup_{\delta \geq 0} Delay_\delta^{-1}(Z)$,
- *Reset*: $R(Z) = Reset_R(Z)$,
- *Inverse reset*: $R^{-1}(Z) = Reset_R^{-1}(Z)$.

For timed automata, an exact pre- and postimage operator can be defined over zones. Let $e = (\ell, g, R, \ell')$ be an edge of a TA with invariants Inv. Then

- $\mathbf{post}(Z, e) = R(Z^\uparrow \cap Inv(\ell) \cap g) \cap Inv(\ell')$,
- $\mathbf{pre}(Z, e) = (R^{-1}(Z \cap Inv(\ell')) \cap g \cap Inv(\ell))_\downarrow$.

B. Difference Bound Matrices

Clock constraints and thus zones can be efficiently represented by difference bound matrices.

A *bound* is either ∞ or of the form $(m, \prec)$ where $m \in \mathbb{Z}$ and $\prec \in \{<, \leq\}$. Difference bounds can be totally ordered by "strength", that is, $(m, \prec) < \infty$, $(m_1, \prec_1) < (m_2, \prec_2)$ iff $m_1 < m_2$ and $(m, <) < (m, \leq)$. Moreover the sum of two bounds is defined as $b + \infty = \infty$, $(m_1, \leq) + (m_2, \leq) = (m_1 + m_2, \leq)$ and $(m_1, <) + (m_2, \prec) = (m_1 + m_2, <)$.

Let $Clock = \{x_1, x_2, \ldots, x_n\}$ and x_0 a reference clock with constant value 0. A *difference bound matrix* (DBM) over $Clock$ is a square matrix D of bounds of order $n + 1$ where an element $D_{ij} = (m, \prec)$ represents the clock constraint $x_i - x_j \prec m$. We say that a clock x_i is constrained in D iff $D_{ii} < (0, \leq)$, or there exists an index $j \neq i$ such that $D_{ij} < \infty$ or $D_{ji} < \infty$.

We denote by $[\![D]\!]$ the zone represented by the constraints in D. When it is unambiguous from the context, we omit the brackets. D is *consistent* iff $D \neq \emptyset$. D is *closed* iff constraints in it can not be strengthened without losing solutions, formally, iff $D_{ij} \leq D_{ik} + D_{kj}$ for all $0 \leq i, j, k \leq n$. We will call D *canonical* iff it is either closed, or $D_{0,0} = (0, <)$. The canonical form of a DBM is unique up to the ordering of clocks. The zone operations above can be efficiently implemented over canonical DBMs [13].

The intersection of DBMs A and B, defined over the same set of clocks with the same ordering, is $A \sqcap B = [\min(A_{ij}, B_{ij})]_{ij}$. Here, $[\![A \sqcap B]\!] = [\![A]\!] \cap [\![B]\!]$, regardless of whether A and B are closed, but $A \sqcap B$ might not be closed even if A and B are. We denote by $A \sqsubseteq B$ iff $[\![A]\!] \subseteq [\![B]\!]$. Moreover, $\top = ((0, \leq))$ and $\bot = ((0, <))$, that is, $\top$ and $\bot$ are the smallest canonical DBMs representing $Eval$ and $\emptyset$, respectively.

III. Interpolation for Timed Automata

A. Binary Interpolants from DBMs

Let A and B two DBMs such that $A \sqcap B$ is inconsistent. An interpolant for the pair (A, B) is a DBM I such that

- $A \sqsubseteq I$,
- $I \sqcap B$ is inconsistent and
- clocks constrained in I are constrained in both A and B.

This definition of a DBM interpolant is analogous to the definition of an interpolant in the usual sense [9]. As DBMs encode formulas in $\mathcal{DL}(\mathbb{Q})$, a theory that admits interpolation [12], an interpolant always exists for a pair of inconsistent DBMs. Our approach for interpolant generation is the direct adaptation of the graph-based algorithm of [12] for DBMs. For the ease of exposition, we assume that A and B are defined over the same set of clocks with the same ordering. This is without the loss of generality, as a DBM can be easily extended with extra columns and rows representing unconstrained clocks without changing its canonicity or the zone it encodes.

The algorithm searches for a negative cycle in $A \sqcap B$ to establish its inconsistency by running a variant of the Floyd-Warshall algorithm [14] on it. In this context, a cycle is negative iff the sum of its constituting bounds is less than $(\leq, 0)$. If no such cycle is found, then A is consistent with B, thus no interpolant exists, and as a side effect the canonical representation of their intersection is obtained. Otherwise, the cycle is reconstructed as a list of indexes and from it an interpolant is extracted in the following way.

