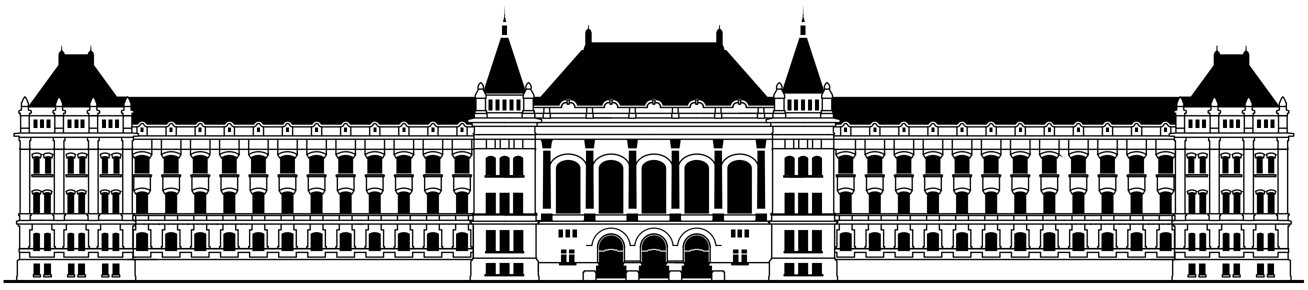
Proceedings of the 27th Minisymposium

of the

Department of Measurement and Information Systems
Budapest University of Technology and Economics

(Minisy@DMIS 2020)

February 5–6, 2020
Budapest University of Technology and Economics
Building I



Budapest University of Technology and Economics
Department of Measurement and Information Systems

Head of the Department:
Tamás Dabóczi

General Chair:
Balázs Renczes

Organizers:
Bence Bruncsics
Kristóf Marussy
Péter Nagy

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

Sponsored by:
Schnell László Foundation
E-Group

Supported by the EFOP-3.6.2-16-2017-00013 grant of the European Union,
co-financed by the European Social Fund.

Foreword

On behalf of the Organizing Committee, I welcome you to the 27th Minisymposium of the Department of Measurement and Information Systems at the Budapest University of Technology and Economics.

As it used to be from the beginning of the history of the Minisymposium, the main idea of this event is to provide the Ph.D. students of the department the opportunity to present and discuss their scientific results. Furthermore, the students can also gain some insights into the practical steps of organizing a scientific event. We are happy to see that in the last few years, the Minisymposium expanded to give space for our international industrial and scientific partners, as well as to our talented Bachelor and Master students.

During the years, besides the circle of lecturers, the scope of covered topics has broadened, as well; though our department still have research topics in the field of measurement and instrumentation, the investigated areas have been gradually growing to cover embedded information systems, dependability and security, artificial intelligence, bioinformatics, and cyber-physical systems.

Similarly to the last years' practice, and also conforming to the international trends, the proceedings will only be published in electronic form. We have experienced that the easy accessibility of the digital proceedings makes it insufficient to publish a printed edition. We hope that any emerging inconvenience will be dominated by the advantages of the electronic form.

We wish that the forthcoming one and a half days will not only be fruitful in a sense that we will be able to gain insight into the research of one another, but it will also be a time for discussions and collecting ideas for future research, as well as for finding possible interdisciplinary cooperation areas.

Budapest, February, 2020

Renczes Balázs

Balázs Renczes
General Chair

Papers of the Minisymposium

Comparison of LMS-based Adaptive Audio Filters

Kristóf Horváth, Balázs Bank
Budapest University of Technology and Economics
Department of Measurement and Information Systems
Budapest, Hungary
Email: {hkristof, bank}@mit.bme.hu

***Abstract*—In the field of audio signal processing, logarithmic frequency resolution IIR filters, such as fixed-pole parallel filters and Kautz filters, are commonly used. These proven structures can efficiently approximate the frequency resolution of hearing, which is a highly desired property in audio applications. In recursive adaptive filtering however, the FIR structure with LMS algorithm is the most commonly used. Since the linear frequency resolution of FIR filters is less than ideal for audio applications, in this paper we explore the possibility of combining the logarithmic frequency resolution IIR filters with the LMS algorithm. To this end the LMS algorithm is applied to fixed-pole parallel and Kautz filters, and the resulting structures are compared against each other and to the FIR-LMS filters in terms of convergence time and remaining error.**

***Index Terms*—audio signal processing, LMS, fixed-pole parallel filters, Kautz filters**

I. Introduction

Infinite impulse response (IIR) filters are commonly used in audio signal processing [1], where logarithmic frequency resolution is highly desired when modeling a transfer function. To achieve this, specialized filter design methodologies have been developed, including warped filters [2], second-order fixed-pole parallel filters [3], and Kautz filters [4].

In adaptive filtering, finite impulse response (FIR) filter structures with least mean squares (LMS) method are popular choices. The reason for their popularity is their global convergence, however, they require more parameters to model a given response, as opposed to IIR filters. Another drawback is that their residual error (misadjustment) is related to the step-size coefficient (μ), and thus, a trade-off must be made between convergence time and residual error [5].

Common applications for adaptive audio filters, such as compensation, or noise reduction, contain an adaptive filter that identifies a given signal path. Thus, as a first step for comparing logarithmic frequency resolution IIR filters in adaptive context, this paper explores the identification capabilities of the different IIR structures using LMS algorithm.

In this paper, the LMS algorithm is applied to the parallel and Kautz filters, and the resulting adaptive IIR filters are compared to each other and to the common FIR-LMS filters.

II. The LMS Algorithm

The Least Mean Squares (LMS) algorithm is a stochastic grade descent method where the coefficients are adapted based on the current error in time [5]. It uses the estimate of the mean square error (MSE) gradient vector from the available data, to make successive corrections to the filter coefficients in the direction of the negative of the gradient vector. This iterative procedure eventually leads to minimum mean square error.

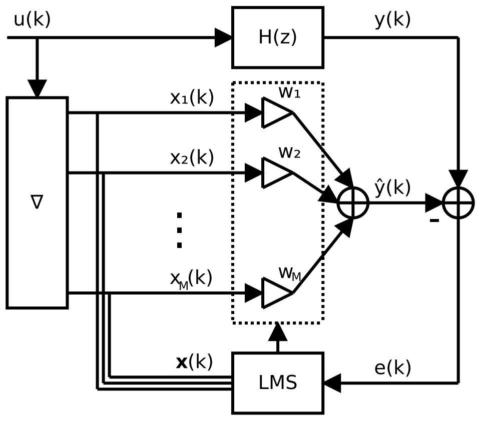


Fig. 1. LMS-based adaptive filter used for identification.

The block scheme of the LMS filter can be found in Fig. 1. The common input of the system to be identified and the adaptive filter is denoted by $u(k)$, and the outputs are marked by $y(k)$ and $\hat{y}(k)$ respectively.

The output of the adaptive filter is computed as

$$\hat{y}(k) = \mathbf{w}^\top(k)\mathbf{x}(k). \tag{1}$$

The recursive function for coefficient adaptation is the following:

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \mu e(k)\mathbf{x}(k), \tag{2}$$

where $\mathbf{w}$ denotes the filter coefficients, k is the discrete time, μ is the step-size parameter, $\mathbf{x}$ is the estimated gradient and e is the output error, where $e(k) = y(k) - \hat{y}(k)$.

The input vector $\mathbf{x}(k)$, which acts as the estimated gradient vector, is unique for every filter structure. For FIR filters, it is a delay line; for other structures it can be deduced using Equation 1.

Note that each element of $\mathbf{x}(k)$ is a function of time, and they span the space of the output function. Because they act as base functions, their correlation has an impact on the convergence time: the lower the eigenvalue spread of the correlation matrix $\mathbf{R}$, the faster the convergence [5].

The estimated autocorrelation matrix of $\mathbf{x}(k)$ is calculated as:

$$\hat{\mathbf{R}} = \frac{1}{L}\sum_{k=0}^{L}\mathbf{x}(k)\cdot\mathbf{x}^{\top}(k), \tag{3}$$

where L denotes the number of samples.

The main drawback of the LMS algorithm is that the gradient vector scales with the input, which can cause instability in the adaption. As a remedy, the Normalized-LMS (NLMS) method is used, which normalizes the power of the input [5]:

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \mu\frac{e(k)\mathbf{x}(k)}{\alpha + \mathbf{x}^{\top}(k)\mathbf{x}(k)}, \tag{4}$$

where α is a small positive number used to avoid the denominator to become zero.

In this paper we used the NLMS algorithm for realizing adaptive filters.

III. Adaptive IIR Filters

Adaptive IIR filters require fewer parameters compared to FIR filters, however, early research showed that adaptively varying both the poles and zeros can lead to suboptimal performance caused by multimodal error surfaces [6] or because they require satisfaction of a strict positive real condition [7].

Alternatively, the poles of the IIR filter can be fixed at predetermined values, which preserves the linearity in parameters and leads to well-behaved adaptation properties [8].

In audio signal processing, fixed-pole filters are commonly used. The Kautz (Fig. 3) and the fixed-pole parallel filters (Fig. 2) are proven to have equivalent transfer functions when designed off-line [3]. The main difference between them lies in the computational demand (see Table I): the fixed-pole parallel filter need approximately 47% less operations compared to the Kautz filter. The tap outputs of the two filters span the same space, but the base functions of the Kautz filter are orthonormal [9]. This results in convergence properties similar to that of FIR filters [8].

The general structure of the parallel second-order structure can be found in Fig. 2. The second-order sections can be implemented as either direct-form, or other structures [10]. Note that the structure of the second-order sections have direct impact on the parameters, and thus, affects the convergence properties if the second-order section is used in an adaptive filter realization.

Adapting the aforementioned fixed-pole audio filters using the LMS algorithm can be done by substituting the IIR filter to the ∇ block in Fig. 1, with the output multiplications and summation replaced by the adaptive linear combination of the LMS algorithm. For example, in case of the Kautz filter in Fig. 3 it means that the c_i coefficients are the tuned parameters.

IV. Orthogonal Second-order Section

In order to improve convergence, we present a new second-order structure (Fig. 4), which, to our knowledge, has not been presented before. The new structure is equivalent to a second-order Kautz filter, therefore its two tap outputs are orthogonal. As this structure is more complex than the direct-form implementation, its usage in parallel filters result in computational demand between the direct-form parallel filter and the Kautz filter.

TABLE I
Number of arithmetic operations requied for the tested adaptive IIR filters having N conjugate-complex pole pairs implemented using direct-form 2 (DF2) or orthogonal second-order sections.

	Multiplication	Addition
Fixed-pole parallel filter (DF2)	$6N$	$3N-1$
Fixed-pole parallel filter (orth.)	$6N$	$5N-1$
Kautz filter (DF2)	$9N+2$	$8N+1$

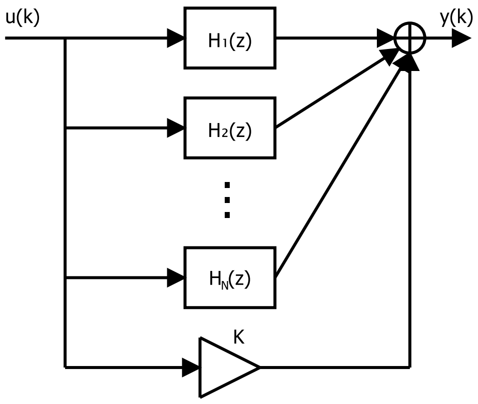


Fig. 2. Parallel second-order filter. Note that in our investigations we omitted the constant K section.

The parameters a_1 and a_2 are the same as in the direct form. The p and q coefficients can be computed from the direct-form parameters b_0 and b_1 with the following formulas:

$$p = \frac{b_0 - b_1}{2}, \tag{5}$$

$$q = \frac{b_0 + b_1}{2}. \tag{6}$$

The estimates of the autocorrelation matrices can be found in Fig. 5. It can be seen that the orthonormal property of the Kautz filter results in a unity autocorrelation matrix. In fixed-pole parallel filters however, the neighboring tap outputs have high levels of cross-correlation. This effect is lower when the orthogonal second-order sections are used: only the tap outputs of the different sections are correlated, resulting in a periodic pattern.

V. Normalizing the Tap Outputs

The convergence rate of the LMS algorithm is related to the eigenvalues of the $\mathbf{R}$ matrix [5]. It is shown that if the eigenvalue spread of the $\mathbf{R}$ matrix is the minimum over all possible matrices, the maximum convergence rate can be achieved. As a consequence, the tap outputs of the filter (denoted by $\mathbf{X}(k)$) having the same output power is a necessary condition. This criterion is inherently satisfied for orthonormal filters [8], but not for fixed-pole second-order

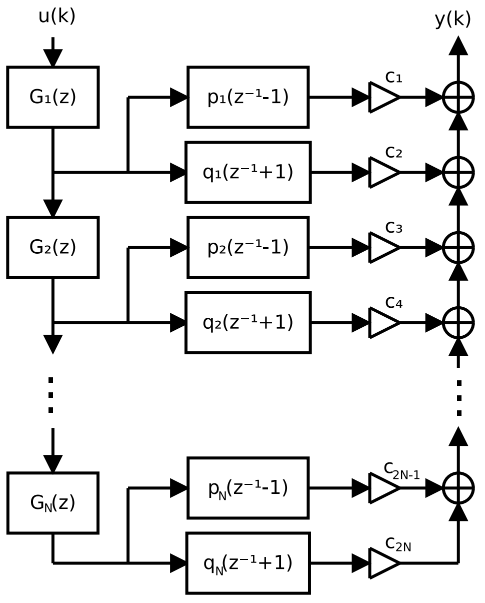


Fig. 3. Kautz filter structure.

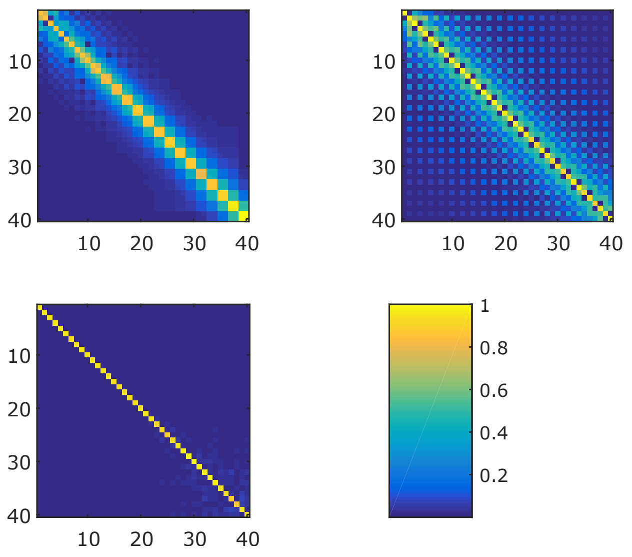


Fig. 5. Visualization of $\hat{\mathbf{R}}$ matrices. Top left: parallel filter with direct-form sections; top right: parallel filter with orthogonal second-order sections; bottom left: Kautz filter.

filters. Therefore, the tap outputs of the second-order sections need to be scaled.

To determine the normalizing coefficients, we compute the impulse responses between the input and the tap outputs. The scaling factors are then determined by the sum of squares of the impulse responses:

$$s_i = \frac{1}{\sum\limits_{k=0}^{\infty} \big(h_i(k)\big)^2}, \tag{7}$$

where h_i denotes the impulse response between the filter input and the i-th filter tap output. Using this scaling, the tap outputs will have the same power when the input is white noise.

VI. Comparisons

In our investigation we used the NLMS algorithm as a method for system identification (Fig. 1). The input was a white noise uniformly distributed in range $[-1;+1]$. The system to be identified was implemented using a 10000-tap long FIR filter, whose coefficients were based on actual impulse response measurements.

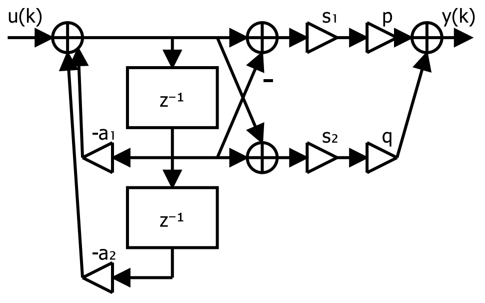


Fig. 4. Orthogonal second-order structure, with normalizing terms s_1 and s_2.

The filters to be compared are fixed-pole second-order parallel filters (without FIR section), implemented using both direct-form and improved second-order sections, a Kautz filter and as a reference, a FIR filter. The IIR filters have 20 conjugate complex pole pairs, placed along a logarithmic scale between 20 Hz and 20 kHz, assuming 44.1 kHz sampling frequency. The quality factors of the poles were chosen that the neighboring sections had their magnitude response cross at their -3 dB point [11]. The FIR filter has 40 taps, thus the filters have the same amount of free parameters.

The mean square error (MSE) of adapted filter parameters are computed on a logarithmic scale: the error, denoted by $e(k)$ in Fig. 1, has its DFT spectrum sampled at certain frequencies having logarithmic distribution. The samples are then squared and summed from 20 Hz to 20 kHz, assuming $f_s = 44.1$ kHz sampling rate:

$$E(j\omega) = \mathrm{DFT}\{e(k)\}, \tag{8}$$

$$\mathrm{MSE} = \sum_{f=20\mathrm{Hz}}^{f=20\mathrm{kHz}} \left|E(j2\pi f/f_s)\right|^2. \tag{9}$$

For comparison, the MSE was calculated for all structures at every 256 samples and then plotted.

In our investigation, we used two example transfer functions for testing the algorithms: a minimumphase one-way loudspeaker (Fig. 6 top) and a larger, two-way loudspeaker with non-minimumphase response (Fig. 6 bottom). In the figures, we marked the result of the off-line LS design as well as the magnitude response of the adaptive fixed-pole parallel filter that is implemented using orthogonal second-order sections. Note that the transfer function of the adaptive Kautz is omitted because it fits the LS solution after the simulation time (65536 samples).

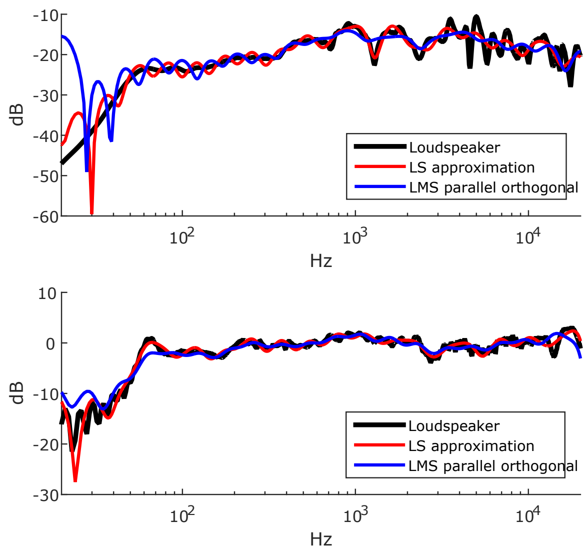


Fig. 6. Magnitude plots of the example transfer functions (black lines). Top: minimumphase one-way loudspeaker; bottom: non-minimumphase two-way loudspeaker. The LS approximations are plotted using red lines. The magnitude responses of the fixed-pole parallel filters using orthogonal second-order sections, after 65536 samples, are also plotted (blue lines).