Without loss of generality, assume that the cycle is $C = (0, 1, \ldots, l-1)$, and addition for indexes is interpreted modulo l. If for all indexes $0 \leq i < l$ it holds that $A_{i,i+1} \geq B_{i,i+1}$, then B is inconsistent, and the interpolant is $\top$. Dually, if for all indexes $0 \leq i < l$ we have

$A_{i,i+1} \leq B_{i,i+1}$, then A is inconsistent and the interpolant is $\bot$. Otherwise maximal A-paths can be established, each being a maximal list $(i, i+1, \ldots, j)$ of indexes in C such that $A_{i-1,i} \geq B_{i-1,i}$, $A_{i,i+1} < B_{i,i+1}$, $A_{k,k+1} \leq B_{k,k+1}$ for all $i \leq k < j$ and $A_{j,j+1} > B_{j,j+1}$. Notice that clocks x_i and x_j are constrained in both A and in B. The interpolant I is then defined as

$$I_{ij} = \begin{cases} \sum_{k=i}^{j-1} A_{k,k+1} & \text{if } (i, \ldots, j) \text{ is a maximal } A\text{-path} \\ (0, \leq) & \text{if } i = j \\ \infty & \text{otherwise} \end{cases}$$

The algorithm thus yields a canonical DBM in time $\mathcal{O}(n^3)$, even if A and B are not canonical. However, the interpolant obtained for canonical DBMs might be simpler.

B. Sequence interpolants for Timed Automata

In SAT/SMT based model checking, interpolation is used to obtain inductive sequences of states in order to refine abstractions [10]. To do this, the transitions constituting of a given path are transformed to a formula that is satisfiable iff the path is feasible. If the formula is unsatisfiable, a sequence interpolant can be extracted from its refutation produced by the solver.

This same approach can be applied to timed automata, as for a sequence of transitions, such a formula can be constructed in $\mathcal{LA}(\mathbb{Q})$ [15], a theory that admits interpolation. However, due to delay transitions, the formula will not be in $\mathcal{DL}(\mathbb{Q})$, thus the interpolant is not guaranteed to be a sequence of zones.

So in order to make interpolation applicable for the zone-based model checking of timed automata, we generalize binary interpolation for DBMs to sequences of transitions of a timed automaton, analogously to the usual definition of sequence interpolation.

Let $(e_1, e_2, \ldots e_n)$ be a sequence of edges of a timed automaton that induce an infeasible path. Then there exists a sequence of zones, represented by DBMs $(I_0, I_1, \ldots, I_n)$ such that

- $I_0 = \top$ and $I_n = \bot$,
- $\mathbf{post}(I_i, e_{i+1}) \sqsubseteq I_{i+1}$ for all $0 \leq i < n$ and
- for all $0 < i < n$, the clocks constrained in I_i are constrained both in $I_0, \ldots, I_i$ and $I_{i+1}, \ldots, I_n$.

Such an interpolant can be calculated by using the image operators $\mathbf{post}$ and $\mathbf{pre}$. Let $B_n = \top$ and $B_{i-i} = \mathbf{pre}(B_i, e_i)$ for all $0 \leq i < n$. Moreover, let $A_0 = \top$, and $A_{i+1} = \mathbf{post}(I_i, e_{1+1})$ for all $0 \leq i < n$. Then we can define I_i for all $0 \leq i \leq n$ as the interpolant for the pair (A_i, B_i). This calculation is analogous to the usual computation of a sequence interpolant in terms of binary interpolation, and it can be easily shown that $(I_0, I_1, \ldots, I_n)$ is indeed a sequence interpolant.

C. Lazy Abstraction for Timed Automata using Interpolants

For our purposes, an *abstract reachability tree* (ART) is a rooted directed tree of nodes labeled by pairs of the form (ℓ, E), where E is an abstract state representing a zone $[\![E]\!]$. A node where $[\![E]\!] = \emptyset$ is called infeasible. A node might be marked covered by an other node, called the covering node. A node is excluded iff it is either covered, infeasible or has an excluded parent. An ART is ℓ-safe iff all nodes labeled by location ℓ are excluded. An ART is said to be well-labeled iff it satisfies the following properties:

1) the root of the tree is labeled (ℓ_0, E_0) for E_0 such that $[\![E_0]\!] = \{\eta_0\}$,
2) for each non-excluded node labeled (ℓ, E) and transition $e = (\ell, g, R, \ell')$, there is a successor node labeled (ℓ', E') such that $\mathbf{post}([\![E]\!], e) \subseteq [\![E]\!]$, and
3) each covered node labeled (ℓ, E) is covered by a non-excluded node (ℓ, E') such that $[\![E]\!] \subseteq [\![E']\!]$.

A well-labeled, ℓ-safe ART for a timed automaton is a proof that ℓ is unreachable. The goal of model checking is to find such an ART, or show a counterexample.