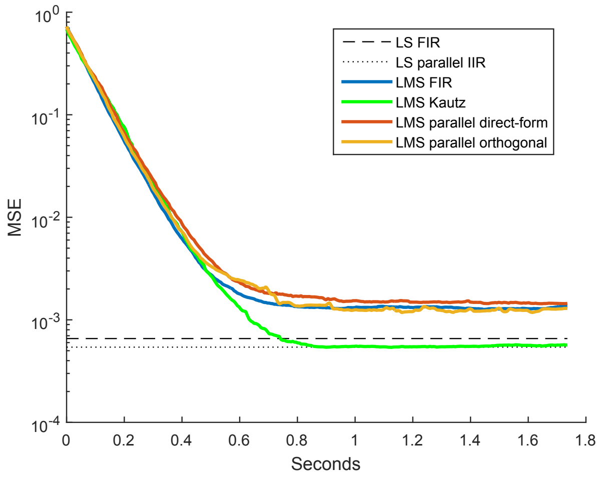


Fig. 7. MSE by time, for a minimumphase one-way loudspeaker response.

The MSE plots of the systems identifying the example transfer functions can be found in Fig. 7 and 8. For each of the filters, the μ step-size parameter is tuned in a way that the curves would have the best fit with each other on the first 12800 samples. As reference, the MSE of offline designed filters, based on the LS approximation, are shown on the figures using dashed and dotted horizontal lines.

According to figures 7 and 8, the Kautz filter has the best convergence: for the minimumphase system its MSE is on par with the LS approximation, and for the non-minimumphase system it has the fastest convergence among the tested structures.

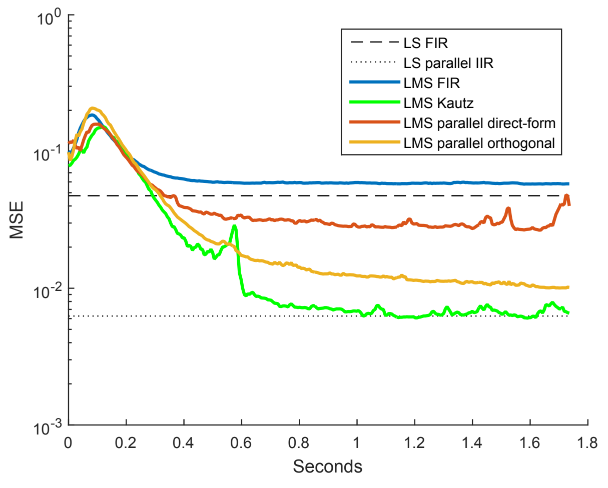


Fig. 8. MSE by time, for a non-minimumphase two-way loudspeaker response.

VII. Conclusion

This paper compared the LMS-based adaptive implementations of the most common fixed-pole IIR filters used in audio. As a result, we recommend to use the Kautz structure in LMS-based adaptive audio filters, if its computational demand can be satisfied.

Future research includes the usage of other filter structures: the delayed fixed-pole parallel filter, a modified Kautz structure with FIR component, and the resonator-based filter.

References

[1] V. Välimäki and J. D. Reiss, "All about audio equalization: Solutions and frontiers," *Applied Sciences*, vol. 6, no. 5, 2016, art. no. 129, doi: https://doi.org/10.3390/app6050129.

[2] A. Härmä, M. Karjalainen, L. Savioja, V. Välimäki, U. K. Laine, and J. Huopaniemi, "Frequency-warped signal processing for audio applications," *J. Audio Eng. Soc.*, vol. 48, no. 11, pp. 1011–1031, Nov. 2000.

[3] B. Bank, "Audio equalization with fixed-pole parallel filters: An efficient alternative to complex smoothing," *J. Audio Eng. Soc.*, vol. 61, no. 1/2, pp. 39–49, Jan. 2013.

[4] T. Paatero and M. Karjalainen, "Kautz filters and generalized frequency resolution: Theory and audio applications," *J. Audio Eng. Soc.*, vol. 51, no. 1–2, pp. 27–44, Jan./Feb. 2003.

[5] S. S. Haykin, B. Widrow, and B. Widrow, *Least-mean-square adaptive filters*. Wiley Online Library, 2003, vol. 31.

[6] S. Stearns, "Error surfaces of recursive adaptive filters," *IEEE Transactions on Circuits and Systems*, vol. 28, no. 6, pp. 603–606, 1981.

[7] C. Johnson, M. Larimore, J. Treichler, and B. Anderson, "Sharf convergence properties," *IEEE Transactions on Circuits and Systems*, vol. 28, no. 6, pp. 499–510, 1981.

[8] G. A. Williamson and S. Zimmermann, "Globally convergent adaptive iir filters based on fixed pole locations," *IEEE transactions on signal processing*, vol. 44, no. 6, pp. 1418–1427, 1996.

[9] J. Cousseau, G. Sentoni, P. Diniz, and O. Agamennoni, "On orthogonal parallel realization for adaptive iir filters," in *Proceedings of Third International Conference on Electronics, Circuits, and Systems*, vol. 2. IEEE, 1996, pp. 856–859.

[10] K. Horváth and B. Bank, "Optimizing the numerical noise of parallel second-order filters in fixed-point arithmetic," *Journal of the Audio Engineering Society*, vol. 67, no. 10, pp. 763–771, 2019.

[11] B. Bank, "Audio equalization with fixed-pole parallel filters: An efficient alternative to complex smoothing," in *Proc. 128$^{\text{th}}$ AES Conv., Preprint No. 7965*, London, UK, May 2010.

Simulating Urban Traffic as a Multilayered Multiagent System

Levente Alekszejenkó
Department of Measurement and Information Systems
Budapest University of Technology and Economics
Budapest, Hungary
ale.levente@gmail.com

Tadeusz Dobrowiecki
Department of Measurement and Information Systems
Budapest University of Technology and Economics
Budapest, Hungary
dobrowiecki@mit.bme.hu

***Abstract*—The vehicles of the future will be capable of communicating with each other and with the road infrastructure as well. Based on this ability, complex multiagent systems can be designed, including smart cars and intelligent traffic control systems (referred to as judges).**

Such system was implemented by extending an open-source traffic simulation tool, called Simulation of Urban MObility platform (SUMO). The implemented system can be used to experiment with and to verify various algorithmic approaches aimed to increase the intelligence of autonomous drivers and urban traffic controllers.

Our study investigated the adaptation of the operating system task schedulers and the Explicit Congestion Notification algorithm of computer networks. It resulted in a layered cooperative multi agent system composed from platooning car drivers in the lower layer and the cooperating intersection judges in the upper layer.

Results indicate that the implemented system can organize the traffic better in extraordinary cases (e.g. an accident, road works on some major streets, etc.). The regulatory capability of the proposed system depends greatly on the topology of the road connections. This aspect (especially the problem of congestions) is currently under investigation.

***Index Terms*—intelligent traffic control, smart vehicles, multiagent system, platooning, explicit congestion notification, scheduling algorithms**

I. Introduction

One of the major problems of our cities is the regular congestion of road networks. As the number of vehicles is rapidly increasing, improving the flow of the traffic and reducing traveling time becomes an even more challenging task. The connected vehicles of the future and the wide variety of IoT devices implemented in the road infrastructure may create new ways to optimize the traffic.

For example, smart cars can form groups, so called *platoons*, near intersections. The cars which form a platoon can change lanes or can pass through intersections together, therefore causing less impact on the traffic.

Another possibility is to create intelligent traffic controllers, so called *judges*. Let us suppose that the number of incoming vehicles from each direction is known. In this case, as the demand is known for the near future, theoretically well established scheduling algorithms can be applied. Moreover, these judges can be made cooperative as well in order to make a globally optimal solution.

Some ideas have already been implemented as a multilayered multi-agent system. In our implemented system there are two types of agents, i.e. smart cars and judges. They can communicate with each other in order to perform some intelligent actions. For example, smart cars can form platoons, or ask the judges whether they can pass through an intersection. Judges can also send messages to each other, cooperatively evaluating the state of the roadnetwork, to attempt to avoid the congestion. The performance of our system was validated by simulations. The used simulation platform was created by extending the Simulation of Urban MObility (SUMO) [5] microscopic traffic simulation program.

II. Literature Review

Creating platoons of smart cars, besides reducing the computational demands on intelligent traffic controllers, results in a more efficient lane-changing strategy. Let us suppose that the lane-changing of a platoon can be modeled as a single lane-change of a truck. In [9] the authors have shown that a double semi-trailer truck is equivalent to 3 personal cars. It takes, however, more space on the road than those 3 cars. Consequently, platooning also seem to be an effective way to reduce the impact of lane-changes.

Consider now the perspective of a judge, i.e. an intersection controller. The task of the judge is analogous to that of the scheduler of an operating system. Both are responsible for deciding which competing entity (task or vehicle) can use a unique resource (the processor or the part of the intersection). A scheduling solution to control intersection lamps was suggested by [1], where so called Minimal Destination Distance First (MDDF) method was used, based on the well known Shortest Job First scheduler (SJF)[1] of operating systems. Unfortunately, that proposed algorithm is not fair and was verified only in a highly regular intersection environment.

Coordination of the traffic signals is an old idea, and for example can be achieved by green-waves. There are some

The research has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013, Thematic Fundamental Research Collaborations Grounding Innovation in Informatics and Infocommunications).

[1]To be precise it is based on the Shortest Remaining Time First, the preemptive version of SJF.

traditional algorithms, like TRANSYT and SCOOT [7], which try to shorten the queue lengths behind the traffic lights. In the last decades, new methods were published, for example the one based on a reservation system [2]. In this algorithm, smart cars have to book time and space slots when they are permitted to pass through the intersection. An intersection manager stores these bookings and checks whether an incoming booking is feasible. This system has a major disadvantage, namely when there is a vast amount of vehicles with a vast amount of bookings, the feasibility check would be a really processing-intensive task. An agent based solution was proposed in [10]. In this approach every neighboring intersection was connected, therefore it is theoretically possible to create unstable states. The green time of a traffic signal is only modified a little bit, making its neighbors also to modify a little bit more, and so on. As a result of this butterfly-effect, the whole system might become unstable, causing unpredictable traffic flows, therefore increasing the risk of an accident.

III. Interaction Between Smart Cars – Platooning

A. Formation of a Platoon

Smart cars are basically competing agents (for the green light slot), but they are willing to form a coalition – a *platoon* – in order to go through an intersection as efficiently as possible, if their interests (path) coincide.

When a smart car approaches an intersection[2], it has to join a platoon. The cars of a platoon have the exactly same trajectory: they arrive at the intersection from the same direction, in the same lane and leave via the same exit lane. For the time the platoon exists, its cars are joined virtually into a chain maintaining about 5 m of distance between each other.[3]

If the platoon in front of a smart car is not suitable, the smart car has to create a new platoon.

In the front of the platoon is the *platoon leader*, all the other vehicles in the platoon are the *platoon members*. Platoon leaders are responsible for their platoons, and the platoon members have to follow the platoon leader.

After crossing the junction, the platoon leader exits its platoon and passes over its prerogatives to the next-in-line in the platoon. This smart car will be the new platoon leader. It is an easy and effective way to avoid the problems which can be caused by the preemptive scheduler of the judges.

B. Lane-Changing of Platoons

Reducing the impact of changing lanes before intersections can provide a significant improvement in the traffic flow. In the SUMO platform sophisticated lane change models are already implemented. In our research we modified the SL2015 model [3] to calculate also with the platooning concept. So while a smart car belongs to a platoon, it has to behave differently, depending on whether it is the leader or a simple member. In a platoon only the platoon leader can make a lane-change decision. All the other members have to follow the car in front of them.

If the platoon leader finds out that a lane change is needed[4], it makes contact with the platoon leader (if there is any) in the target lane. The two platoon leaders make an agreement whose platoon will be ahead of the other.[5] Platoon in the target lane will slow down or even stop if necessary to make sure that the maneuver will be successful. Another possibility is that the mover leader has to wait until the asked leader and its platoon leaves the target lane. The platoon manages the lane change car-by-car, sending lane-change command down the platoon chain.

IV. Intersection Controlling Algorithms of Unconnected Judges

In our first approach, unconnected (i.e. non-communicating) judges were implemented. From the operating system field we borrow two simple scheduling algorithms. One is the Round Robin (RR) algorithm, which is fair (free of starvation) and the other is the Shortest Job First (SJF), which yields an optimal response time, but is unfair. These two simple schedulers (and their preemptive versions respectively) provide the basics of all kinds of much more complex scheduling algorithms. Due to this fact, we decided to try out these two methods, as conflict class[6] selector algorithms of an intelligent judge agent.

1) RR: A simple round robin scheduler can be implemented as a traffic controlling method without any significant modification to the original algorithm. We prescribe time slices to each conflict class. This will be the maximal amount of time in which a conflict class can be active. After this time slice is elapsed, we simply select the next conflict class from the list.

2) MDDF: [1] *Minimal Destination Distance First* traffic controlling system is based on the optimal scheduler, called Shortest Job First (specifically its preemptive version, the so-called Shortest Remaining Time first). The problem with this solution is that it is not fair.

Let us suppose that a lonely car is waiting in an intersection to pass. This car is at the beginning of its route to a very distant destination, but vehicles with significantly shorter routes are continuously arriving. The car with the long route to its destination can wait forever without getting through this intersection.

To make the algorithm fair, we redefined our scheduler as a two-level scheduler. On the higher priority level a simple Round Robin scheduler is running, and on the lower priority level a scheduling algorithm similar to the implementation of [1] is used. At first, every conflict class is scheduled by the lower priority level. If a conflict class was not active in the last 90 s, it would change its priority to the high level. This

[2]Some markers are placed as new traffic signs which instruct the smart cars to join or leave a platoon.

[3]Platooning in this case is an adhoc formation and slightly differs from platoons created on highways. The aim of our platoons is to pass through intersections more effectively than individual vehicles are able to do so.

[4]In this state, smart cars' lane change model calculates with length of platoons instead of single vehicles.

[5] [3] has already worked out the protocol and algorithm of this agreement. We modified the existing solution to have the contract made only between platoon leaders instead of single vehicles.

[6]A conflict class is a group of cars, which are permitted to pass through an intersection simultaneously.

way it is guaranteed that every vehicle will be scheduled in a limited amount of time. The Round Robin also prevents the occurrence of starvation.

V. Intersection Controlling Algorithms of Connected Judges

A. The ECN-based method

It is a simple idea to connect the judges (i.e. to permit them to communicate) with each other in order to improve the capabilities of the system. This improvement is a signal coordination which aims to prevent the formation of congestions. Such algorithms are already in use in the domain of computer networking. However, the character of the traffic is different and the majority of them cannot be applied in the road traffic environment. An algorithm which could be applied, or at least be experimented with, is the so called Explicit Congestion Notification (ECN) method [4], which we implemented in our simulations.

The basic idea behind ECN is that the receiver node (an intersection manager or a network router) can inform the sender if the queue length (of vehicles or datagrams) at the receiver side reaches a certain level (let us call this signal the *ECN-signal*). This means that a congestion is about to form. To avoid the congestion, the sender must decrease its output in this case. A new kind of judge, the so-called ECN-judge, was implemented which is based on this discussed method.

The ECN algorithm has a great advantage that it does not require the definition of arterial directions[7]. Defining arterials would demolish the merits of the intelligent system in extraordinary situations, when the proposed system can clearly outperform the traditional system, for details, see Section VI.

B. Challenges in the Implementation

The state-space of the ECN judge can be enormous since it depends both on the number of incoming vehicles and on the number of the neighboring intersections. Therefore, storing a signal plan for all of the states is quite memory-consuming. Instead of doing this, a dynamical signal plan generation method was implemented (for an overview, see Figure 1).

The calculation of simple signal phase can be formalized as an *integer programming problem (IP)*. Our goal is to maximize the number of directions which receive green light at the same time, subject to the actual state of the network. This state consists of dynamic parts, like the incoming ECN-signals or the decision of a scheduling algorithm (eg. a Round Robin) as well as static parts, which describes which directions cannot pass through an intersection simultaneously.

If the IP is solved, we only know the signal phase for a given moment. In order to generate a signal plan (which describes how long a direction should get a green or a red light), it is necessary to recalculate this IP problem from time to time. In our implementation the recalculation time is a linear function of the number of incoming vehicles, but cannot exceed 45 seconds.

[7]Arterial direction is the main route which for example receives a green-wave.

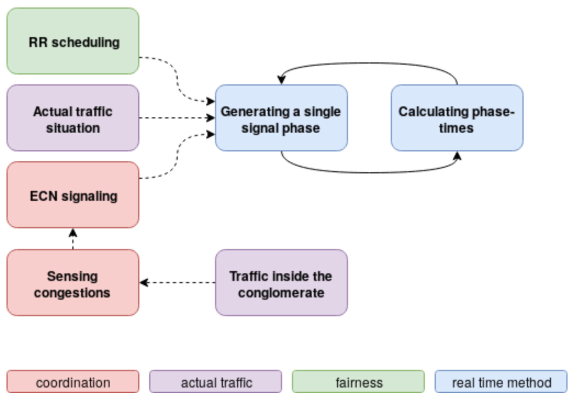


Fig. 1. Overview of the ECN-judge

There is another topic which should be discussed, namely the identification of the forming congestions. It is a quite difficult task [6], [8], and our research did not focus on solving this problem, thus based on preliminary simulations, we simply calculated the traffic density, which can provide the highest traffic flow. We say that there is a congestion forming when the 90% of this level is reached, so the ECN-signal is sent at this event.

VI. Simulations

Our solutions were tested by an extension to the Simulation of Urban MObility program (see Figure 2 for details). The simulated network was the BAH intersection[8] of Budapest and its close neighborhood.

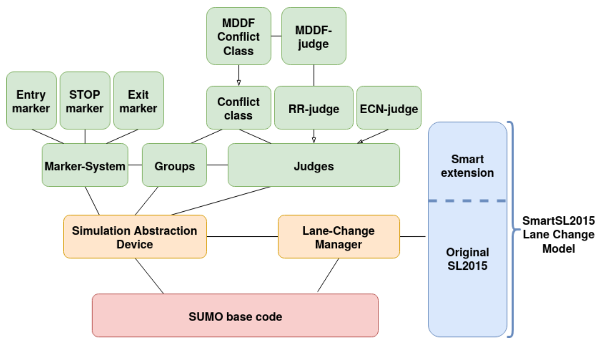


Fig. 2. The developed extension of the SUMO. The components of the multilayered, multiagent system are shown in green. Some modules are necessary to create an abstraction layer between the original source code of the SUMO and the intelligent system's layer. This abstraction layer is presented in orange and blue in this figure.

Basically two types of traffic demand were modeled: some cases of regular traffic (eg. night traffic, morning traffic, noun traffic) and irregular traffic (Budaörsi út is closed[9]) were fed into the simulator.

[8]Where streets of Hegyalja út, Jagelló út, Villányi út, Budaörsi út and Alkotás utca intersect.

[9]*Irregular1* case: Obstacle is northbound of "Budaörsi út", can be bypassed via Karolina and Villányi streets.
Irregular2 case: Obstacle is southbound of "Budaörsi út", bypass route is via "Hegyalja út".

A. Simulating The Unconnected Judges

As a first attempt, we tested the behavior of the platooning system and the simple, unconnected judges. These measurements basically show that such kind of systems may be able to reduce waiting and traveling times through this intersection.

The results show (see Table I and Table II) that such a system is able to decrease the waiting (for example at red lights) and average traveling times in irregular situations. On the other hand, the improvement of the traffic flow[10] is not so obvious in regular cases, see Figure 3.

TABLE I
SIMULATION RESULTS OF "IRREGULAR1" CASE

Test case	Arrived (%)	Waiting Time (s)	Average Traveling Time (s)
Traditional	33.81	29.68	170.55
RR	29.19	12.117	174.87
MDDF	22.77	12.41	154.02

TABLE II
SIMULATION RESULTS OF "IRREGULAR2" CASE

Test case	Arrived (%)	Waiting Time (s)	Average Traveling Time (s)
Traditional	38.48	36.44	199.38
RR	32.71	11.43	170.07
MDDF	34.39	10.74	176.72

B. Simulating The Connected Judges

In order to improve the traffic flow, some judges[11] were reprogrammed to ECN-judges. Theoretically this system would have greater chance to find a globally optimal solution, than the unconnected judges, which are only capable of finding a locally optimal scheduling.

The trial of the system gave surprising results. Instead of improving the flow of the traffic in the BAH-intersection, this method rather reduced this value. As it can be seen in Figure 3, the new judges limit the density of the traffic to around 65-70 vehicles/km, almost regardless of the height of the traffic demand. (With a combined system, which contains both connected ECN-type and unconnected Round Robin-type judges, this limit is slightly higher.) Partly by this density limitation, partly by some yet unknown effects, the traffic flow is strongly reduced by the ECN-judge system.

VII. Conclusion and Further Research Aims

As the traditional system is likely to be numerically optimized, it is a challenging task to achieve the same or even better results with a new intelligent solution in regular cases. On the other hand, in extraordinary situations, an intelligent,

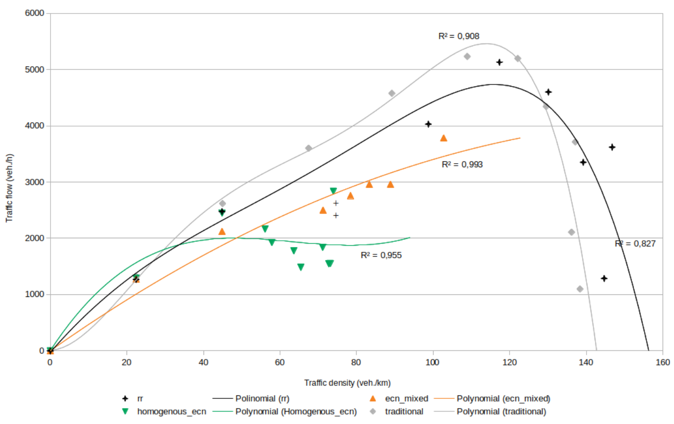


Fig. 3. Traffic flow in the traditional and in the unconnected intelligent system, consisting of RR-type judegs.

multi-agent based solution can be much more flexible. This flexibility provides better traveling times by reducing the unnecessary waiting times.

The ECN-judges have no benefits in the BAH intersection scenario, if our goal is to improve the flow of the traffic. Supposing that there are situations where the traffic density (and therefore the flow) limitation is a desired effect, our proposed system might be beneficial as well. Such situations can be the limitation of the traffic going through residential areas or nature reserves.

Further research is needed to verify that the ECN-judge system is able to cause such effect in these kinds of networks.

References

[1] F. Ahmad, S. A. Mahmud, G. M. Khan, and F. Z. Yousaf, "Shortest remaining processing time based schedulers for reduction of traffic congestion," in 2013 International Conference on Connected Vehicles and Expo (ICCVE), Las Vegas, NV, USA, 2013, pp. 271–276.

[2] K. Dresner and P. Stone, "A Multiagent Approach to Autonomous Intersection Management," JAIR, vol. 31, pp. 591–656, Mar. 2008.

[3] J. Erdmann, "SUMO's Lane-Changing Model", in Modeling Mobility with Open Data, M. Behrisch and M. Weber, Edit. Cham: Springer International Publishing, 2015, pp. 105–123.

[4] S. Floyd, K. K. Ramakrishnan, and D. L. Black, "The Addition of Explicit Congestion Notification (ECN) to IP." [Online]. Accessible: https://tools.ietf.org/html/rfc3168#section-1. [Accessed: 2019. okt. 25.].

[5] P. A. Lopez et al., "Microscopic Traffic Simulation using SUMO," in 2018 21st International Conference on Intelligent Transportation Systems (ITSC), 2018, pp. 2575–2582.

[6] K. Nagel, P. Wagner, and R. Woesler, "Still Flowing: Approaches to Traffic Flow and Traffic Jam Modeling," Operations Research, vol. 51, no. 5, pp. 681–710, Oct. 2003.

[7] D. I. Robertson, "Research on the TRANSYT and SCOOT Methods of Signal Coordination," ITE Journal, Jan 1986 pp. 37-40.

[8] M. Treiber and A. Kesting, Traffic Flow Dynamics. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[9] N. Webster and L. Elefteriadou, "A simulation study of truck passenger car equivalents (PCE) on basic freeway sections," Transportation Research Part B: Methodological, vol. 33, no. 5, pp. 323–336, 1999.

[10] J. Withanawasam and A. Karunananda, "Multi-agent based road traffic control optimization," in 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC), Yokohama, 2017, pp. 977–981.

[10] The traffic flow is a commonly calculated value. It is the product of the traffic density ($\frac{\text{vehicles}}{km}$) and the mean velocity of the vehicles ($\frac{km}{h}$). These values can be measured by different types of detectors, cameras, etc.

[11] Namely the judge supervising the intersection of Villányi and Budaörsi streets, the one supervising the Budaörsi, Hegyalja and Alkotás street intersection and the one placed at the Jagelló and Hegyalja crossing.

Support of System Identification by Knowledge Graph-Based Information Fusion

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

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

***Abstract*—The paper presents a knowledge graph-based solution for creating the core models for supervisory control of complex Cyber-Physical Systems (CPS) and computing infrastructures from design models and operational logs.**

The core element of modern supervisory control approaches is a digital twin, which maps the observations about the system into a hybrid run-time model representing the expected system state. It serves as a basis for interaction between the controller and the controlled system.

The high-level discrete-state machine of digital twins represents the different operational regimes (domains of similar behavior) and transitions between them. A continuous model describes the intra-domain behavior in detail. A special case is the qualitative domain model using discretized state variables in the form of a few ordered values (e.g. low, medium, high). This model category is extremely beneficial, when representing partial, dimensioning dependent behavior.

Parts of the digital twin model can be directly derived from the design models, but the description of dynamics of the qualitative domain models necessitates system identification from observations (operation logs or benchmark results). This way the creation of the digital twin necessitates information fusion from different sources. Knowledge graphs provide an abstract semantic framework for this purpose.

Our goal is to support system identification by deductive reasoning performing step-by-step checks of the abstract model to assure consistency and completeness of the observations and their respective evolving models.

***Index Terms*—cyber-physical systems, system identification, knowledge graph, digital twin**

I. Introduction

The purpose of cyber-physical systems (CPS) [1] is to observe and control the physical world through intelligent mechanisms. They operate over continuous and discrete signals originating in the physical world, for which they consist of physical and computational components interacting through communication layers [2].

The core concept in modern supervisory control of CPSs is the “digital twin.” Data delivered by sensors continuously synchronize this model of the system under control with the physical world. Assurance of dependability and resilience of critical CPSs necessitates the faithfulness of the twin model.

Modern CPS design relies on the integration of pre-implemented components. The compliance to the designated temporal properties (timeliness, throughput, etc.) necessitates a proper dimensioning of the resources allocated to the components prior to the deployment.

The performance domain can influence the logic behavior. Non-linear effects resulting in bottlenecks, like the saturation of a particular resource, may change the dynamic behavior of the system. Moreover, CPS activates the built-in overload protection mechanisms, this way the behavior of a particular component, subsystem, and the entire system depend both on the functional logic (functional architecture of the system) and on its parametrization.

This way, scalability of the digital twin, similar to the deployed system requires hybrid modeling (Fig. 1) approach separating the dimensioning-independent overall logic of the behavior (discrete domain) and its actual state within an operation regime under the current workload and parametrization (continuous domain).

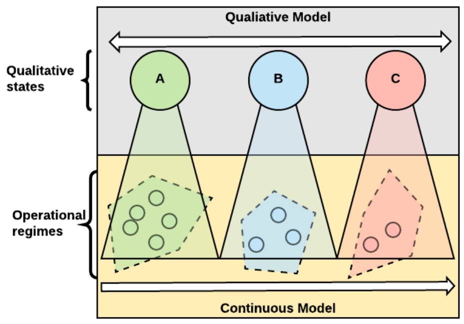


Fig. 1. Hybrid modeling

Creating a hybrid model as part of the system identification process necessitates the clustering of the data into domains (operational regimes), which show qualitatively identical behavior of the system. This task referred to as discretization can

The results presented in this research report were established in the framework of the professional community of Balatonfüred Student Research Group of BME-VIK to promote the economic development of the region. During the development of the achievements, we took into consideration the goals set by the Balatonfüred System Science Innovation Cluster and the plans of the ”BME Balatonfüred Knowledge Center”, supported by EFOP 4.2.1-16-2017-00021.).

be performed by an expert manually or by automated means.

Manual methods executed by an expert have the advantage that the discretization process can rely on the background knowledge of the expert. Furthermore, experts are also able to use their domain skills for variable selections, among others. On the other hand, if the number of the measured characteristics is large (as required for finite granular models), a pure expert-based approach becomes impossible.

A. *Objective*

Our goal was to support system identification (Fig. 2) by merging the prior knowledge and the qualitative system model into a knowledge graph and perform step-by-step checks of the abstract model to assure consistency and completeness of the observations and their respective evolving models.

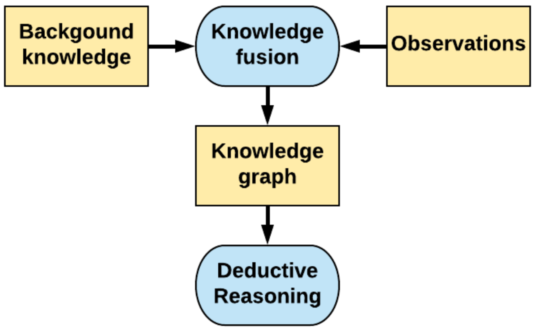


Fig. 2. Reasoning

One approach to achieve this goal uses the *phenomenological behavior* of the system which provides an abstract view of the system.

In our approach, we extend the phenomenon analysis with *prior knowledge* about the system. Knowledge graphs serve as information representation and fusion tool. Extending the knowledge graph with prior knowledge about the system provides a more detailed view and allows more precise reasoning about the system state.

If a new information (observation, input) does not fit into the knowledge graph it could indicate that 1) the digital twin model does not fit to the real system; 2) it indicates a faulty operation in the real system; or 3) the inputs are noisy. This way, deductive reasoning on the knowledge graph helps to identify these behaviors.

B. *Structure of the paper*

The rest of the paper is split into four main sections.

1) Section II presents how qualitative reasoning supports the definition of operational modes.
2) Section III presents the types of the prior knowledge in form of engineering models.
3) Section IV presents causal models in details and presents a causal model building method based on the engineering model.
4) Sections V presents a knowledge graph-based information fusion and deductive reasoning.

II. Qualitative reasoning

The discrete (qualitative) state machine is an abstract form of representing the logic of the dynamic behavior of the system. Its granularity corresponds to individual operation regimes (clusters of states of similar behavior) mapped to individual states with transitions activated by crossing the inter-cluster boundaries in the continuous state space. A continuous sub-model associated with each discrete state describes the intra-cluster behavior in detail.

An upper, discrete ”super”-model assures portability independent of the actual dimensioning when it covers the union of all abstract behaviors potentially occurring in some configuration.

It allows (qualitative) reasoning [3] about the system behavior by highlighting potential phenomena at a logic level. Moreover, it allows running simulations after the parametrization of the qualitative model to a quantitative one. Although, discrete modeling has many advantages, due to the high level of abstraction it may also cause ambiguity.

Moreover, the structure of the model is typically unknown, and its creation necessitates *observation-based system identification* or *prior knowledge-based model building*. Benchmarking and operational log analysis are the primary means to ensure the match between model architecture and observations.

The model formulation is about to determine the input description of the system. Input description takes into account the knowledge of the kinds of entities and phenomena that can occur (model fragment). It is also necessary to add constraints to the model about the boundaries of the system — this collected knowledge called domain theory. Knowledge bases allow storing the domain theory by providing a rich set of functionality (e.g., built-in reasoning) and representation mechanisms (relations, attributes, rules, etc.).

A. *Clustering*

The goal of clustering is the aggregation of the fundamentally similar states into a uniform qualitative state in the discrete state machine of the digital twin representing different operational modes. There are several approaches to achieve this goal:

1) **Speculative approach:** As operational modes at least in the logic domain and runtime resource management are subjects of the design process an initial clustering can be extracted from the design models. However, due to the complex interaction between logic functionality and resource management, the initial model has to be refined on the basis of observations originating in targeted experiments, benchmarks and operational log mining.
2) **Visual methods:** For example, visual EDA uses diagrams (e.g., scatter plots, time-series diagrams) to identify cluster and their respective boundaries of each operational mode. Because it is a heuristic process, it requires comprehensive domain knowledge.

3) **Algorithmic clustering:** Algorithmic clustering (e.g., decision tree, support vector machines, random forest, k-means) partitions the observed data into blocks corresponding to uniform operation modes. These are algorithmic processes that need manual verification to check the consistency with other models.

The representativity of the inout dataset needs verification by comparing the generated set of clusters with the initial state machine for proper coverage of states and transitions in a similar way as testing of models.

a) Cluster boundaries: Mapping continuous variables into qualitative values requires the definition of thresholds for discretization by a classification algorithm according to the clusters identified:

- **Input data**: The input of the digital twin is quantitative data. The discretized data perform synchronization of the digital twin with the compatible sets of the controlled system states. Classification (into operational modes) of the continuous incoming data is based on the identified thresholds.
- **Magnitude of the actuation**: Thresholds can be used to identify the magnitude of the actuation. This way, it can provide a more precise value for actuation by transforming the qualitative values into continuous ones.

III. Background knowledge

The input information of the method is a set of observations that usually came from benchmarks or operational logs. They describe the system (output metrics) under specific workload parameters.

The first step is to analyze the measurement campaign on its own. Experts have to take into account the context of the measurement and outlier data.

The context of the measurement covers the measured parameters and boundaries of the measurement campaign. Outlier data can warn about a non-functional operation of the system or indicates that the measurement is not trustworthy.

However, different parametrizations of the same experiment (i.e., different resource allocation) may expose profoundly different phenomena. A scalable model has to merge all of this even potentially different behaviors. This way, the model building has to be adopted to the fundamental configuration settings.

A deeper understanding of the measurement data requires a priori knowledge of the domain expert.

A. Modeling approaches

Processing the measurement requires having (partially) the system architecture and functional model. The information extracted from these models can be used during the evaluation phase. The design and development phase of the system provides background information on different abstraction levels.

However, it is possible to work with partial knowledge about the system. It is not necessary to know all the details (e.g., third-party components as a black box, only the input-output parameters are known with integration details).

Analysis of a system requires the collection of all prior knowledge that is available for the analyst. The background knowledge comes from different sources and covers different aspects of the system and includes the *architecture*, *functional*, *resource allocation*, *deployment*, and *causal model* of the system.

The *system architecture* and the *functional model* provides a high abstraction about the system components and their objectives. Causal models can be built by extracting information from other engineering models.

The *resource allocation* model closely connects to the functional model. It describes which component uses which resource (e.g., networking capabilities, CPU, RAM). The installation model presents the physical or logical layout of the system.

The *causal model* expresses the causal connections in the system based on prior knowledge about the domain and the previous models. Furthermore, it is possible to extend the causal model by adding external (out of the measurement campaign's context) causal connections to the model.

Collecting and systematizing the background knowledge is necessary for further analysis.

IV. Causal model

Causality is a natural, universal concept, so deeply present in our everyday life that we instinctively think in causal relations without pondering about their actual complexity and importance. The whole physical world around us is fueled by causality. It is the connection through which -under certain circumstances- one thing (the cause) influences another (the effect) in a deterministic way.

Causal models [4] [5] allow the exploration of the causal context of a system and the detection of independent properties and events. Causal graphs are one representation of causal models.

A causal graph is a *Directed Acyclic Graph* (DAG), where the relations represent the causation among the variables. Two variables of interest are distinguishable: 1) the *exposure* (independent variable, cause); 2) and the *outcome* (dependent variable, effect). Other variables (whether measured or not measured) are called *covariates*. Covariates can be categorized into several roles and they help in the further analysis of the system.

It is possible to build causal models (Fig. 3) by using classical engineering techniques (e.g. UML, SysML [6]). Classical engineering models collect the background knowledge that is required for building the causal model. The causal model is derivable from the functional model of the system and its resource allocation model (together with the deployment instance).

The *functional model* describes the continuous processes of the system, which defines the skeleton of the causal model. The causal model uses the described data flow by the functional model.

Extending the functional model with the *resource allocation model* also extends the causal graph with detailed causal

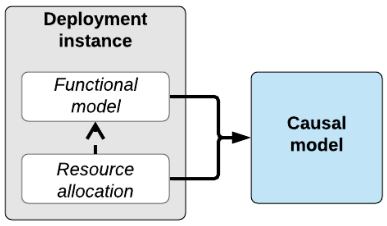


Fig. 3. Causal model building

relations. This model can present the causal connections between the resources and the functions (e.g., it is observable if two components using the same resource). This knowledge is usually verified by an expert who knows the system and its domain in great detail.