For this end, one extreme would be to expand the tree based on a precise post-image operator $\mathbf{post}$ and never apply refinement. An other approach, known as IMPACT [10] in software model checking, would expand the tree with abstraction $\top$ for each new node, and apply interpolation-based refinement to maintain well-labeledness and safety in the following cases:

- A non-excluded node labeled with ℓ_{err} is encountered. To exclude the error node, an interpolant is calculated for the path from root to the node. If such an interpolant exists, it is used to strengthen the current abstraction, otherwise a counterexample is given.
- To enforce coverage. To enforce $[\![E]\!] \subseteq [\![E']\!]$ and enable coverage between non-excluded nodes (ℓ, E) and (ℓ, E'), an interpolant is calculated for the path from the least common ancestor of the two nodes and the node to cover. If such an interpolant exists, it is used to strengthen the abstraction, and thus enable coverage.

Our approach can be seen as the combination of the two algorithms above. It is the adaptation of the strategy proposed in [7] for interpolation-based refinement. In this framework, an abstract state is essentially a pair of zones (Z, W). Here, Z tracks the precise reachability information, and W an abstraction of it, thus we define $[\![(Z, W)]\!]$ as $[\![W]\!]$. The root of the tree is labeled (Z_0, Z_0). When expanding a node labeled (ℓ, Z, W) for a transition $e = (\ell, g, R, \ell')$, the successor obtains label (ℓ', Z', W') such that $Z' = \mathbf{post}(Z, e)$ and $W' = \top$. Once a node labeled (ℓ_{err}, Z, W) is encountered where $Z \neq \bot$, we know that ℓ_{err} is reachable, and the counterexample can be reported.

We apply interpolation in the following two cases to maintain well-labeledness:

- A non-excluded node labeled $(\ell_{err}, \bot, W)$ is encountered. In this case, the only reasonable thing to do is to exclude the node. Suppose the node can not be excluded by covering any of its ancestors. We then strengthen the abstraction by a zone interpolant computed for the path from the root to the node, thus excluding the node.

- To enforce coverage. Given a pair of non-excluded nodes labeled (ℓ, Z, W) and (ℓ, Z', W'), respectively, such that $Z \subseteq W'$ but $W \not\subseteq W'$, we can strengthen the abstraction to obtain $W \subseteq W'$. This however requires the complementation of zone W', which may yield more than one zones. We use each of those zones as a target for computing an interpolant from the root to the node. By strengthening the abstraction with each of those interpolants, we effectively enforce coverage.

Note that by strengthening the abstraction for a given path, some previously covered nodes might become uncovered. For these nodes, we can try to enforce coverage, which might be a costly operation. However, forced coverage can be applied incrementally: when the abstraction W of a covering node is strengthened by an interpolant I, it is sufficient to use I as a target for interpolation, which is possibly simpler than the new abstraction. Moreover, a heuristic can be applied that restricts forced coverage for cases where the target is represented by a small DBM.

IV. Implementation

We implemented the DBM-based interpolation approach described in Section III-A. Moreover, as a proof of concept, we implemented the algorithm proposed in Section III-C as a prototype in the formal analysis framework THETA. Our current implementation does not support forced coverage.

Nevertheless, the algorithm shows promising results in terms of both speed and the size of the ART it constructs. We compared our algorithm with basic forward search (without extrapolation), and two versions of IMPACT without forced covering implemented for timed automata, one for predicate abstraction (P), and one for zone abstraction (Z). We used a small model, the model of Fischer's protocol for four processes. Besides execution time of the analysis and the number of nodes in the resulting ART, we calculated the number of nodes that are either non-excluded or leaf, as this is the number of meaningful nodes.

	e. time (s)	#n	#n minimized
our algorithm	7	4946	985
basic search	10	9857	9857
IMPACT(Z)	TO	-	-
IMPACT(P)	TO	-	-

From the results in the table above, it can be seen that IMPACT without forced covering reaches timeout (set to one minute) regardless of the abstract domain used, as the algorithm spends a significant amount of time searching in unreachable segments of the state space. On the other hand, our algorithm generates an ART half as big as the basic algorithm, even without forced covering. As the number of nodes in the reduced ART suggests, with a better search and covering strategy, the number of nodes in the ART can be significantly decreased.

V. Conclusions and Future Work

In this paper, we proposed a lazy abstraction algorithm for reachability checking of timed automata based on interpolation for zones. In order to efficiently handle zone abstraction, we treat interpolant generation as an operation over difference bound matrices. Furthermore, we extended the notion of zone interpolation to sequences of transitions of a timed automaton. By using interpolants for abstraction, the proposed algorithm has the potential to significantly speed up the convergence of reachability checking.

We are currently extending our implementation with forced covering. In the future, we plan to investigate how interpolation can be efficiently used in conjunction with known abstractions for zones to obtain coarse abstractions for timed automata. Moreover, we intend to search for more efficient interpolant generation algorithms for DBMs. Furthermore, we are going to thoroughly evaluate an optimized version of our algorithm on usual benchmarks for timed automata, and compare it to the lazy abstraction algorithm based on *LU*-bounds [7].

References

[1] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994.