V. Knowledge representation

Knowledge graph (KG) is a kind of a database to organize complex networks of data. It provides a general-purpose approach to organize, efficiently store and query complex schemata to capture abstract concepts, entities, instances (represented as nodes of a graph) and their relations (the edges). Moreover, advanced knowledge database engines provide strong reasoning capabilities in an explainable and reusable form.

The simplicity and general validity of the underlying mathematical paradigm facilitate the use of a KG-based NoSQL database as the core element of digital twin creation and instantiation by merging the a priori knowledge with the observations.

The key asset (Fig. 4) ***in the creation phase of the digital twin model structure*** is an ontology-style merging of the design models (*architecture*, *functional*, *resource allocation*, *deployment*, and *causal model*) describing the different aspects of the system [7]. The methodology considers different input data and metamodels during the information fusion and uniformization.

A refinement of the initial core model in the KG enriches it with the domains of the variables, and their interactions as formulated in the qualitative model after processing the teaching set of observations.

Finally, the incoming stream of observations triggers a check of the consistency of the incoming data with the system model and updates the state of the digital twin model in the KG.

In this paper, GRAKN.AI [8] was used to store the models and qualitative benchmarking data.

A. Knowledge graph building

The schema of the GRAKN.AI knowledge graph is based on ER (Entity-Relationship) modeling. ER models include entities, relations, and attributes. It defines the objects of the examined world and the relationship between the objects. The objects could have attributes that describe their properties.

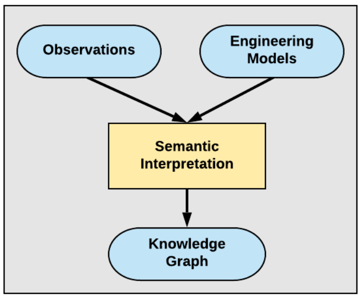


Fig. 4. Knowledge database

The language allows the definition of type hierarchies, hyper-entities, hyper-relations, and rules.

This way, it is possible to define the knowledge graph on different abstraction levels. The traceability between the abstraction levels is performed by the built-in reasoning mechanism.

B. Deductive reasoning

The knowledge graph accepts those observations which comply with the operation of the system represented by the knowledge graph. One of our research question is the following: How should we handle data that violates the operation of the system represented by the knowledge graph?

Violation can indicate different behaviors:

1) the digital twin model does not fit to the real system;
2) it indicates a faulty operation in the real system;
3) the inputs are noisy.

The identification of the violation requires further analysis involving domain experts and algorithmic mechanisms.

GRAKN.AI provides user-defined rules to support deductive reasoning. Rules look for a given pattern in the dataset and when found, create the given queryable relation. The rule-based reasoning allows automated capture and evolution of patterns within the knowledge graph.

VI. Summary and further research

Qualitative reasoning and knowledge graph management of the system models provide an abstract semantic framework for information fusion from different sources and automated model extraction. They define the operational modes of the system and makes it possible to verify the ranges by discrete value representation.

Deductive reasoning checks the compliance and completeness of the observations and their respective evolving models.

Further research is needed to generalize the models with respect to the observations. This way a general hypothesis can be constructed that is generally valid in similar operational modes. Also, if the hypothesis is proven to be valid, it will be reusable.

References

[1] A. Bondavalli, S. Bouchenak, and H. Kopetz, *Cyber-Physical Systems of Systems: Foundations–A Conceptual Model and Some Derivations: the AMADEOS Legacy*. Springer, 2016, vol. 10099.

[2] "ISO/IEC/IEEE42010 Systems and software engineering – Architecture description," International Organization for Standardization, Standard, 2011.

[3] K. D. Forbus, "Qualitative modeling," *Foundations of Artificial Intelligence*, vol. 3, pp. 361–393, 2008.

[4] J. Pearl, *Causality: models, reasoning and inference*. Springer, 2000, vol. 29.

[5] J. Pearl and D. Mackenzie, *The book of why: the new science of cause and effect*. Basic Books, 2018.

[6] S. Friedenthal, A. Moore, and R. Steiner, *A Practical Guide to SysML: Systems Modeling Language*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008.

[7] A. Pataricza, L. Gönczy, A. Kövi, and Z. Szatmári, "A methodology for standards-driven metamodel fusion," in *Proceedings of the First International Conference on Model and Data Engineering*, ser. MEDI'11. Berlin, Heidelberg: Springer-Verlag, 2011, p. 270–277.

[8] "Grakn Labs Ltd: GRAKN.AI." https://grakn.ai.

Heart period is defined by P-waves in ECG

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

Abstract—**Heart rate variability (HRV) is a widely used measure to assess emotional arousal and stress level. It measures the variation in the duration of heart cycles. If HRV is determined based on the ECG signal, the duration of heart cycles is conventionally calculated as the time difference between successive R-peaks. However, the heart cycle begins with atrial depolarization, therefore, the onset of the P-wave is a physiologically more appropriate fiducial point to define successive heart cycles. This paper investigates the effect of using the onset of P-waves instead of R-peaks on HRV calculation. Measurements containing ECG signals recorded in Einthoven II lead and one measurement containing simultaneously recorded intracardiac electrograms and surface ECG signals were used. Our results suggest that the classification of successive heart cycle length differences is different depending on whether the onset of P-waves or R-peaks are used as fiducial points.**

Keywords—heart period; heart rate variability; electrocardiogram; intracardiac electrogram; P-wave delineation

I. Introduction

Blood pressure measurement is one of the most commonly used daily procedures in medical examinations and home health monitoring to assess the state of the cardiovascular system. However, the accuracy of the measurement can be influenced by many physiological and external factors [1] [2]. Stress level of the examined person can have a large impact on the accuracy of blood pressure measurement results and may induce incorrect medical conclusions if high stress level remains undetected [3]. Heart rate variability (HRV) is a widely used measure to assess momentary stress level of the tested person [4]. The calculation of HRV is based on the measurement of heart periods (the duration of heart cycles), also designated as beat-to-beat intervals. Heart periods can be measured in different ways. One of the most commonly used methods is to define heart periods as the time difference between successive R-peaks in the ECG signal. However, the heart cycle begins with atrial depolarization, while the R-peak corresponds to ventricular depolarization. Accurate measurement of heart periods should be based on precise detection of the initiation of atrial activity [5]. The P-wave corresponds to atrial depolarization in the ECG signal, however, accurate detection of the onset of P-waves is a challenging task, especially when the amplitude of P-waves is small. In this paper, we calculate HRV values using the onset of P-waves as fiducial points for recordings with high signal-to-noise ratio (SNR) and compare these results to HRV values calculated using R-peaks as fiducial points. Besides surface ECG signals, we analyze a measurement where intracardiac electrogram was also recorded.

II. Materials and Methods

A. *Detecting the Onset of Atrial Activity in the Intracardiac Signal*

Validating the detection of the onset of P-waves can be difficult, because there is no universally accepted rule for the onset and offset of the P-wave [6], moreover, in annotated databases like the QT database [7], manual annotations by experts may be inaccurate in some cases. For validation purposes, we used a clinical recording, where 12-channel surface ECG and intracardiac electrogram (EGM) were measured simultaneously. The intracardiac signals were recorded by a 4-electrode catheter. The bipolar signal of the electrode pair, closest to the sinoatrial node was used to locate the onset of atrial activity. Signals were sampled with 1 kHz sampling rate.

For the detection of the onset of atrial activity in the intracardiac signal, we used the algorithm described by Schilling [8] which is based on the non-linear energy operator (NLEO). The NLEO is a measure for the energy of a discrete-time signal. It is proportional to the squared amplitude as well as squared frequency of the given signal. Application of the NLEO to the EGM followed by filtering and thresholding can be used to analyze atrial activity.

B. *Detecting the Onset of P-waves in the Surface ECG Signal*

In the clinical recording, 12-channel ECG signals were recorded in parallel with the intracardiac signals. Moreover, a measurement series was conducted in laboratory environment, where only ECG in Einthoven II lead was recorded. One healthy senior adult and one healthy young adult participated in the measurement series. Healthy adults had normal ECG with no arrhythmia. 5 measurements were recorded for both tested persons. The recording length was between 100 and 120 seconds. Signals were sampled with 1 kHz sampling rate. The onset of P-waves was detected using the algorithm described by Martínez et al. [9]. The algorithm is based on wavelet transformation of the ECG signal with different scales. For the transformation, a quadratic spline wavelet is used. First, the QRS complex is located. After that, the P-wave is located using thresholds based on the root mean square of the transformed signal. The peak of the P-wave is also detected. In this study, the peak of the P-wave was defined as the local maximum between the P-wave onset and the Q-wave in the corresponding heart cycle.

C. Detecting R-peaks in the Surface ECG Signal

Localization of R-peaks was carried out in two steps as described and evaluated in a previous study [10]. The first step is the designation of the QRS complex with any of the usual techniques. In this study, the QRS complex was located as part of the P-wave onset detection. In the second step, the original signal is re-filtered (independently of the filtering in the first step) with two notch filters at 50 Hz and 100 Hz (4th order Butterworth), and a low-pass filter at 120 Hz (3rd order Butterworth) and the maximum value is searched for within the QRS complex. We chose the described method because it showed very high accuracy for simulated noisy ECG signals.

D. Measurements for Experimentally Induced Physical Stress

For the analysis of the effect of stress on HRV, data were also analyzed from measurements where short-term physical stress was induced for the tested persons by running 1 floor downstairs then 1 floor upstairs. One healthy senior adult and one healthy young adult participated in the measurement. Data were recorded directly before and immediately after physical stress. The recording length was between 100 and 120 seconds. Signals were sampled with 1 kHz sampling rate.

E. Characterizing HRV in Short Recordings

HRV contains dominant frequency components between 0.0033 - 0.4 Hz [11]. Therefore, frequency domain analysis of HRV is not appropriate for short recordings (1-2 minutes) typically applicable before or during blood pressure measurement. For short recordings, time domain analysis is more appropriate. In the present study we used the pNN0_20, pNN20_50 and pNN50 parameters to characterize HRV. pNN0_20 is the ratio of Differences in Subsequent Heart Periods (DSHP) that lie between 0 and 20 ms compared to the total number of DSHP. pNN20_50 stands for the same ratio but for DSHP that lie between 20 and 50 ms. pNN50 designates the ratio for DSHP greater than 50 ms. In a previous study, these parameters reflected changes in stress level in situations, where the widely used pNN50 alone indicated no or only negligible changes in stress level [12].

III. Results

A. Analyzing the Effect of Fiducial Point Designation Using ECG and EGM Signals

The effect of fiducial point designation was analyzed using the clinical measurement where 12-channel surface ECG and intracardiac EGM were measured simultaneously. For the analysis, Einthoven II lead was selected from the ECG, because it was also available in other recordings. From the intracardiac recording, the bipolar signal of the electrode pair, closest to the sinoatrial node was used. Figure 1 shows a P-wave in the ECG signal with the detected P-wave onset and the point corresponding to the time point of the onset of atrial activity in the EGM signal.

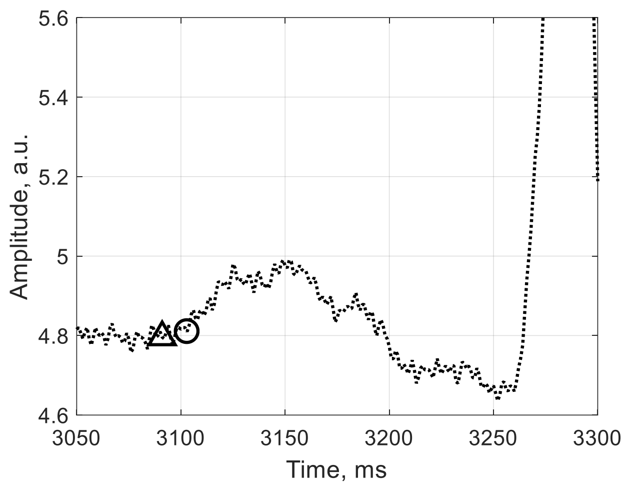


Fig. 1. A P-wave in the ECG signal (Einthoven II lead) with the P-wave onset point (circle) detected by the method described in chapter II.B and the point corresponding to the time point of the onset of atrial activity in the EGM signal (triangle) detected by the method described in chapter II.A.

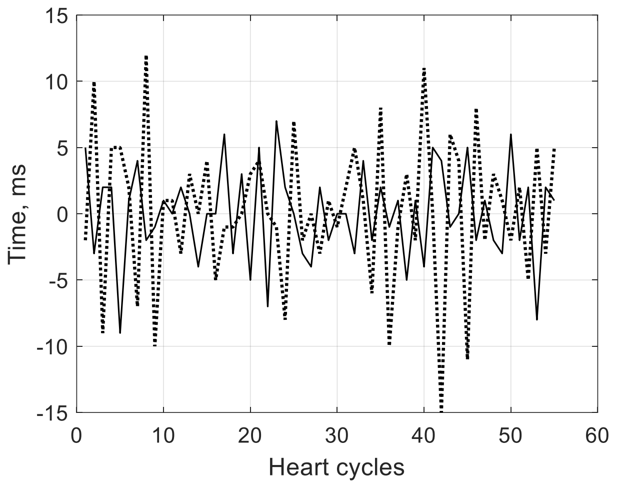


Fig. 2. The differences between heart periods calculated based on tRR, tPP and tOnOn. Solid line: tRR-tPP; Dotted line: tRR-tOnOn.

Heart periods were calculated based on R-peaks from the ECG (tRR), the onset of P-waves from the ECG (tPP) and the onset of atrial activity from the EGM signal (tOnOn). Figure 2 shows the differences between calculated heart periods. Table I summarizes the pNN0_20, pNN20_50 and pNN50 values calculated using three different fiducial point definitions. Note that the length of the recording was approximately 50 seconds (55 heart cycles), so identical values in the cells of the table are not improbable (e.g. pNN0_20 = 30 % means that 16 of 54 DSHP lie between 0 and 20 ms).

The effect of fiducial point designation was also analyzed in recordings, where only ECG in Einthoven II lead was recorded. Recordings from one healthy senior adult (HSA) and one healthy young adult (HYA) were used. The difference between heart periods calculated based on R-peaks and the onset of P-waves (tRR-tPP) for one recording of the senior adult is plotted in Figure 3.

TABLE I. pNN0_20, pNN20_50 AND pNN50 VALUES CALCULATED BASED ON DIFFERENT FIDUCIAL POINTS

	pNN0_20 (%)	pNN20_50 (%)	pNN50 (%)
tRR	32	46	22
tPP	30	46	24
tOnOn	30	40	30

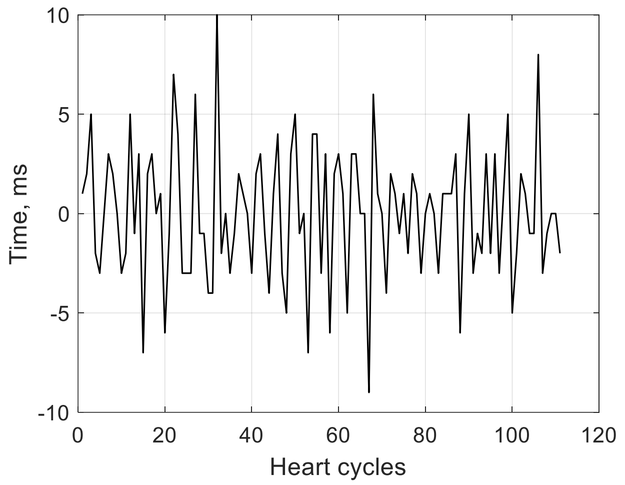


Fig. 3. The difference between heart periods calculated based on R-peaks and the onset of P-waves (tRR-tPP) for one recording of the senior adult.

The average difference in pNN0_20, pNN20_50 and pNN50 values calculated based on R-peaks and P-wave onset points was also calculated. The results are shown in Table II.

TABLE II. AVERAGE DIFFERENCE IN pNN0_20, pNN20_50 AND pNN50 VALUES CALCULATED BASED ON R-PEAKS AND P-WAVE ONSET POINTS

	Diff(pNN0_20) (%)	Diff(pNN20_50) (%)	Diff(pNN50) (%)
HSA	2	2	0
HYA	2	1	1

B. The Effect of Physical Stress on HRV

The pNN0_20, pNN20_50 and pNN50 values were calculated for measurements recorded before and after short physical stress was induced for the tested person. Values were calculated based on both R-peaks and the onset of P-waves. Table III shows the calculated values for both conditions, before stress (Pre) and after stress (Post) based on tRR and tPP, for the healthy senior adult (HSA) and for the healthy young adult (HYA).

In order to investigate the change in the conduction time through the atrioventricular node during regeneration after physical stress, the P-peak-R-peak interval was also calculated. We used the interval between peaks instead of the commonly used P-R interval because the detection of peaks is more robust than the detection of onset points. Figure 4 shows the calculated intervals for both tested persons.

TABLE III. pNN0_20, pNN20_50 AND pNN50 VALUES CALCULATED BEFORE AND AFTER SHORT PHYSICAL STRESS

	pNN0_20 (%)		pNN20_50 (%)		pNN50 (%)	
	Pre	**Post**	**Pre**	**Post**	**Pre**	**Post**
tRR, HSA	56	36	43	48	1	16
tPP, HSA	49	30	50	44	1	26
tRR, HYA	10	8	27	16	63	76
tPP, HYA	10	8	30	14	60	78

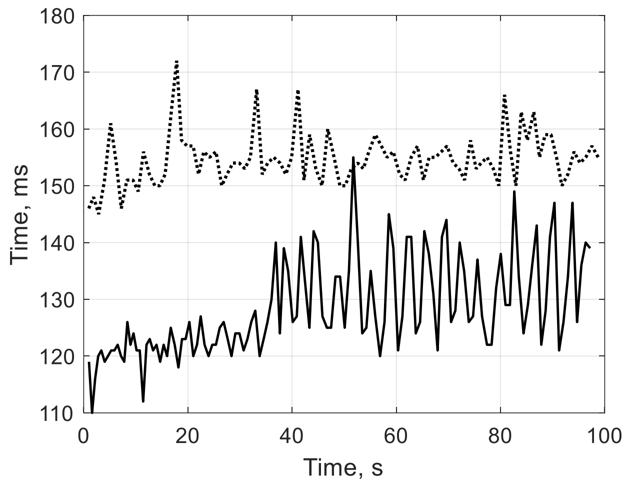


Fig. 4. P-peak-R-peak intevals after short physical stress for the healthy senior adult (solid line) and the healthy young adult (dotted line).