[2] G. Behrmann, A. David, K. G. Larsen, J. Håkansson, P. Petterson, Y. Wang, and M. Hendriks, "UPPAAL 4.0," in *Third International Conference on the Quantitative Evaluation of Systems*. IEEE, 2006, pp. 125–126.

[3] C. Daws and S. Tripakis, *Model checking of real-time reachability properties using abstractions*. Springer Berlin Heidelberg, 1998, vol. 1384 LNCS, pp. 313–329.

[4] G. Behrmann, P. Bouyer, K. G. Larsen, and R. Pelánek, *Lower and Upper Bounds in Zone Based Abstractions of Timed Automata*. Springer Berlin Heidelberg, 2004, vol. 2988 LNCS, pp. 312–326.

[5] F. Herbreteau, B. Srivathsan, and I. Walukiewicz, "Better abstractions for timed automata," in *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science*, ser. LICS. IEEE, 2012, pp. 375–384.

[6] F. Herbreteau, D. Kini, B. Srivathsan, and I. Walukiewicz, "Using non-convex approximations for efficient analysis of timed automata," in *FSTTCS 2011*, ser. LIPIcs, vol. 13, 2011, pp. 78–89.

[7] F. Herbreteau, B. Srivathsan, and I. Walukiewicz, "Lazy abstractions for timed automata," in *Computer Aided Verification*, vol. 8044 LNCS. Springer International Publishing, 2013, pp. 990–1005.

[8] W. Craig, "Three uses of the herbrand-gentzen theorem in relating model theory and proof theory," *The Journal of Symbolic Logic*, vol. 22, no. 3, pp. 269–285, 1957.

[9] K. L. McMillan, "Interpolation and sat-based model checking," in *Computer Aided Verification*, vol. 2725 LNCS. Springer Berlin Heidelberg, 2003, pp. 1–13.

[10] ——, "Lazy abstraction with interpolants," in *Computer Aided Verification*, vol. 4144 LNCS. Springer Berlin Heidelberg, 2006, pp. 123–136.

[11] 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.

[12] A. Cimatti, A. Griggio, and R. Sebastiani, "Efficient interpolant generation in satisfiability modulo theories," in *Tools and Algorithms for the Construction and Analysis of Systems*, vol. 4963 LNCS. Springer Berlin Heidelberg, 2008, pp. 397–412.

[13] J. Bengtsson and W. Yi, *Timed Automata: Semantics, Algorithms and Tools*. Springer Berlin Heidelberg, 2004, vol. 3098 LNCS, pp. 87–124.

[14] R. W. Floyd, "Algorithm 97: Shortest path," *Communications of the ACM*, vol. 5, no. 6, pp. 345–, 1962.

[15] R. Kindermann, T. Junttila, and I. Niemelä, "Beyond Lassos: Complete SMT-Based Bounded Model Checking for Timed Automata," in *Formal Techniques for Distributed Systems*, vol. 7273 LNCS. Springer Berlin Heidelberg, 2012, pp. 84–100.

Executing Online Anomaly Detection in Complex Dynamic Systems

Tommaso Zoppi
Department of Mathematics and Informatics, University of Florence
Viale Morgagni 65, Florence, Italy
tommaso.zoppiunifi.it

***Abstract*—Revealing anomalies in data usually suggests significant - also critical - actionable information in a wide variety of application domains. Anomaly detection can support dependability monitoring when traditional detection mechanisms e.g., based on event logs, probes and heartbeats, are considered inadequate or not applicable. On the other hand, checking the behavior of complex and dynamic system it is not trivial, since the notion of "normal" – and, consequently, anomalous - behavior is changing frequently according to the characteristics of such system. In such a context, performing anomaly detection calls for dedicate strategies and techniques that are not consolidated in the state-of-the-art. The paper expands the context, the challenges and the work done so far in association with our current research direction. The aim is to highlight the challenges and the future works that the PhD student tackled and will tackle in the next years.**

Keywords—anomaly detection; monitoring; multi-layer; dynamicity; complex system; online

I. Introduction

Using anomaly detectors to assess the behavior of a target complex system at runtime is a promising approach that was explored in the last decade [1], [2]. Previous works showed that anomaly detection is a very flexible technique, analyzing different monitored behavioral data flows, finally allowing correlation between different events. This technique is commonly used to build error detectors [5], intrusion detectors [4] or failure predictors [3], assuming that a manifestation of an error or an adversarial attacker activity leads to an increasingly unstable performance-related behavior before escalating into a (catastrophic) failure. Anomaly detectors are in charge of i) detecting these fluctuations, and ii) alerting the administrator – who triggers proactive recovery or dumps critical data - with a sufficient look-ahead window. As stated in [1], anomaly detection strictly depends on the ability of distinguishing among normal and anomalous behavior. Unfortunately, complex and dynamic systems can often hide behavioral details (e.g., *Off-The-Shelf* components) or call for frequent reconfigurations, negatively affecting our ability in performing anomaly detection.