IV. DISCUSSION

The difference in the pNN0_20, pNN20_50 and pNN50 values is less than 2 % if R-peaks and P-wave onsets are compared in the clinical recording. However, if R-peaks in the ECG and the onset of atrial activity in the EGM are compared, the difference is more that 5 % for the pNN20_50 and pNN50 parameters. In the previous study [12], pNN0_20, pNN20_50 and pNN50 values were determined in different physical and psychical conditions. According to the results in [12], 5 % difference can mask the change in stress level between certain conditions.

Recordings from the measurement series, where only ECG in Einthoven II lead was recorded yielded similar results to the clinical recording with respect to the pNN0_20, pNN20_50 and pNN50 differences between R-peak- and P-wave onset-based calculations. The very small difference in pNN50 of the healthy senior adult is in accordance with the fact, that the number of DSHP that exceed 50 ms can be very small for senior adults. It can be even zero for a short recording.

Physical stress has different effect on the pNN0_20, pNN20_50 and pNN50 parameters. For pNN50, physical stress increased values by more than 10 % for both tested persons. The pNN0_20 decreased for both persons, but for the young adult, the amount of decrease is less than 2 %. The pNN20_50 decreased as a result of physical stress by more than 10 % for the young adult and decreased or increased for the senior adult depending on whether R-peaks or P-wave onsets were used for the calculation. The effect of fiducial point designation resulted in differences smaller than 5 % for all parameters in case of the young adult. For the senior adult, differences larger than 5 % appeared for both conditions in the pNN0_20 parameter, before physical stress in the pNN20_50 parameter and after physical stress in the pNN50 parameter. Moreover, the effect of physical stress on the pNN20_50 parameter is an increase if the calculation is based on R-peaks and a decrease if the calculation is based on the onset of P-waves. This result demonstrates that the effect of fiducial point designation can mask the change in stress level between different conditions.

The P-peak-R-peak intervals after short physical stress show an upward trend for the healthy senior adult, with more than 40 ms difference between the shortest and longest interval in the recording. For the healthy young adult, the upward trend can be observed only in the first 20 seconds but the difference between the shortest and longest interval is more than 25 ms. This result suggests that the conduction time through the atrioventricular node during regeneration after physical stress can change significantly in time. Thus, the effect of fiducial point designation on the calculated heart periods is not stable for a person-specific time interval after physical stress.

Interpretation of the results requires consideration of the accuracy of methods used to detect characteristic points in the ECG signal. The accuracy of the method we used for R-peak detection was assessed in [10]. The mean absolute error and standard deviation for a simulated noisy ECG signal were below 1 ms. Although the method was not evaluated on standard databases, we can expect similar results for real recordings because the SNR of the simulated signal was lower than that of most real ECG signals. The accuracy of the method we used for the detection of the onset of P-waves was assessed in [9]. The reported mean and standard deviation of the error of P-wave onset detection is 2.0 ± 14.8 ms for the QT database [7] and -4.9 ± 5.4 ms for the CSE database [13]. These error values are comparable to the effect of fiducial point designation on the heart period calculation (see Figure 3). Therefore, in case of recordings, where no intracardiac signal is available, heart period values based on the onset of P-waves must be handled carefully. R-peaks can be designated more accurately than P-wave onsets, however, the onset of the P-wave is physiologically more appropriate to define heart cycles. Further measurements and cooperation with medical experts can help to define HRV metrics using the information in both tRR and tPP for more accurate assessment of actual stress level.

V. Conclusion

Stress level of the examined person can have a large impact on the accuracy of blood pressure measurement. In this paper we investigated the effect of fiducial point designation on the calculation of HRV. The pNN0_20, pNN20_50 and the pNN50 parameters were investigated. Our results show that using the onset of P-waves in the surface ECG instead of R-peaks can lead to more than 5 % difference in the calculated values and may result in significant differences in stress level assessment. However, the inaccuracy of existing methods for the delineation of P-waves is comparable to the effect of fiducial point designation on heart period calculation. Further research work is needed to improve the accuracy of methods to detect characteristic points in the ECG signal and to assess actual stress level.

Acknowledgment

The research has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013).

References

[1] U. Tholl, K. Forstner, M. Anlauf "Measuring blood pressure: pitfalls and recommendations", Nephrology Dialysis Transplantation, vol. 19, no. 4, 2004, pp. 766-770.

[2] Á. Jobbágy, P. Csordás, A. Mersich "Blood Pressure Measurement: Assessment of a Variable Quantity", Zdravniški Vestnik-Slovenian Medical Journal, vol. 80, no. 4, 2011, pp. 316-324.

[3] Y. Kawano, T. Horio, T. Matayoshi, K. Kamide "Masked Hypertension: Subtypes and Target Organ Damage", Clinical and Experimental Hypertension, vol. 30, no. 3-4, 2008, pp. 289-296.

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

[5] G. D. Clifford, et al. "Advanced methods and tools for ECG data analysis", Artech house, Boston, 2006, p. 72.

[6] A. Martínez, R. Alcaraz, J. J. Rieta "Application of the phasor transform for automatic delineation of single-lead ECG fiducial points", Physiological measurement, vol. 31, no. 11, 2010, pp. 1467-1485.

[7] P. Laguna, R. G. Mark, A. Goldberg, G. B. Moody "A database for evaluation of algorithms for measurement of QT and other waveform intervals in the ECG", Computers in Cardiology, 1997, pp. 673-676.

[8] C. Schilling "Analysis of atrial electrograms", PhD thesis, Karlsruhe Institute of Technology (KIT), Karlsruhe, 2012, pp. 83-95.

[9] J. P. Martínez, R. Almeida, S. Olmos, A. P. Rocha, P. Laguna "A Wavelet-Based ECG Delineator: Evaluation on Standard Databases", IEEE Transactions on Biomedical Engineering, vol. 51, no. 4, 2004, pp. 570-581.

[10] P. Nagy, Á. Jobbágy "Accurate Calculation of Heart Period and Pulse Wave Transit Time", In: P. de Carvalho, N. Neves, J. Henriques (Eds) XV Mediterranean Conference on Medical and Biological Engineering and Computing – MEDICON 2019, IFMBE Proceedings, vol. 76 Springer, Cham, 2019, pp. 267-275.

[11] F. Shaffer, J. P. Ginsberg "An Overview of Heart Rate Variability Metrics and Norms", Frontiers in Public Health, vol. 5, no. 258, 2017.

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

[13] J. L. Willems et al., "A reference data base for multilead electrocardiographic computer measurement programs", Journal of the American College of Cardiology, vol. 10, no. 6, 1987, pp. 1313-1321.

Analysis of Ethereum Vulnerabilities and Further Steps

Mirko Staderini, Caterina Palli
Department of Mathematics and Informatics
University of Florence
Florence, Italy
mirko.staderini@unifi.it, caterina.palli@stud.unifi.it

***Abstract*— Blockchain technology is having an ever-increasing impact on distributed applications domain, since the adoption of Blockchain 2.0 led to the spread of smart contracts. In such context, Ethereum is the framework with the highest diffusion in terms of smart contract's development, with a consequent rise of code vulnerabilities exploitations, some of which causing bad financial losses. This work focuses on the issues of Ethereum smart contracts implementation by analyzing known vulnerabilities and gives an overview to further perform a comparison among existing static tools for vulnerability detection. This analysis aims to select the less detected vulnerabilities that need deeper investigations to reduce their impact.**

***Keywords*— smart contracts, Ethereum, Solidity, vulnerabilities, tools.**

I. Introduction

In the last years, Blockchain 2.0 and smart contracts exhibited the potential to have a disruptive impact in transforming some important industrial areas (e.g. medical-care, supply chain), due to the capability of automatically executing computerized transactions. Several platforms had a rapid and considerable diffusion. Among these, Ethereum is the most used framework to develop smart contracts and Dapps (decentralized applications), having a really active community and counting the highest number of deployed smart contract, developed in the language Solidity. Security issues related to the development of contracts in Solidity are particularly severe, considering that a smart contract, once deployed, cannot be patched: this can lead to funds stealing, and even to bad financial losses as happened in some well-known attacks [14], [15].

Therefore, in order to have a global view of the topic, we focused on studying code-related vulnerabilities, in particular examining several papers that analyze vulnerabilities, common patterns and countermeasures [1], [2], [3], [4], exploits [14], [15] and some analysis tools for vulnerability detection including [6], [7], [8].

Considering previous studies we noticed a lack in agreement in the number of vulnerabilities and in their categorization; the missing agreement could lead to user-confusion and to vulnerabilities proliferation as well as a difficulty for researchers to compare weaknesses with different platforms. Moreover, analysis tools for vulnerability detection that have been developed provide only a partial discovery of the comprehensive known vulnerabilities, and we noticed a lack in researches on benchmarking the existing tools, comparing performances and results. Our aim is to provide a Solidity-specific vulnerabilities analysis, categorizing each vulnerability with classes that are based on a general-purpose (not language-specific) classification, since we believe that this work may help software developers in limiting weaknesses explosion and researcher in comparing other platforms vulnerabilities. In order to do that, we first studied the Common Weakness Enumerator (CWE) [13] classification that provided us with an abstract point of view to systematize the vulnerabilities. After this work, we aim at performing a benchmark on the existing static analyzers for Solidity vulnerability detection, since we believe that it may be useful to understand firstly which tools are the most effective and performing, and secondly which vulnerabilities are the most difficult to be detected, and thus require a deeper analysis.

Our Contribution:

- A comprehensive list of Ethereum smart contracts vulnerabilities, grouped following CWE categories.
- A short description of further steps aiming at performing a benchmark among static analysers for vulnerability detection.

The paper is organized as follows. Section II shortly describes the CWE hierarchical representation. Section III presents a systematization of the vulnerabilities. Section IV describes further steps of our work and Section V concludes the work.

II. CWE Hierarchical Representation

The CWE (Common Weakness Enumeration) is a list of community-developed security software weaknesses, used in multiple contexts (e.g. industries, academia, and standards). Considering that a vulnerability is a weakness that has been exploited (according to the Common Attack Pattern Enumeration and classification - CAPEC) [17], we decided to use CWE categories for our classification, abstracting from the language Solidity. CWE is organized in three different representations, corresponding to three different points of view; among these, we chose the one that is focused on software behaviours. Each representation is structured in a hierarchical way, that is a tree where the root contains the most generic category, and the leaves contain the most specific ones (mostly language-dependent). For our purposes, we chose the categories that allowed us to obtain groups of vulnerabilities having common characteristics, abstracting from Solidity.

III. Vulnerabilities Analysis

In our work, we studied research publications of the last years, using keywords as 'Vulnerabilities, 'Solidity', 'Known attacks', 'Smart contract' 'Survey', 'Common patterns'. We also consulted the official Solidity documentation [16], the Smart Contract Weakness Classification (SWC) [12], the National Vulnerable Database (NVD) [18] and web pages related to recommendations, best practices and known attacks. In order to define our set of vulnerabilities, at first we built a comprehensive list of weaknesses; then we considered only the vulnerabilities (weaknesses that have been exploited at least once), and finally we choose the ones that are mentioned at least twice crossing all the analyzed resources.

Below, we shortly describe the CWE classes used to systematize the collected vulnerabilities; moreover, for each of them, we indicate the most referenced vulnerabilities belonging to it, providing a short description and a reason for the chosen categorization. Please refer to TABLE I for a comprehensive list of vulnerabilities, each associated to its CWE class, to a short description and to the main reference.

TABLE I. VULNERABILITIES AND THEIR CLASSES

Class	**Vulnerabilities Features and References**	
	Vulnerabilities	***Features and References***
CWE-20	Ether Lost in Transfer	If the address of money transfer is orphan, the money will be lost [2].
CWE-20	Short Addresses	If the address length is not checked in a contract invocation, an attacker could gain funds [1], [11].
CWE-20	Requirement Violation	It results from a violation of an external function input validation [11].
CWE-20	Malicious Libraries	Caused by the use of a library from untrusted sources [8].
CWE-284	Tx.origin	When a contract uses tx.origin for authorization, it can be compromised by a phishing attack [12].
CWE-284	Visibility of Exposed Functions	When a function is defined with a wrong access policy, an attacker could execute it arbitrarily [1].
CWE-284	Unprotected Selfdestruct	The vulnerability results from the use of *selfdestruct* primitive, without a proper authorization check [12].
CWE-284	Unprotected Ether Withdrawal	Due to a missing or extraneous access control, an attacker can drain funds from a contract [12].
CWE-330	Bad Randomness	The use of variables as a seed to generate pseudo-random values may allow an attacker to nullify the randomness [2], [9], [12].
CWE-400	DoS costly Patterns and Loops	When the code contains unbounded operations, the gas needed to complete an execution may exceed the gas limit resources [6].
CWE-400	Call stack Depth Value	An attacker may force to exceed the stack limit size (1024 frames): if the exception is not correctly managed, the attack may succeed [16].
CWE-400	Gasless send	An out-of-gas exception occurs when an operation execution exceeds the expected amount of gas [2].
CWE-400	Under-priced Opcode	Excessive resources consumption at a low price could lead to resources exhaustion [1].
CWE-682	Integer Overflow/Underflow	Missing/wrong/extraneous control in mathematical operation could lead to an overflow/underflow [1].
CWE-691	Unpredictable State	If the order in which transactions are executed is crucial for a contract, it may reach an unpredictable state [1].
CWE-691	Reentrancy	When a callee calls the calling function back before its completion, an attacker may drain funds from a contract [9].
CWE-691	Freezing Ethers	It happens when, due to the wrong control flow, it is no more possible to transfer funds [6].
CWE-703	Unchecked Call Return Values	It is caused by a missing check in the return value results [8].
CWE-703	Unchecked Send	It is caused by a missing check in the return value of *send* primitive [8].
CWE-703	Exceptions Disorder	It is due to the inconsistency of exceptions propagation in Solidity [2].
CWE-668	Secrecy Failure	Using a variable (even private) for secret information may allow an attacker to discover them [2].
CWE-668	Lack of Transactional Privacy	Normally, the privacy of data transactions is not guaranteed [4].
CWE-668	Blockhash Usage	The *blockhash* global variable value should not be used in critical operations, because a malicious miner can manipulate it [5].
CWE-668	Timestamp Dependency	A *timestamp* global variable value should not be used in a critical operation, because a malicious miner can manipulate it [5].
CWE-345	Missing Protection against Signature Attack	In case of insufficient signature verification, an attacker could perform a replay attack [12].
CWE-345	Typecast	An attacker can execute arbitrary code, simply passing a malicious contract as a parameter in a contract function call that calls it back [2].
CWE-669	Call to the Unknown	Solidity primitives may invoke the fallback function of the callee, allowing some external portion of code to be executed [2].
CWE-669	Delegatecall to Untrusted Calle	The *delegatecall* primitive executes the code in the context of the called contract [7].
CWE-669	DoS by External Contract	External calls that depend on conditional statements may lead to a DoS situations [8].

A. CWE-20 Improper Input Validation

An attacker can be able to execute arbitrary operations and steal funds in case of an improper input validation. The exploitation of this group of vulnerabilities may lead to the excessive consumption of resources in the availability scope, to the reading of confidential data or to the alteration of the flow control (including arbitrary code execution).

1) The vulnerability *Short Addresses* is due to a missing check of an address validity. An attacker can perform a call with an address that is shorter than expected, causing the left shift of the following function parameter: if this represents an amount of funds, it may allow the adversary to gain money improperly [1], [11].

2) Other vulnerabilities that belong to this category are *Ether Lost in Transfer* [2], *Requirement Violation* [11] and *Malicious Libraries* [8].

B. CWE-284 Improper Access Control

There is a missed or improper restriction in accessing a resource. Improper authentication, incorrect and missing authorization are some of the mechanisms that characterize this class. An attack may lead to read or modify sensitive data, to gain unintended privileges or to allow an attacker to execute arbitrary code.

1) An improper definition of the access control of a function may lead to a *Visibility of Exposed Functions:* an unintended adversary may be able to execute the function for arbitrary reasons. In practice, the use of a wrong modifier may allow unauthorized execution, with various possible effects [1].

2) Due to an Improper Access Control a *Tx.origin* [12], an *Unprotected Selfestruct* [12] and an *Unprotected Ether Withdrawal* [12] could be generated and exploited.

C. *CWE-330 Use of Insufficiently Random Values*

The software uses insufficient entropy generators when an operation depends on unpredictable numbers. This may lead to bypass protection mechanisms, allowing improper access to protected resources or restricted functionalities.

1) Considering that the execution of the bytecode is deterministic, generating pseudo-random numbers using block-based values (i.e. blockchain global variables) can cause *Bad Randomness*. Since miners control blocks, a malicious one may bias specific values; moreover, even private variables can be read easily due to the public nature of the blockchain [2], [9], [12]: thus an attacker can guess random values and manipulate events. A possible countermeasure is using external sources via oracles.

D. *CWE-400 Uncontrolled Resource Consumption*

Resources are consumed without any software control: this may lead to their exhaustion. Some consequences could be a DoS situation or an impairment of software status.

1) In case in which loops or useless code [6] are used in a Solidity smart contract, a *DoS Costly Pattern and Loops* vulnerability may occur. This may lead to a DoS situation if the gas needed to execute an operation exceeds the gas limit.

2) Other vulnerabilities that may be generated in case of resource exhaustion are the *Call-stack DepthValu*e [16] and the *Gasless send* [2].

E. *CWE-682 Incorrect Calculation*

This class includes vulnerabilities in which the software performs calculations that generate unintended results [19]. After an incorrect calculation, a program may move in an incorrect state, causing unintended resources consumption, compromising protection mechanisms and giving access to sensitive resources.

1) Arithmetic operations that are performed without any check may lead to an *Integer overflow/underflow* vulnerability: this can cause some different effects depending on the way in which the result is used (e.g. managing resources, controlling the execution flow).

F. *CWE-691 Insufficient Control Flow Management*

Unexpected computations are caused by an incorrect control of the execution flow; a common consequence is an alteration of the execution logic of the program.

1) *Rentrancy* is a vulnerability that has been exploited in the famous *The DAO Attack* [14]. When a callee calls the calling function back before its completion, this function may be executed repeatedly [9], [10], [12]. In the function is meant to execute only once, the attacker can drain all the funds of the contract. This is easily achievable due to the implementation of some Solidity primitives, mainly for sending Ethers, since the *fallback function* of the callee is executed.

2) If a software relies on the order of the execution of transactions an *Unpredictable State* [1] or a *Freezing Ether* [6] vulnerability may be generated.

G. *CWE 703 - Improper Check or Handling of Exceptional Conditions*

Exceptional conditions that happen at run-time are not properly handled; possible consequences are improper reading of application data, unexpected states or DoS situations.

1) A missing or wrong check on a Solidity function return value, mainly in case of transferring Ethers, may lead to the *Unchecked Call Return Values* vulnerability, in case of failure without raising an exception [8] and its subset Unchecked send [8].

2) The inconsistency in Solidity exceptions propagation in functions for transferring Ethers may lead to the *Exceptions Disorder* vulnerability [2].