In particular, enterprise solutions such as *Nagios*, *Ganglia* or *Zenoss* allow the administrator to choose which indicators (e.g., CPU usage, accesses to hard disk) to observe, tracing their evolution through time. These enterprise tools also give the chance to setup static thresholds for each indicator, testing the availability of each single functionality or service exposed by the target system. Nevertheless, as expanded in Section III, they i) do not implement dynamic thresholds (e.g., statistical) for the monitored indicator, and ii) cannot easily adapt their behaviour when the configuration of the target system changes, calling for manual reconfigurations. This represents a strong limitation for the usage of such tools in dynamic systems.

The paper is structured as follows: Section II reports on anomaly detection, while Section III points out the motivations of our work and the related challenges. Section IV describes the framework for anomaly detection that is currently under investigation, while Section V and Section VI conclude the paper focusing on the ongoing and planned future works.

II. Anomaly Detection in Complex Dynamic Systems

Dynamicity is the property of an entity that constantly changes in term of offered services, built-in structure and interactions with other entities. From one side, this defines systems that can adapt their behavior according to the actual needs, but on the other side it makes all the behavioral checks harder to execute since the normal behavior is changing very often. Regarding dependability assessment, this means that failure predictors must be kept up-to-date as the system is running, calling for a new training phase aimed at reconfiguring all the involved parameters. This calls for a monitoring solution that i) continuously observes the system to avoid or mitigate failures of attack, ii) gathers data from modules or layers where possible, and iii) is able to infer the status of the whole system looking *only* at data collected at its constituent modules. It follows that detection algorithms based on fingerprints e.g., antiviruses [14], intrusion detectors [13] or failure predictors [9], may result not adequate for complex systems due to their intrinsic dynamicity.

In such a context, anomaly detection seems one of the most suitable approaches in detecting unexpected behaviors in dynamic and complex systems. In the security domain, this technique was proven effective [14] in detecting zero-day attacks, which exploit unknown vulnerabilities to get into the targeted system. Antiviruses and intrusion detectors can detect hazards when they identify a behavior that is compliant with a known fingerprint of an attacker or a malware, but they need also rules to detect zero-day attacks or attacks from unknown adversaries [12]. The same approach is commonly used to detect threats to dependability in complex systems, also when the system is composed by OTS components [10], [7]. Moreover, unexpected or previously unknown system failures can be predicted observing specific indicators to characterize if the runtime system behavior is compliant with given performance expectations [9], [10]. Several frameworks targeting anomaly

detection in a specific subset of complex systems, namely Systems-of-Systems (SoSs) are summarized in [20]. More in detail, some of them deal with dynamicity [11], while others tackle systems composed of OTS components [9], [10]. All the considered frameworks are realized either for dependability [9], [10], [11] or security [4], [12] purposes.

III. Main Challenges

To the authors' knowledge, the topic of bringing anomaly detection into the design of complex dynamic systems e.g., Service Oriented Architectures (SOAs) [8] or SoSs [20], was not explored in the recent years. Consequently, after expanding the topic of *anomaly detection*, in the rest of the paper we will report on both the motivations of the research and the challenges that the 3rd-year PhD student tackled in the first two years, with a closer look to the next planned research steps.

Summarizing, the tackled research challenges are:

CH1. *Design a monitoring and anomaly detection framework that is suitable for dynamic and/or distributed systems* and therefore is tailored to automatically adapt its parameters depending on the current configuration of the target system;

CH2. Provide a *flexible monitoring* strategy that copes with dynamicity of complex systems. The strategy must allow the collection of data coming from different system layers and constituent machines that can be updated frequently;

CH3. Understand the expected behavior of the system according to its current context. The *context must be detected at runtime*, calling for specific training phases that should not interfere with the normal usage of the platform (e.g., availability to the users)

CH4. Analyze monitored data to *extract the minimum set of features* (i.e., anomaly checkers and, consequently, monitored indicators) which guarantees the best tradeoff between monitoring effort and efficiency of the anomaly detection process at runtime;

IV. Building a Framework for Multi-Layer Anomaly Detection in Complex Dynamic Systems

A. Designing the framework

In [7] the authors applied the *Statistical Predictor and Safety Margin* (SPS) algorithm to detect the activation of software faults in an *Air Traffic Management* (ATM) system, that has few defined functionalities with respect to a SOA. Observing only *Operating System* (OS) indicators, SPS allowed performing error detection with high precision. This is a promising approach since it i) relies on a multi-layer monitoring strategy that allows to infer the state of the applications looking only at the underlying system layers (see CH2), and ii) uses a sliding-window-based machine learning algorithm that automatically tunes its parameters as the data flows (see CH1). Therefore we adapted this approach to work in a more dynamic context [5] where we instantiated the framework on one of the 4 virtual machines running the prototype of the Secure! [6] SOA.

The results achieved showed that analysing such a dynamic system without adequate knowledge on its behavior reduces the efficiency of the whole solution. We explain these outcomes as follows. SPS detects changes in a stream of observations identifying variations with respect to a predicted trend: when an observation does not comply with the predicted trend, an alert is raised. If the system has high dynamicity due to frequent changes or updates of the system components, or due to variations of user behaviour or workload, such trend may be difficult to identify and thus predict. Consequently, our ability in identifying anomalies is affected because boundaries between normal and anomalous behaviour cannot be defined properly.

Consequently, we investigated which information on SOA services we can obtain in absence of details on the services internals and without requiring user context (i.e., user profile, user location). In SOAs, the different services share common information through an *Enterprise Service Bus* (*ESB*, [8]) that is in charge of i) integrating and standardizing common functionalities, and ii) collecting data about the services. This means that static (e.g., services description available in *Service Level Agreements* - SLAs) or runtime (e.g., the time instant a service is requested or replies, or the expected resources usage) information about the context can be retrieved using knowledge given by ESB. In particular, having access to the ESB provides knowledge on the set of generic services running at any time *t*. We refer to this information as *context-awareness* of the considered SOA.

We can exploit this information to define more precisely the boundaries between normal and anomalous behaviour of the system under observation. For example, consider a user that invokes a *store file* service at time *t*. We can combine context-awareness with information on the usual behaviour of the service, which here regards data transfer. Therefore, if the *store file* service is invoked at time *t*, we expect the exchange of data during almost the entire execution of the service. Contrary, we can reveal that something anomalous is happening. This also highlights that the observation of lower levels make us able to identify the manifestation of errors at service level, ultimately providing both i) monitoring flexibility, and ii) maximum detection capability.

B. High-Level Architecture

In Figure 1 we depicted a high-level view of the framework. Starting from the upper left part of the figure, the framework can be described as follows. The user executes a *workload*, which is a sequence of invocations of SOA services hosted on the *Target Machine*. In this machine, *probes* are running, observing the indicators coming from 3 different system layers: i) *OS*, ii) *middleware* and iii) *network*. These probes collect data, providing a *snapshot* of the target system composed by the observation of indicators retrieved at a defined time instant. The probes forward the snapshot to the *communication handler*, which encapsulates and sends the snapshot to the other *communication handler*. Data is analyzed on a separate machine, the *Detector Machine*. This allows i) not being intrusive on the Target Machine, and ii) connecting more Target Machines to the same Detector Machine (note that the number of Target Machines is limited by the computational resources of the Detector Machine). The communication handler of the Detector Machine collects and sends these data to the *monitor aggregator*, which merges them with *runtime information* on the

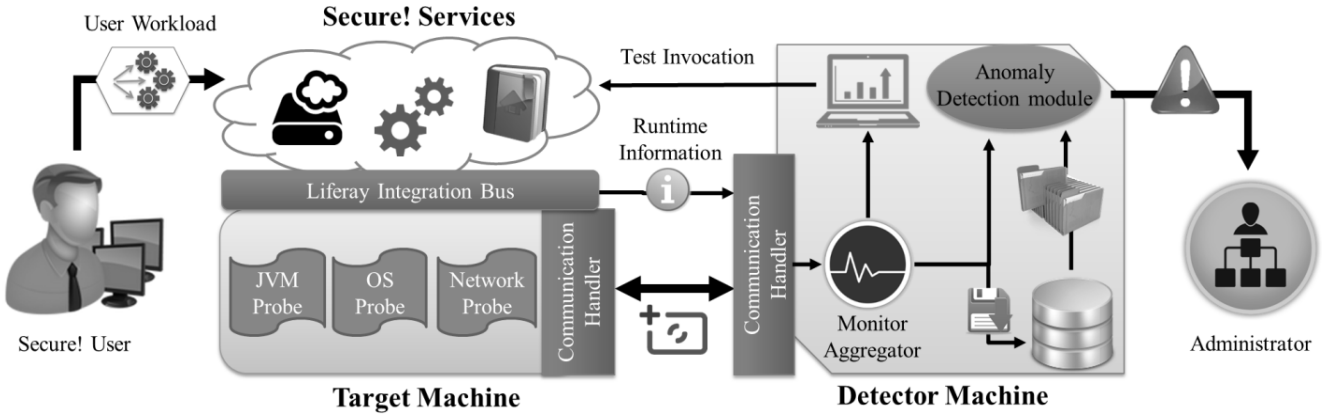


Fig 1. High-level view of the multi-layer anomaly detection framework

context (e.g., the sequence of service calls) obtained from the ESB. Looking at *runtime information*, the monitor aggregator can detect changes in the SOA and notify the administrator that up-to-date services information is needed to appropriately tune the anomaly detector. The administrator is in charge of running tests (*test invocation*) to gather novel information on such services.