H. *CWE-668 Exposure of Resource to Wrong Sphere*

Unintended actors can inappropriately access to resources, due to an improper resources exposition (e.g. insecure permissions).

1) Since all variables values are published onto the Ethereum blockchain, declaring a private variable does not guarantee its secrecy, generating a *Secrecy Failure* vulnerability: in fact, anyone can inspect published private values [2]. The improper exposure of transactions leads to *Lack of transactional privacy* [4].

2) When *blockhash* or *timestamp* global variables (that can be manipulated by a malicious miner) are used for critical operations, they can cause *Blockhash Usage* [5] and *Timestamp Dependency* [2] respectively.

I. *CWE-345 Insufficient Verification of Data Authenticity*

The software accepts invalid data, not sufficiently verifying their authenticity. This has various consequences, mostly related to data integrity.

1) Protection against the Signature Replay Attack is missing in operations that need a signature verification: the contract may be vulnerable having a *Missing Protection against Signature Replay Attack* [12].

2) Solidity type checker does not properly verify the correctness of contracts types; thus receiving a contract as a function argument and invoking its functions with insufficient verification leads to a *Typecast* vulnerability: an attacker may pass a malicious contract, having a function with the same name of an invoked one, achieving the execution of arbitrary code [2].

J. *CWE-669 Incorrect Resource Transfer Between Spheres*

An unintended control over the resource is allowed, due to an improper transfer/import management of a resource to/from another sphere. The most critical consequences are unexpected states or the possibility to read/modify data.

1) Using some primitives for functions invocation and for Ethers transfer may lead to unexpected behaviours, because of their adverse effect to invoke the *fallback function* of the callees. This is known as *Call to the Unknown* [2], [10].

2) The *Incorrect Resource Transfer Between Spheres* category also includes the *Delegatecall to Untrusted Callee* and the *DoS by External Contracts* vulnerabilities [8].

IV. Further steps

In this section, we summarize the steps of our future work that aims at providing a comparison among static analyzers for vulnerability detection in Solidity code. Our final goal is to make a benchmark on these tools and to discover consequently which vulnerabilities are the most difficult to detect.

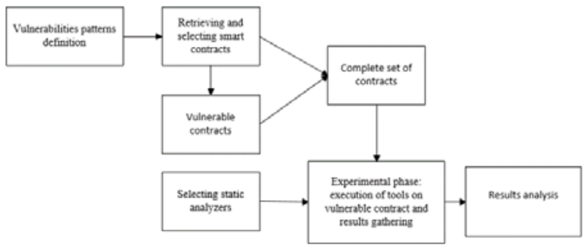

Vulnerabilities patterns definition
Retrieving and selecting smart contracts
Vulnerable contracts
Complete set of contracts
Selecting static analyzers
Experimental phase: execution of tools on vulnerable contract and results gathering
Results analysis

Fig. 1 Further steps: methodology proposal

Fig. 1 illustrates a conceptual overview of this work, highlighting the single mid-term steps that we will follow. We now provide a brief description of the whole process. Starting from the comprehensive list of categorized vulnerabilities that we provided, we should individuate the common patterns of their occurrence forms in Solidity code (e.g. missing check, wrong check, usage of unsafe constructs), in order to use them in the further steps; moreover, we should investigate actual occurrences in deployed smart contracts, such as those reported by NIST, in order to understand if some of them are more critical, in terms of detrimental effects. After the identification of the vulnerabilities occurrences, we should define an automatic procedure to inject the corresponding vulnerabilities patterns in actual smart contracts, in order to generate a set of vulnerable contracts. Therefore, it is necessary to define a set of contracts (several of them have already been retrieved from etherscan.io) to perform the injection, that should be carried out through a software mutation mechanism. By producing different combinations of vulnerabilities patterns and by injecting them into smart contracts code, we should obtain a set of vulnerable contracts that will be used to execute a set of static analyzers for vulnerability detection on them. Thus, another crucial step is the selection of a set of static tools that will be the object of our analysis: after a preliminary investigation on the state of art of static analyzers for Solidity vulnerability detection, we should individuate which among them are publicly available; starting from these, we should then obtain a final set, by selecting the most referenced ones. Moreover, in order to collect the results of the tools and analyze them in a meaningful way, we should generate a homogeneous representation of the identified values of interest. In particular, we should inspect the different tools outcomes stating the detection of each vulnerability, possibly investigating true/false positive, and true/false negative cases. The main steps of the whole process can be summarized in the following: the injection of vulnerabilities into the selected smart contracts code; the experimental phase, that is the execution of each selected tool on the obtained vulnerable contracts, alongside the results gathering in a meaningful and homogeneous way; and finally the analysis of the results. This analysis will focus on the comparison of the effectiveness and efficiency of the chosen set of tools. Moreover, this analysis will allow the identification of the vulnerabilities that are most difficult to be discovered, and that will require a deeper investigation.

V. Conclusion

Because of the spreading of Ethereum smart contracts, we focused on the analysis of software vulnerabilities related to the programming language Solidity. This work presented a systematization of such vulnerabilities, furthermore categorized using CWE classes, in order to help researchers, through an abstract view, to compare them with those in different environments. Starting from a better understanding of vulnerabilities behaviours, we proposed a roadmap defining the steps to carry out a benchmark on existing static tools for the detection of smart contracts vulnerabilities, with the aim of allowing to identify the most undetected vulnerabilities, which further require a deeper study.

References

[1] H. Chen, M. Pendleton, L. Njilla, and S. Xu, “A survey on Ethereum systems security: Vulnerabilities, Attacks and Defenses”, arXiv:1908.04507, 2019.

[2] N. Atzei, M. Bartoletti and T. Cimoli, “A survey on attacks of Ethereum smart contracts,(SoK)” in *In Principle of Security and Trust*, Springer, 2017, pp. 164-186.

[3] A. Dika and M. Nowostawski, "Security Vulnerabilities in Ethereum Smart Contracts," 2018 IEEE International Conference on Internet of Things (iThings), Halifax, NS, Canada, pp. 955-962(2018), DOI: 10.1109/Cybermatics_2018.2018.00182.

[4] A. Mense and M. Flatscher, “Security Vulnerabilities in Ethereum Smart Contracts,” in Proceedings of the 20th International Conference on Information Integration and Web-based Applications & Services. ACM, 2018, pp. 375–380.

[5] H. Hasanova, U.-J. Baek, M.-G. Shin, K. Cho, and M.-S. Kim, “A survey on blockchain cybersecurity vulnerabilities and possible countermeasures,” Int. J. Netw. Manage., vol. 29, no. 2, p. e2060, 2019.

[6] B. Jiang, Y. Liu, and W. Chan, “ContractFuzzer: Fuzzing Smart Contracts for Vulnerability Detection,” in Proceedings of the 33rd ACM/IEEE International Conference on ASE, 2018, pp. 259–269.

[7] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, “Making Smart Contracts Smarter,” in Proceedings of the 2016 ACM SIGSAC, *Conference on Computer and Communications Security*, pp. 254–269, ACM, 2016.

[8] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev E. Marchenko, and Y. Alexandrov, “SmartCheck: Static Analysis of Ethereum Smart Contracts,” in *2018 IEEE/ACM, 1st WETSEB*, pp. 9–16, IEEE, 2018.

[9] «DASP - TOP10,» Dasp, 2018. http://www.dasp.co/ (visited on 01.12.2019).

[10] “Known attacks - Ethereum Smart Contract Best Practices.” https: //consensys.github.io/smart-contract-best-practices/known_attacks/.

[11] “Solidity Security: Comprehensive list of known attack vectors and common anti-patterns”, [Online]. Available: https://blog.sigmaprime.io/solidity-security.html#re-vuln (visited on 12/12/2019).

[12] “SWC Registry,” [Online]. Available: https://smartcontractsecurity.github.io/SWC-registry (visited on 12/11/2019).

[13] "CWE View: Research Concept," [Online]. Available: https://cwe.mitre.org/data/definitions/1000.html (visited on 12/11/2019).

[14] “Understanding the DAO Attack,” [Online]. Available: https://www.coindesk.com/understanding-dao-hack-journalists

[15] “Governmental’s 1100 eth jackpot payout is stuck because it uses too much gas” [Online]. Available: https://www.reddit.com/r/ethereum/comments/4ghzhv/governmentals_1100_eth_jackpot_payout_is_stuck/ (visited on 12/11/2019).

[16] “Solidity,” [Online]. Available: https://solidity.readthedocs.io/en/latest/ (visited on 05/07/2019).

[17] “CAPEC,” [Online]. Available: https://capec.mitre.org/ (visited on 15/07/2019).

[18] “NVD,”[Online]. Available: https://nvd.nist.gov/ (visited on 12/10/2019)

Modeling and Analysis of an Industrial Communication Protocol in the Gamma Framework

Bence Graics*†, István Majzik*
*Budapest University of Technology and Economics, Department of Measurement and Information Systems
Budapest, Hungary
†MTA-BME Lendület Cyber-physical Systems Research Group
Budapest, Hungary
Email: {graics, majzik}@mit.bme.hu

Abstract—**Communication protocols are often designed on the basis of state-based models. During protocol design, the use of formal verification is indispensable, as concurrent behavior is notorious for hidden and sophisticated bugs. This paper presents a formal verification approach to verify an industrial communication protocol using the Gamma Statechart Composition Framework. Gamma is a modeling toolset for the design and analysis of reactive systems. It supports a family of modeling languages with formal semantics for the component-based definition of state-based behavior. It also supports formal verification by automatically mapping the defined models to the input formalisms of verification backends and back-annotating the results. The verification approach is presented in the context of the Orion industrial communication protocol. The verification approach supports the introduction of channel models with different message transmission characteristics and failure modes. Different execution modes of the components are also analyzed.**

I. Introduction

Communication protocols are inherently event-driven and are frequently designed using state-based models, e.g., statecharts [1]. Furthermore, communication protocols are often used in safety-critical systems where correct behavior is crucial, which makes formal modeling languages as well as sophisticated verification and validation (V&V) techniques, e.g., formal verification, necessary during the design process.

As communication protocols have multiple participants, the modeling language must support composition functionalities in addition to supporting individual component design. Also, to make formal verification feasible, the modeling language must have a formal semantics both at component and system level, defining how a standalone component is executed, and describing the execution and communication of contained components. Such a language can be supported by a modeling and analysis tool, which can facilitate the design and V&V of communication protocols.

The Gamma Statechart Composition Framework is such a tool, providing a language for composing individual statechart components (possibly created in other tools) while supporting verification and validation (V&V) capabilities. In this paper we propose a formal verification approach for communication protocols using the Gamma framework, which includes *1)* the construction of protocol participant models as well as channel models with different failure modes, *2)* the composition of protocol participant and channel models to form system models and *3)* model checking on the system models with automatic back-annotation of the results. The process is presented in the context of Orion, a master-slave communication protocol under design targeted to be used in the railway industry.

II. Gamma Statechart Composition Framework

The Gamma Statechart Composition Framework [2] is an open-source, integrated modeling toolset to support the semantically sound composition of heterogeneous statechart components. The framework reuses statechart models of third-party tools and their code generators for separate components, e.g., Yakindu[1] and MagicDraw[2], thus UML/SysML state machine models are supported. The mapping of these external models to the internal statechart representation of Gamma (Gamma Statechart Language – GSL) is supported by automatic model transformations. The framework provides the Gamma Composition Language (GCL), which supports the interconnection of components according to different composition modes based on precise semantics. Furthermore, Gamma provides code generators for deriving implementation from defined models as well as test case generators for the analysis of component interactions. Gamma also supports system-level formal verification and validation (V&V) functionalities by mapping statechart and composition models into formal automata of the UPPAAL [3] model checker. Also, the automatic back-annotation of the verification results is supported.

GCL supports three composition modes, namely *synchronous*, *cascade* and *asynchronous*, which fundamentally determine the execution of the resulting composite models. The detailed introduction of these composition modes can be found in [4], here we include a summary of their properties.

Synchronous A synchronous model represents a coherent unit consisting of strongly coupled but concurrent components, which are executed in a lock-step fashion and communicate in a synchronous manner using signals.

Cascade Cascade models are special synchronous models whose components are executed in a sequential manner. Contained components can be considered as a set of filters applied sequentially to derive an output from an input.

[1]https://www.itemis.com/en/yakindu/state-machine/
[2]https://www.nomagic.com/products/magicdraw

Asynchronous Asynchronous models represent independently running components. There is no guarantee on the execution time or the execution frequency of such components, thus, they communicate with queued (persistent) messages.

III. Modeling of the Communication Protocol

This section introduces the modeling process of our proposed verification approach in the context of Orion.

A. Protocol Participants

Orion is a master-slave communication protocol, where the establishment of a connection between two participants is always initiated by a master and the connection request is either accepted or rejected by a slave. Both the master and the slave participants were designed on the basis of statecharts in MagicDraw and have the same events (commands and messages) that can be classified into two groups:

- *Connect* and *Disconnect* events come from the environment and can be used as external commands to initiate a connection or break down an established connection. *Invalid* event is also an external event indicating an invalid status in the environment of the system.
- Events of the Orion protocol are transmitted between protocol participants and can be used to establish (*OrionConnReq*, *OrionConnResp* and *OrionConnConf*) or break down a connection (*OrionDisconnCause*), send data in established connections (*OrionAppData*) or keep established connection alive in the absence of transmittable data (*OrionKeepAlive*).

The initial state of the *master* statechart (depicted in Fig. 1) is *Closed*. Upon receiving a *Connect* event or after a specified timeout (*TReconn*: 5 seconds in the example), it goes to state *Connecting* while sending an *OrionConnReq* event to the slave. If it receives an *OrionConnResp* event within a specified time interval, it goes to state *Connected* while sending an *OrionConnConf* event to the slave. If in state *Connecting* it receives any other events, or does not receive any events in a specified time interval (*TConn*: 5 sec), it goes back to state *Closed* and sends an *OrionDisconn* event when necessary, that is, if the received event was not *OrionDisconnCause*. In state *Connected*, application specific data, or in the absence of data for a specified time interval (*TKeepAlive*: 4 sec) an *OrionKeepAlive* event are sent (child state *KeepAliveSendTimeout*). Also in state *Connected*, data as well as *OrionKeepAlive* events are received (child state *KeepAliveReceiveTimeout*). However, if any other event is received or no events are received in a specified time interval (*TInactive*: 5 sec), the master goes back to state *Closed* and sends an *OrionDisconn* event if necessary.

The *slave* statechart (see Fig. 2) is similar to the master.

The models can be automatically transformed to the GSL using the model transformers of Gamma, in which they can be validated based on statechart-related well-formedness rules [5]. According to the validators of Gamma, the presented statechart models are well-formed.

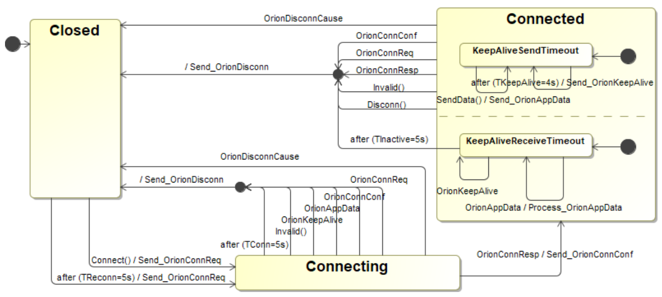


Fig. 1. The statechart model describing the behavior of the master component.

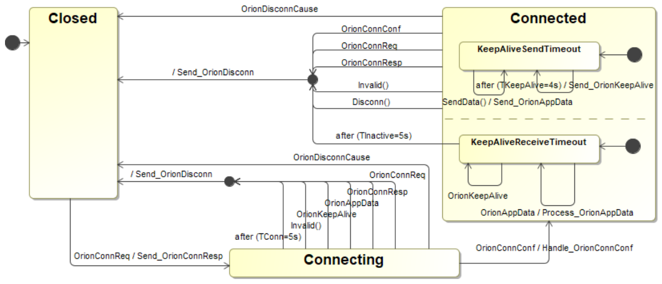


Fig. 2. The statechart model describing the behavior of the slave component.

B. Channel Models

Several failure modes of event transmission between protocol participants can be considered [6]. In this work we focus on *loss of events* and *delay of events* failure modes, as

- the *duplication of events* can be filtered using sequence numbering, this failure does not reach the protocol level,
- the *reordering of events* can be detected using sequence numbering, on protocol level this failure is mapped to the loss of these events, and
- the *alteration of event content* can be detected using integrity checking, on protocol level this failure is also mapped to the loss of these events.

Therefore, by focusing on *loss of events* and *delay of events*, we cover all relevant failure modes of [6].

In this work we defined five atomic channel models in Yakindu: one *ideal channel*, three models describing loss of events failure modes (*bursty message losing channel*, *arbitrary message losing channel* and *timed message losing channel*) and one model related to delay of events failure mode (*delay channel*). The following paragraphs present these models using graphical statechart representations. Note that these representations are simplified versions of the real models and include behavior only for a single event (*OrionConnReq*), however, additional events in the real models are handled analogously.

Fig. 3 depicts the *ideal channel* model. When it receives a certain event on its input, it forwards the event to its output, events are not lost or delayed.

Fig. 4 depicts the *bursty message losing channel* model, which models a channel that can lose a given amount (*LOST_MESSAGE_MAX*) of subsequent incoming events. It has two states, *Operating* (initial state) and *MessageLosing*. If

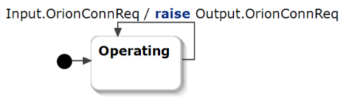
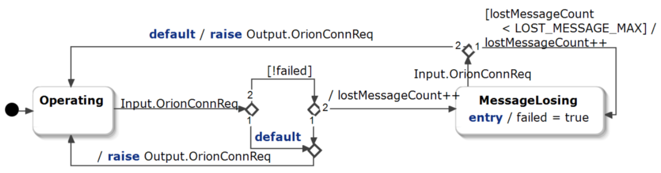


Fig. 3. The statechart model of the *ideal channel*.

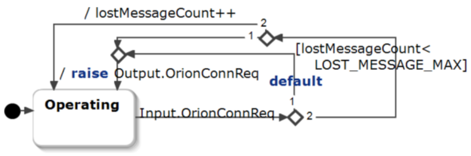
the model receives a certain event in state *Operating* it either forwards the event to its output, or (if there has been no failure before) goes to state *MessageLosing* without forwarding the event. In state *MessageLosing*, the specified amount of events are absorbed before going back to state *Operating*. Note the nondeterministic nature of this model: the loss of subsequent events can start on any incoming event.



default / raise Output.OrionConnReq
[lostMessageCount < LOST_MESSAGE_MAX] / lostMessageCount++
[!failed]
Input.OrionConnReq
Operating
Input.OrionConnReq
/ lostMessageCount++
MessageLosing
entry / failed = true
default
/ raise Output.OrionConnReq

Fig. 4. The statechart model of the *bursty message losing channel*.

Fig. 5 depicts the *arbitrary message losing channel* model. It overapproximates the behavior of the *bursty message losing channel* model, as it supports the loss of events regardless of their order, that is, a given amount of events can be lost but these losses can occur any time, the lost events do not have to be necessarily subsequent.



/ lostMessageCount++
[lostMessageCount< LOST_MESSAGE_MAX]
/ raise Output.OrionConnReq
default
Operating
Input.OrionConnReq

Fig. 5. The statechart model of the *arbitrary message losing channel*.

Fig. 6 depicts the *timed message losing channel* model, which loses messages that are received in a specified time interval. It has two states, *Operating* (initial state) and *MessageLosing*. In state *Operating*, incoming events are forwarded to the output. After a certain time (S sec), if the model has not failed before, it goes to state *MessageLosing*, where incoming events are absorbed. It goes back to state *Operating* after a specified time (E sec) and remains there.

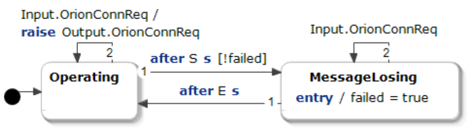
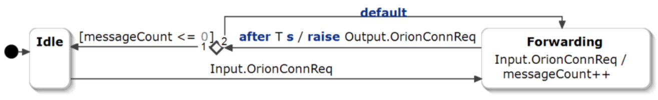


Fig. 6. The statechart model of the *timed message losing channel*.

Fig. 7 depicts the *delay channel* model, which delays the transmission of events with a given time. In this model each event type is handled in an orthogonal region. A region has two states, *Idle* (initial state) where there is no event in the channel, and *Forwarding* where the transmission of events of a certain type is delayed. If an event is received in state *Idle*, the model goes to state *Forwarding* where additional incoming events are queued (variable *messageCount*). After a specified time (T sec), the delayed event is forwarded. If there is no additional queued event, the model goes to state *Idle*, otherwise, it goes back to state *Forwarding*.



default
Idle
[messageCount <= 0]
after T s / raise Output.OrionConnReq
Forwarding
Input.OrionConnReq / messageCount++
Input.OrionConnReq

Fig. 7. The statechart model of the *delay channel*.

C. System Models

We analyzed the behavior of the Orion protocol considering different channel failure modes and different execution modes of the participants. Therefore, for each channel model we defined cascade, synchronous and asynchronous composite Gamma models, which differ only in the execution mode, the components and their connections are the same. In this work we focused on the time-driven behavior and the events of the Orion protocol in the master and slave components and did not consider the external events and commands that may directly close the connection.

Fig. 8 describes the GCL model the variations of which were used with different channel models and execution modes. It consists of a master component, a slave component, and two channel components that connect the output and input ports of the protocol participants. The concrete models differ only in the first keyword that can be either *sync*, *cascade* or *async*. All in all, fifteen composite system models were defined, five (as there are five channel models) for each composition mode. In the asynchronous composite models message queues with capacity 2 were used.

```
[sync | cascade | async] OrionSystem [] {
  // Declaration of components
  component master : OrionMaster
  component m2S : Channel
  component slave : OrionSlave
  component s2M : Channel
  // Connecting component ports via channels
  channel [master.SendOrion] —o)— [m2S.Input]
  channel [m2S.Output] —o)— [slave.ReceiveOrion]
  channel [slave.SendOrion] —o)— [s2M.Input]
  channel [s2M.Output] —o)— [master.ReceiveOrion]
}
```

Fig. 8. The GCL model of protocol participants and channel models.

IV. Analysis of the Communication Protocol

We analyzed liveness properties of the system models introduced in Section III-C, that is, the reachability of system states using different channel models and execution modes. The analyzed properties (formalized in CTL) are the following.

P_1 The system *can reach* a state in which both the master and the slave are in state *Connected*: `EF master.Connected && slave.Connected`.

P_2 The system *must eventually reach* a state in which both the master and the slave are in state *Connected*: `AF master.Connected && slave.Connected`.

P_1 means the models do not contain fundamental faults. P_2, as a robustness property means the protocol is always able to recover despite the specified failure mode of the channel.

According to the verification executed in Gamma, P_1 *holds* in the case of every system model introduced in Section III-C. The analysis results for P_2 are shown in the following sections.

A. Synchronous Composition Mode

As the protocol has real-time timeouts, the fulfillment of the property depends on the execution frequency (indicated by f) of the system components in the case of each channel model.

In the case of the *ideal channel* model, f has to be higher than $4/TConn$ for the property to hold due to the *lock-step* execution mode of the components: event transmission between the master and the slave is delayed as events are transmitted through separate channel components (the value in the nominator refers to the number of components in the composite model). The timeout in state *Connecting* (both for the master and the slave) is *TConn* sec. Therefore, event *OrionConnResp* in response to *OrionConnReq* in the case of the master, and event *OrionConnConf* in response to *OrionConnReq* in the case of the slave have to be received in lesser time to enable the reaching of state *Connected*.

The property holds in cases of both the *bursty* and *arbitrary message losing channel* models for all *LOST_MESSAGE_MAX* values between 1 and 9, if f is higher than $4/TConn$.

In the case of the *timed message losing channel* model, the property was checked for the following *S* and *E* values: 4 and 9, 4 and 14, 4 and 19, 9 and 14, 9 and 19, 14 and 19 (so that multiples of parameters *TReconn* and *TConn* in the master and slave models fall into these intervals). If f is higher than $4/TConn$, the property holds.

In the case of the *delay message losing channel* model, f, the T parameter of the channel and the *TConn* parameter in the master and slave have to satisfy the following constraint: $2/f + T < TConn/2$. If this constraint is not satisfied, an execution of the components can exist where the master and slave get desynchronized due to the late arrival of messages and the *Connected* states are never reached at the same time.

B. Cascade Composition Mode

The cascade composition mode defines a *sequential* execution semantics. Similarly to the synchronous composition mode, the fulfillment of the analyzed property depends on the execution frequency of components. The property can be fulfilled in the case of every channel model, and in this composition mode the execution frequencies can be lower.

In the case of the *ideal channel*, *bursty-*, *arbitrary-* and *timed message losing channel* models, f has to be higher than $1/TConn$ for the property to hold, as components are executed in the following order: *master*, *m2S*, *slave*, *s2M*. Therefore, events from the master to the slave and from the slave to the master can be transmitted in a single execution cycle.

In the case of the *delay message losing channel* model, f, the T and the *TConn* parameters have to satisfy the following constraint: $1/f + T < TConn/2$.

C. Asynchronous Composition Mode

In the case of the asynchronous composition mode, there is no guarantee on the execution frequency of the components of the composite model. Therefore, in the case of any channel model, it is possible to delay the execution of either the master or slave component in state *Connecting* until the timeout is reached (*TConn* sec). Thus, the property *does not hold* regardless of the defined channel models. To examine this problem, we introduced constraints on the execution frequency of the components in the formal models. We observed that if the execution frequencies of the components are in the order $f(master) < f(m2S) < f(slave) < f(s2M)$, and are higher than $4/TConn$ in the cases of the first four channel models, or the $2/f + T < TConn/2$ constraint holds in the case of the *delay* channel model, the property holds.

V. Conclusion

We proposed a verification approach for communication protocols using Gamma in the context of Orion, an industrial master-slave communication protocol. We defined multiple channel models describing loss of events and delay of events failure modes. Using model checking, we verified whether the system *1)* might eventually and *2)* must eventually reach a state in which both the master and slave are in state *Connected*.

In the future we plan to investigate the composition of channel models in a single GCL component where the operating channel model is selected by a selector component. This way, both the model construction and the analysis phases could be simplified as the number of resulting system models would decrease to three. We also aim to introduce full support for execution frequency constraints in asynchronous components.

Acknowledgment Supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013) and the ÚNKP-19-3 New National Excellence Program of the Ministry for Innovation and Technology.

References

[1] D. Harel, "Statecharts: A visual formalism for complex systems," *Sci. Comput. Program.*, vol. 8, no. 3, pp. 231–274, Jun. 1987.

[2] V. Molnár, B. Graics, A. Vörös, I. Majzik, and D. Varró, "The Gamma Statechart Composition Framework," in *40th International Conference on Software Engineering (ICSE 2018)*. Gothenburg, Sweden: ACM, 2018.

[3] G. Behrmann, A. David, K. G. Larsen, J. Håkansson, P. Pettersson, W. Yi, and M. Hendriks, "Uppaal 4.0," 2006.

[4] B. Graics and V. Molnár, "Mix-and-match composition in the Gamma Framework," in *25th Minisymposium, Department of Measurement and Information Systems*, Budapest, Hungary, January 2018.

[5] B. Graics, "Documentation of the Gamma Statechart Composition Framework," Budapest Univ. of Technology and Economics, Dept. of Measurement and Information Systems, Tech. Rep., 2016. [Online]. Available: https://inf.mit.bme.hu/en/gamma/

[6] S. Procter and P. Feiler, "The AADL error library: An operationalized taxonomy of system errors," *Ada Lett.*, vol. 39, no. 1, p. 6370, Jan. 2020. [Online]. Available: https://doi.org/10.1145/3379106.3379113

Abstraction-Based Model Checking of Linear Temporal Properties

Milán Mondok, András Vörös
Budapest University of Technology and Economics
Department of Measurement and Information Systems
Email: `mondokm@edu.bme.hu, vori@mit.bme.hu`

***Abstract*—Even though the expressiveness of linear temporal logic (LTL) supports engineering application, model checking of such properties is a computationally complex task and state space explosion often hinders successful verification. LTL model checking consists of constructing automata from the property and the system, generating the synchronous product of the two automata and checking its language emptiness. We propose a novel LTL model checking algorithm that uses abstraction to tackle the challenge of state space explosion. This algorithm combines the advantages of two commonly used model checking approaches, counterexample-guided abstraction refinement and automata theoretic LTL model checking. The main challenge in combining these is the refinement of "lasso"-shaped counterexamples, for which task we propose a novel refinement strategy based on interpolation.**

I. Introduction

Linear temporal logic (LTL) specifications are particularly expressive and thus easy to use for engineers, but LTL model checking is a computationally expensive task. An efficient and commonly used linear temporal logic verification algorithm is based on automata theory. It consists of constructing automata from the property and the system, generating the synchronous product of the two automata and checking its language emptiness. This reduces the LTL model checking task to product calculation and language emptiness checking, which can be efficiently computed on Büchi automata, but the problem of state space explosion still hinders verification.

As the number of state variables in a system increases, the system's state space grows at least exponentially, which makes the exploration resource-intensive. Several approaches were developed to tackle the challenge of state space explosion. Counterexample-guided abstraction refinement checks a simplified model instead of the original problem, iteratively adding more detail until the verification task can be decided. Abstraction-based solutions proved efficient in reachability analysis, but have not been elaborated in the domain of LTL model checking yet.

We propose a novel LTL model checking algorithm that performs automata theoretic model checking on an iteratively refined abstract model. The abstraction is refined using a novel algorithm based on interpolation.

Our approach is similar to the one described by Zhao Duan et al. in [4]. As an optimization, they limit the scope of the verification to terminable programs and define an alternate version of LTL that is interpreted over finite paths. These alternate LTL formulas can be expressed using deterministic finite automata, which makes their verification computationally less demanding than regular LTL model checking.

II. Background

We use the following notation [6] from first-order logic (FOL) throughout our paper. Given a set of variables $V = \{v_1, v_2, ...\}$ let $V' = \{v_1', v_2', ...\}$ and $V^{\langle i \rangle} = \{v_1^{\langle i \rangle}, v_2^{\langle i \rangle}, ...\}$ represent the primed and indexed version of the variables. We use V' to refer to successor states and $V^{\langle i \rangle}$ for paths. Given an expression φ over $V \cup V'$, let $\varphi^{\langle i \rangle}$ denote the indexed expression obtained by replacing V and V' with $V^{\langle i \rangle}$ and $V^{\langle i+1 \rangle}$ respectively in φ.

A. Control flow automata

In our work we describe programs using *Control flow automata* (CFA) [6]. We define a *Control flow automaton* as a 4-tuple $\langle V, L, l_0, E \rangle$, where:

- $V = \{v_1, v_2, ..., v_k\}$ is the set of *variables*. Each variable v_i has an associated domain D_{v_i};
- L is the set of *control locations*, which model the program counter;
- $l_0 \in L$ is the initial location;
- $E \subseteq L \times Ops \times L$, where $op \in Ops$ are FOL formulas over V and V', is a set of directed edges representing the operations that are executed when control flows from the source location to the target.

A *concrete state* (l, c) is a pair of a location $l \in L$ and an interpretation $c \in D_{v_0} \times ... \times D_{v_n}$ that assigns a value $c(v) = d \in D_v$ to each variable $v \in V$ of its domain D_v. The set of initial states is $\{(l, c) | l = l_0\}$ and a transition exists between states (l, c) and (l', c') if an edge $(l, op, l') \in E$ exists with $(c, c') \models \text{op}$. A *concrete path* is a finite, alternating sequence of concrete states and operations $\sigma = ((l_1, c_1), \text{op}_1, ..., \text{op}_{n-1}, (l_n, c_n))$ if $(l_i, \text{op}_i, l_{i+1}) \in E$ for every $1 \le i < n$ and $(c_1^{\langle 1 \rangle}, c_2^{\langle 2 \rangle}, ..., c_n^{\langle n \rangle}) \models \bigwedge_{1 \le i < n} op_i^{\langle i \rangle}$, i.e., there is a sequence of edges starting from the initial location and the interpretations satisfy the semantics of the operations.

B. Counterexample-guided abstraction refinement

Counterexample-guided abstraction refinement (CEGAR) [2] [6] aims to tackle the problem of state space explosion by performing the verification task on a simpler, abstract model. The abstract model is an overapproximation of the concrete model: it contains all behaviours of the concrete model, but can contain additional behaviour as well. Such models are sufficient to prove the absence of counterexamples but can contain false positives, meaning that the counterexamples in the abstract models have to undergo further analysis.

The core of the CEGAR algorithm is the CEGAR-loop, which consists of two components, the *abstractor* and the *refiner*. The task of the abstractor is to calculate the abstract state space based on the current precision and to search for counterexamples, while the task of the refiner is to verify the concretizability of the abstract counterexample and refine the precision accordingly. The loop can only be left in two scenarios, either if the abstractor finds no counterexamples, or if the refiner finds that an abstract counterexample is feasible.

In *Boolean predicate abstraction* [6], an abstract state $s \in S$ in the set of abstract states is a Boolean combination of FOL predicates. A precision $\pi \in \Pi$ is a set of FOL predicates that are currently tracked by the algorithm. For example, if the current precision π contains two predicates, $(x < 0)$ and $(x < 1)$, then $true$, $x < 0$ or $!(x < 0) \land x < 1$ are examples of possible abstract states.

The result of the transfer function [6] $T(s, op, \pi)$ is the strongest Boolean combination of predicates in the precision that is entailed by the source state s and the operation op. This can be calculated by assigning a fresh propositional variable v_i to each predicate $p_i \in \pi$ and enumerating all satisfying assignments of the variables v_i in the formula $s \land op \land \bigwedge_{p_i \in \pi}(v_i \leftrightarrow p'_i)$. For each assignment, a conjunction of predicates is formed by taking predicates with positive variables and the negations of predicates with negative variables. The disjunction of all such conjunctions is the successor state s'.

Locations of the CFA are tracked explicitly. Abstract states $S_L = L \times S$ are pairs of a location $l \in L$ and a state $s \in S$. The transfer function extended with locations is $T_L((l,s),\pi) = \{(l',s') | (l, op, l') \in E, s' \in T(s, op, \pi)\}$, i.e., (l', s') is a successor of (l, s) if there is an edge between l and l' with op and s' is a successor of s with respect to the inner transfer function T.

An *abstract path* $\sigma = ((l_1, s_1), op_1, ..., op_{n-1}, (l_n, s_n))$ is an alternating sequence of abstract states and operations. An abstract path is *feasible* if a corresponding concrete path $((l_1, c_1), op_1, ..., op_{n-1}, (l_n, c_n))$ exists, where each c_i is mapped to s_i.

The abstractor explores the abstract state space using a search strategy (such as DFS of BFS) looking for counterexamples, i.e., abstract paths that start in the initial state and end in an error state. The exploration starts in the abstract state $(l_0, true)$. When visiting a state, all of its unvisited successors with respect to the transfer function T_L are visited by the search. The search can be optimized by not visiting *covered* successors, i.e. abstract states (l_c, s_c), for which an already visited (l_v, s_v) exists such that $l_c = l_v$ and $(s_c \Rightarrow s_v)$. If all reachable states were visited and no counterexample was found, then the model is *safe*, however, if a counterexample was found the refiner needs to check its validity.

The refinement [6] happens as follows. The input is a path $\sigma = ((l_1, s_1), op_1, (l_2, s_2), op_2, ..., op_{n-1}, (l_n, s_n))$ and the current precision π. First, the feasibility of the path is decided by querying an SMT solver with the formula $s_1^{\langle 1 \rangle} \land op_1^{\langle 1 \rangle} \land s_2^{\langle 2 \rangle} \land op_2^{\langle 2 \rangle} \land ... \land op_{n-1}^{\langle n-1 \rangle} \land s_n^{\langle n \rangle}$. If this formula is satisfiable, then the model is *unsafe* and a satisfying assignment to this formula is returned as the counterexample. Otherwise, an interpolant is calculated from the infeasible path σ that holds information for the further steps of refinement.

A Craig interpolant [7] for a mutually inconsistent pair of formulas (A,B) is a formula that is (1) implied by A, (2) inconsistent with B, and (3) expressed over the common variables of A and B.

A binary interpolant for an infeasible path σ can be calculated by defining $A \equiv s_1^{\langle 1 \rangle} \land op_1^{\langle 1 \rangle} \land ... \land op_{i-1}^{\langle i-1 \rangle} \land s_i^{\langle i \rangle}$ and $B \equiv op_i^{\langle i \rangle} \land s_{i+1}^{\langle i+1 \rangle}$, where i corresponds to the longest prefix of σ that is still feasible. The refined precision returned is the union of π and the new predicate that is obtained by replacing the variables $V^{\langle i \rangle}$ with V in this interpolant.

C. Automata theoretic LTL model checking

Kripke structures, LTL expressions and Büchi automata can all be used to characterize ω-regular languages [10]. As LTL expressions can only characterize a strict subset of ω-regular languages, while every ω-regular language can be recognized by a Büchi automaton, all LTL-expressions can be transformed to equivalent Büchi automata, for example using the algorithm of Gerth et al [5].