The snapshots collected when the SOA is opened to users are sent to the *anomaly detection module*, which can query the database for contextual services information. The *anomaly detection module* analyzes each observed snapshot to detect anomalies. If an anomaly is detected, an alert is raised to the *system administrator* that takes countermeasures and applies reaction strategies. These are outside from the scope of this work and will not be elaborated further.

C. Exercising the framework

The framework is instantiated specifying: i) a *workload*, ii) the approach to obtain contextual data (as mentioned in Section 4.A), iii) the set of monitored indicators, and iv) the amount of data needed for training. The methodology is composed of two phases: *Training Phase* and *Runtime Execution*.

Training Phase. First, the approach (static or runtime) to obtain *contextual data* characterizing the fingerprint of the investigated services is instantiated. Then, training data is collected during the execution of the chosen workload, storing in a database a time series for each monitored indicator representing the evolution of its value during the train experiment. These data are complemented with data collected conducting anomaly injection campaigns, where anomalies are injected in one of the SOA services, to witness the behavior of the target system in such situations. These data are lastly used by the *anomaly detection module* to tune its parameters depending on the current context. Injected anomalies simulate the effect of the manifestation of an error or of an upcoming failure, e.g., anomalous resource usage.

Runtime Execution. Once the training phase is ended and the parameters of the anomaly detector are defined, the system is opened to users. *Monitor aggregator* merges each snapshot observed by the *probing system* with *runtime information*, and it sends them to the *anomaly detection module*. This module provides a numeric anomaly score: if the score reaches a specified threshold, an anomaly alert is risen and the administrator is notified. If during this phase a change in the system is detected, a new *training phase* is scheduled and it will be executed depending on the policies defined by the administrator (e.g., during lunchtime, instantly, at night).

V. Dealing with Online Training

According to the description in Section 4, we can observe how the availability of the anomaly detector is affected from the time it needs to train its algorithms and to detect the contextual data (see CH3). All the detection systems that depend on training data have to deal with this turnover between training phase – in which the system is tuning the detector – and runtime execution – where the system is opened to users and executing its tasks with anomaly detection support.

A. Limitations of Training Phases

The time requested by the training phase is considered not influent in systems that i) can be put offline in defined time periods (e.g., servers that are unused at night), or ii) have static behaviors (e.g., air traffic management systems), meaning that the training phase can be executed once keeping their results valid through time. Nevertheless, a big subset of systems do not adhere with these specifications since they have a dynamic behavior that calls for frequent training phases needed to adapt the parameters of the anomaly detector to the current context. In such a context, frequent training phases are needed to keep the anomaly detection logic compliant with the current notion of "normal" and "anomalous" behavior. Unfortunately, during these training phases the anomaly detector is working with outdated parameters negatively affecting the correct detection of anomalies. To limit these adverse effects, several authors [16], [17], [18], [19] dealing with detector or predictors in the context of complex systems proposed an "online training" approach.

B. Online Training

A strong support for the design of online training techniques comes from systems that study trajectories [15], [16]. In this field, abnormal trajectories tend to carry critical information of potential problems that require immediate attention and need to be resolved at an early stage [15]. The trajectories are continuously monitored as they evolve to understand if they are following normal or anomalous paths, ultimately breaking the classic training-validation turnover (see *conformal anomaly*

detection [16]). In [17], authors tackle online training for failure prediction purposes i) continuously increasing the training set during the system operation, and ii) dynamically modifying the rules of failure patterns by tracing prediction accuracy at runtime. A similar approach is adopted also to model up-to-date *Finite State Automata* tailored on sequences of system calls for anomaly-based intrusion detection purposes [18] or *Hidden Semi Markov Models* targeting online failure prediction [19].

C. Shaping Online Training for Dynamic Systems

When the target system is dynamic, it can change its behavior in different ways, consequently affecting the notion of normal or expected behavior. These changes must trigger new training phases aimed at defining the "new" normal behavior, allowing the parameters of the anomaly detector to be tailored on the current version of the system. Moreover, according to [17], the training set is continuously enriched by the data collected during the executions of services, providing wide and updated datasets that can be used for training purposes. This training phase starts once one of the triggers will activate, calling for complex data analytics that are executed on a dedicated machine, to do not bother the target system with these heavy computations.

Looking at the possible ways systems have to dynamically change their behavior, we are currently considering as triggers: i) *update of the workload,* ii) *addition or update of a service* in the platform, iii) *hardware update*, and iv) *degradation of the detection scores*. While the first three triggers can be detected easily either looking at the basic setup of the SOA or after a notification of the administrator, the degradation of detection scores needs more attention. Concisely, the dynamicity of the system is not only due to events that can alter its behavior (i.e., the first three triggers). The notion of normal behavior may be affected also by changes of the environment or of some internals of the systems than cannot be easily identified. Unfortunately, they might lead to a performance degradation of the anomaly detector (e.g., higher number of false positives) ultimately calling for an additional training phase.

VI. Conclusions and Future Works

This paper presents the topic the student is tackling during his PhD. More in detail, after describing the research area and the state of the art that was consolidated in the recent years, the paper addressed the motivations and the related challenges. Then, we described a framework for multi-layer anomaly detection in complex and dynamic systems that we developed during the PhD research period.

Future works will be oriented to deal with the dynamicity of complex systems. While a strategy for the collection and the analysis of data is already implemented in the framework mentioned before, some improvements need to be considered in order to make this framework suitable for dynamic systems. In particular, we will go through the online training approach, looking for strategies that will allow having always a clear definition of normal behavior. As discussed in Section 5.C, this will require deeper investigations on all the possible ways systems have to dynamically change their behavior. Moreover, we are planning to tackle the feature selection problem (see CH4) after the dynamic characteristics will be clearly defined.

Acknowledgment

This work has been partially supported by the IRENE JPI Urban Europe, the EU-FP7-ICT-2013-10-610535 AMADEOS and the EU-FP7-IRSES DEVASSES projects.

References

[1] Chandola, Varun, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey." ACM computing surveys (CSUR) 41.3 (2009): 15.

[2] Rajasegarar, Sutharshan, Christopher Leckie, and Marimuthu Palaniswami. "Anomaly detection in wireless sensor networks." IEEE Wireless Communications 15.4 (2008): 34-40.

[3] Özçelik, Burcu, and Cemal Yılmaz. "Seer: a lightweight online failure prediction approach." IEEE Transactions on Software Engineering (2013).

[4] Salama, Shaimaa Ezzat, et al. "Web anomaly misuse intrusion detection framework for SQL injection detection." Editorial Preface 3.3 (2012).

[5] Ceccarelli, Andrea, et al. "A multi-layer anomaly detector for dynamic service-based systems." Int. Conference on Computer Safety, Reliability and Security (pp. 166-180). Springer Int. Publishing, SAFECOMP 2015.

[6] Secure! project, http://secure.eng.it/ (last accessed 1st December 2016)

[7] Bovenzi, Antonio, et al. "An OS-level Framework for Anomaly Detection in Complex Software Systems." Dependable and Secure Computing, IEEE Transactions on 12.3 (2015): 366-372.

[8] Erl, Thomas. Soa: principles of service design. Vol. 1. Upper Saddle River: Prentice Hall, 2008.

[9] Baldoni, Roberto, Luca Montanari, and Marco Rizzuto. "On-line failure prediction in safety-critical systems." Future Generation Computer Systems 45 (2015): 123-132.

[10] Williams, Andrew W., Soila M. Pertet, and Priya Narasimhan. "Tiresias: Black-box failure prediction in distributed systems." Parallel and Distributed Processing Symposium, IEEE 2007 (pp. 1-8). IPDPS 2007.

[11] Zoppi, Tommaso, Andrea Ceccarelli, and Andrea Bondavalli. "Context-Awareness to Improve Anomaly Detection in Dynamic Service Oriented Architectures." International Conference on Computer Safety, Reliability, and Security (pp 145-158). Springer International Publishing, 2016.

[12] Perdisci, Roberto, et al. "McPAD: A multiple classifier system for accurate payload-based anomaly detection." Computer Networks 53.6 (2009): 864-881.

[13] Mukkamala, Srinivas, Guadalupe Janoski, and Andrew Sung. "Intrusion detection using neural networks and support vector machines." Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on. Vol. 2. IEEE, 2002.

[14] Comar, Prakash Mandayam, et al. "Combining supervised and unsupervised learning for zero-day malware detection." INFOCOM, 2013 Proceedings IEEE (pp. 2022-2030). IEEE, 2013.

[15] Bu, Y., Chen, L., Fu, A. W. C., & Liu, D. (2009, June). Efficient anomaly monitoring over moving object trajectory streams. In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 159-168). ACM.

[16] Laxhammar, Rikard, and Göran Falkman. "Online learning and sequential anomaly detection in trajectories." IEEE transactions on pattern analysis and machine intelligence 36.6 (2014): 1158-1173.

[17] Gu, J., Zheng, Z., Lan, Z., White, J., Hocks, E., & Park, B. H. (2008, September). Dynamic meta-learning for failure prediction in large-scale systems: A case study. In 2008 37th International Conference on Parallel Processing (pp. 157-164). IEEE.

[18] Sekar, R., Bendre, M., Dhurjati, D., & Bollineni, P. (2001). A fast automaton-based method for detecting anomalous program behaviors. In Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on (pp. 144-155). IEEE.

[19] Salfner, F., & Malek, M. (2007, October). Using hidden semi-Markov models for effective online failure prediction. In Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE Int. Symp. on (pp. 161-174). IEEE.

[20] Zoppi, Tommaso, Andrea Ceccarelli, and Andrea Bondavalli. " Exploring Anomaly Detection in Systems of Systems." To Appear at Symposium on Applied Computing (SAC) 2017, Marrakesh, Morocco.