We regard the state space of the model as a Kripke structure M. Given an LTL-formula φ let $L(M)$ and $L(\varphi)$ denote the language that the Kripke structure can produce and the language that the LTL-formula specifies. The LTL model checking problem [3] can now be restated as follows: is the set of provided behaviours a subset of the valid behaviours, i.e., does $L(M) \subseteq L(\varphi)$ hold?

An equivalent formalization is $L(M) \cap \overline{L(\varphi)} \overset{?}{=} \emptyset$, where $\overline{L(\varphi)}$ is the complement of the language $L(\varphi)$. Complementation is computationally hard, but it can avoided in case of LTL model checking by utilizing that the complement of the language of an LTL-formula is the language of the negated formula: $\overline{L(\varphi)} \equiv L(\neg\varphi)$. This allows the model checking problem to be reduced to language intersection and language emptiness, both of which can be efficiently computed on Büchi automata.

A possible way of checking the language emptiness of a Büchi automaton is checking whether at least one strongly connected component (SCC) that contains an accepting state is reachable from the initial state. If such an SCC is reachable, then the Büchi automaton contains at least one run that contains an accepting state infinitely many times, fulfilling the acceptance condition of Büchi automata. Tarjan's algorithm [9] identifies SCCs using a single depth-first search (DFS) and clever indexing. Algorithms based on Nested DFS [8] offer a different approach. These algorithms usually conduct two depth-first searches, the former one to find and sort accepting states, and the latter one to find cycles that contain accepting states.

III. Overview of the Approach

LTL model checkers have always struggled with performance. We propose to use counterexample-guided abstraction refinement in LTL model checking. The key idea of our approach (Fig. 1) is that we conduct the automata theoretic LTL model checking on abstract models that we iteratively refine to the required precision using the the CEGAR algorithm.

The algorithm can work with various abstract domains, such as explicit value abstraction [1], predicate abstraction, or even a mix of the two. The appropriate abstraction method can only be selected based on the desired application domain. In this paper, we present the algorithm using predicate abstraction, a variant more suited for reactive systems as variables in such systems usually only get assigned a relatively small subset of their domains as values.

The algorithm has the following steps:

1) The requirement specification is given in the form of an LTL-formula φ. Negate this formula and transform it to an equivalent Büchi automaton S;
2) Apply abstraction to the concrete model with the current precision, calculate the abstract state space and represent it with an automaton M;
3) Calculate the synchronous product of the two automata $S \times M$. During each step of the product the model automaton steps first, then the specification automaton steps based on the target state of the model automaton.
4) Check the language emptiness of the product automaton $S \times M$;
   - If the language of the product is empty, then the model meets the correctness specification as no counterexamples were found;
   - If a counterexample is found in the abstract state space, then verify whether it is feasible in the concrete state space as well;
     – A feasible counterexample means that the model does not meet the correctness specification (i.e. is *unsafe*), as we found a contradicting trace;
     – If the counterexample isn't feasible in the concrete system (i.e. *spurious*), then refine the precision and jump to step 2.

When using a suitable language emptiness checking algorithm such as Nested DFS [8], the tasks of state space generation, calculation of the product automaton and language emptiness checking can be conducted together, which can result in a significant increase in performance. If these three tasks are carried out at the same time, then the model checking is said to happen "on-the-fly".

IV. Refinement

In this section we present a novel refinement method for predicate abstraction. The algorithm searches for counterexamples that have a "lasso"-like form. The first part of the counterexample is a path leading to an accepting state and the second part is a cycle which starts and ends in said accepting state. If such a counterexample is found, then an accepting run is possible, because by repeatedly traversing the cycle, an accepting state can be explored infinitely many times, fulfilling the Büchi acceptance condition.

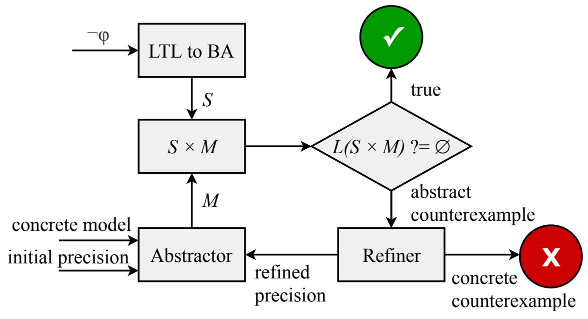


Fig. 1. Overview of CEGAR-based LTL model checking.

The CEGAR algorithm is usually used for reachability checking, where counterexamples are abstract paths leading from the initial state to an error state. When verifying these counterexamples the only thing that needs to be checked is whether such a path exists in the concrete model, whose states and transitions all correspond to the states and transitions of the abstract path. However, the fulfillment of this condition is required, but not enough, when analysing a cycle. A path that is not a cycle in the concrete model might appear as one in the abstract model.

We developed a novel counterexample refinement strategy that is capable of handling "lasso"-like counterexamples. The input is an abstract path $\sigma = ((l_1, s_1), op_1, (l_2, s_2), op_2, ..., op_{n-1}, (l_n, s_n))$ and an integer $1 \leq cycle \leq n$ that is the index of the initial state of the cycle, i.e. the recurrent accepting state ($s_{cycle} = s_n$). The path is first fed to the traditional CEGAR refinement algorithm presented in Section II-B. Based on the result of this algorithm, we have two options. If the algorithm finds that the path isn't traversable and returns a refined precision, then we simply return this refined precision. However, if the algorithm finds that the path is traversable, then we conduct further analysis to decide whether it is traversable in such a way that the initial and the end state of the cycle are the same concrete states.

Control locations are tracked explicitly during state space exploration, thus deciding whether two concrete states that belong to the same abstract state are identical can be done by comparing their data values (i.e. the values assigned to the variables in them). We construct a constraint $B \equiv \bigwedge_{v \in V} v^{\langle cycle \rangle} = v^{\langle n \rangle}$, which expresses that each variable has the same value in the initial and end state of the cycle, i.e. they are the same concrete states. We also construct the same formula that the refinement algorithm in II-B used to verify traversability, $A \equiv s_1^{\langle 1 \rangle} \wedge op_1^{\langle 1 \rangle} \wedge ... \wedge s_{cycle}^{\langle cycle \rangle} \wedge op_{cycle}^{\langle cycle \rangle} \wedge ... \wedge op_{n-1}^{\langle n-1 \rangle} \wedge s_n^{\langle n \rangle}$. By querying an SMT solver with the conjuntion of these two formulas, i.e., $A \wedge B$, we

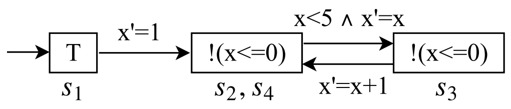


Fig. 2. Example of an abstract counterexample.

verify whether the counterexample is feasible. If this formula is satisfiable then the model does not meet the requirement specification (i.e. is *unsafe*) and a satisfying assignment is returned as counterexample. If the formula isn't satisfiable (i.e. is *spurious*), then we refine the precision by calculating an interpolant based on this formula.

Algorithm 1 Lasso refinement

```
    Input: $\sigma$: abstract path, $cycle$: initial index of cycle
    Output: ($unsafe$ or $spurious$, $\pi'$)
 1: procedure $lasso\_refine(\sigma, cycle)$
 2:     $res := refine(\sigma)$
 3:     if $res$ is $spurious$ then return $res$
 4:     else
 5:         $A \equiv s_1^{\langle 1 \rangle} \wedge op_1^{\langle 1 \rangle} \wedge ... \wedge op_{n-1}^{\langle n-1 \rangle} \wedge s_n^{\langle n \rangle}$
 6:         $B \equiv \bigwedge_{v \in V} v^{\langle cycle \rangle} = v^{\langle n \rangle}$
 7:         if $A \wedge B$ is feasible then return $(unsafe, \pi)$
 8:         else
 9:             $I \leftarrow$ get interpolant for $(A, B)$
10:             $\eta \leftarrow$ get satisfying assignment for $A$
11:             $\pi' \leftarrow$ create predicate from $I$:
12:                 replace all $v^{\langle n \rangle} \in I$ with $v$
13:                 replace all $v^{\langle cycle \rangle} \in I$ with $\eta(v^{\langle cycle \rangle})$
14:             return ($spurious$, $\pi \cup \pi'$)
```

To refine the precision we obtain an interpolant I for A and B. This interpolant is interpreted over $V^{\langle cycle \rangle}$ and $V^{\langle n \rangle}$, let's denote this with $I(v_1^{\langle cycle \rangle}, ..., v_k^{\langle cycle \rangle}, v_1^{\langle n \rangle}, ..., v_k^{\langle n \rangle})$. We also query the SMT solver for a satisfying assignment η to the formula A, which describes a concrete path $\sigma = ((l_1, c_1), \mathrm{op}_1, ..., \mathrm{op}_{n-1}, (l_n, c_n))$, where $c_{cycle} \neq c_n$. To ensure that the spurious counterexample described by η isn't found again during later explorations of the abstract state space, we need to extend our precision with a new predicate $\pi'(v_1, ..., v_k)$ that evaluates to $false$ in c_{cycle} and to $true$ in c_n (or vice versa), so that c_{cycle} and c_n get mapped to different abstract states. Formally, $\pi'(\eta(v_1^{\langle cycle \rangle}), ..., \eta(v_k^{\langle cycle \rangle})) = false$ and $\pi'(\eta(v_1^{\langle n \rangle}), ..., \eta(v_k^{\langle n \rangle})) = true$. To construct the predicate π' from the interpolant I, we replace the variables $V^{\langle n \rangle}$ with V, and $V^{\langle cycle \rangle}$ with values that are assigned to them by η. Formally, $\pi'(v_1, ..., v_k) := I(\eta(v_1^{\langle cycle \rangle}), ..., \eta(v_k^{\langle cycle \rangle}), v_1, ..., v_k)$.

If we evaluate $\pi'(\eta(v_1^{\langle n \rangle}), ..., \eta(v_k^{\langle n \rangle}))$, i.e. π' in c_n, we get $I(\eta(v_1^{\langle cycle \rangle}), ..., \eta(v_k^{\langle cycle \rangle}), \eta(v_1^{\langle n \rangle}), ..., \eta(v_k^{\langle n \rangle}))$, which is $true$, because of the first property of Craig interpolants ($A \rightarrow I$), from which it follows that if an assignment η satisfies A, then it also satisfies I.

Evaluating π' in c_{cycle} however, results in $I(\eta(v_1^{\langle cycle \rangle}), ..., \eta(v_k^{\langle cycle \rangle}), \eta(v_1^{\langle cycle \rangle}), ..., \eta(v_k^{\langle cycle \rangle}))$, which is $false$. In this case the variables $V^{\langle cycle \rangle}$ are assigned the same values as their counterparts $V^{\langle n \rangle}$, which means that B is $true$. It follows that I in this case is $false$, because of the second property of Craig interpolants ($I \wedge B$ is unsatisfiable),

We demonstrate the refinement process on the abstract counterexample in Fig. 2. The white rectangles represent abstract states with the applying predicates displayed inside them, the arrows represent transitions, the precision only contains one predicate, $(x \leq 0)$. The value of cycle and n is 2 and 4, respectively. We construct the following formulas based on this path:

$$A \equiv true \ \wedge \ x^{\langle 2 \rangle}{=}1 \wedge !(x^{\langle 2 \rangle}{\leq}0) \ \wedge \ x^{\langle 2 \rangle}{<}5 \wedge x^{\langle 3 \rangle}{=}x^{\langle 2 \rangle} \ \wedge \ !(x^{\langle 3 \rangle}{\leq}0) \ \wedge \ x^{\langle 4 \rangle}{=}x^{\langle 3 \rangle}{+}1 \wedge !(x^{\langle 4 \rangle}{\leq}0)$$

$$B \equiv x^{\langle 2 \rangle}{=}x^{\langle 4 \rangle}$$

By querying an SMT solver with the formula $A \wedge B$ we find that the counterexample in spurious, as $A \wedge B$ isn't satisfiable. The solver returns the interpolant $I \equiv x^{\langle 2 \rangle} < x^{\langle 4 \rangle}$ (note that this is only one of the possible interpolants). We request a satisfying assignment for A from the solver, and construct a predicate from I by replacing $x^{\langle 4 \rangle}$ with x and $x^{\langle 2 \rangle}$ with 1 (the value that is assigned to it in the satisfying assignment). Finally, we return that the counterexample is spurious, accompanied by the refined precision $(x \leq 0), (1 < x)$.

V. Conclusion

In our paper we examined LTL model checking and proposed a novel algorithm, which combines the advantages of counterexample-guided abstraction refinement and automata theoretic LTL model checking. We also proposed a novel refinement method for predicate abstraction. We implemented our algorithm in the Theta framework [11], but chose to omit experimental evaluation from this paper due to the lack of space.

References

[1] Dirk Beyer and Stefan Löwe. Explicit-state software model checking based on cegar and interpolation. 2013.

[2] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. 2003.

[3] Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 1999.

[4] Zhao Duan, Cong Tian, and Zhenhua Duan. Verifying temporal properties of c programs via lazy abstraction. 2017.

[5] Rob Gerth, Doron Peled, Moshe Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. 1996.

[6] Ákos Hajdu and Zoltán Micskei. Efficient strategies for cegar-based model checking. 2019.

[7] Kenneth McMillan. Applications of craig interpolation to model checking. 2005.

[8] Stefan Schwoon and Javier Esparza. A note on on-the-fly verification algorithms. 2005.

[9] Robert Tarjan. Depth first search and linear graph algorithms. 1972.

[10] Wolfgang Thomas. Handbook of theoretical computer science (vol. b). chapter Automata on Infinite Objects. 1990.

[11] Tamás Tóth, Ákos Hajdu, András Vörös, Zoltán Micskei, and István Majzik. Theta: a framework for abstraction refinement-based model checking. 2017.

Incremental view maintenance in graph databases: A case study in Neo4j

Márton Elekes, Gábor Szárnyas
Budapest University of Technology and Economics
Department of Measurement and Information Systems
Budapest, Hungary
Email: {elekes, szarnyas}@mit.bme.hu

***Abstract*—With the increasing amount of densely connected data sets, graph analysis has become an integral part of data processing pipelines. Therefore, the last decade saw the emergence of numerous dedicated graph analytical systems along with specialized graph database management systems. Traditionally, graph analytical tools targeted global fixed-point computations, while graph databases focused on simpler transactional read operations such as retrieving the neighbours of a node. However, recent applications of graph processing (such as financial fraud detection and serving personalized recommendations) often necessitate a mix of the two workload profiles. Following this trend, the 2018 Transformation Tool Contest (an annual competition for graph transformation tools) presented a case study that requires participants to compute complex graph queries defined on a continuously changing social network graph. The solutions are assessed based on their scalability and query reevaluation time, therefore, solutions are encouraged to incrementalize their implementations. This paper demonstrates a solution in the popular Neo4j graph database using several incrementalization techniques and compares them against the reference implementation of the case study.**

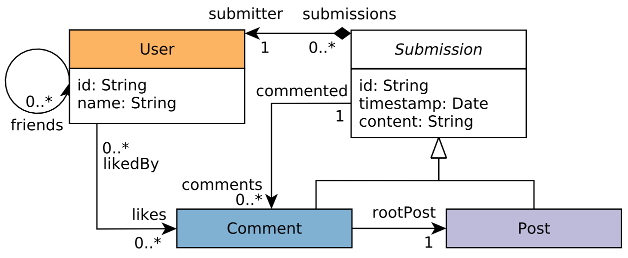


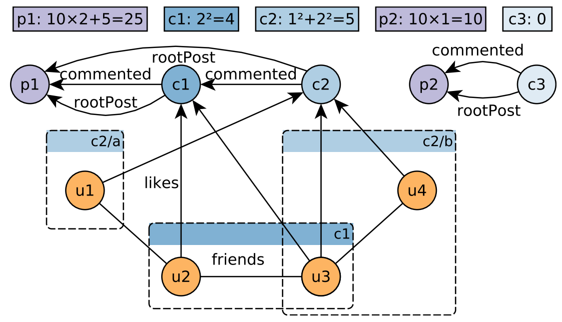
Fig. 1: Graph schema of the case study.

I. Introduction

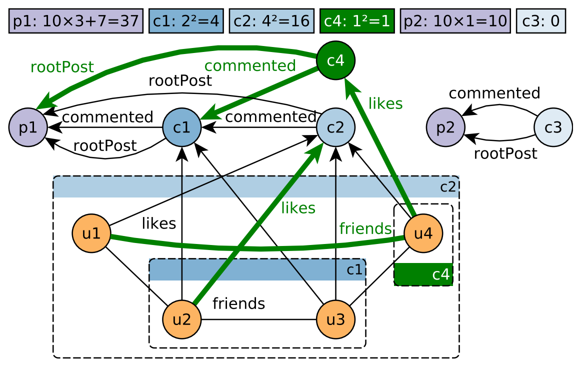
We start by describing the "Social Media" case study of the 2018 Transformation Tool Contest [5]. This case study is defined using a familiar social network-like data model (Fig. 1) consisting of Users and their Submissions. These submissions form a tree where the root node is a Post and the rest of the nodes are Comments. Users can like Comments and form friends relations with each other. Additionally, Comments have a direct pointer rootPost to the root Post to allow quick lookups. Fig. 2a shows an example graph with two Posts (p1, p2), three Comments (c1, c2, c3) and four Users (u1, ..., u4). Solutions are required to compute two queries:

Q1: influential posts. Assign a score to each Post, defined as 10 times the number of their (direct or indirect) Comments plus the number of Users liking those Comments. Return the top 3 Posts according to their score.

Q2: influential comments. Assign a score to each Comment, based on the friendships of the Users who like that Comment. Based on the graph formed by the User nodes and their friends edges, for every comment we define an induced subgraph which contains the Users who like the Comment and their friends edges. The subgraph contains connected components,



(a) Initial graph and scores. Comment c2 has two components: c2/a consists of User u1, while c2/b consists of Users u3 and u4. Its total score is the sum of the component sizes, i.e. $1^2 + 2^2 = 5$.



(b) Graph after performing an update that inserted six entities: (1) a friends edge between Users u1 and u4, (2) a likes edge from User u2 to Comment c2, (3) a Comment node c4 with (4) an outgoing rootPost edge to Post p1, (5) an outgoing commented edge to Comment c1, and (6) an incoming likes edge from User u4. The changes have increased the score of Post p1 and resulted in Comment c2 having a single component of size 4, therefore receiving a score of $4^2 = 16$. Comment c4 getting a score of $1^2 = 1$.

Fig. 2: Example graphs: initial and updated versions.

		EnumPaths (very slow)	EnumPaths Subgraph △	Reachability Subgraph (incompatible)	**MatComp FixedPoint** ◻	MatComp PathOp ▨	**GraphAlgo** ⊠	GraphAlgo EdgeFilter ×	MatComp GraphAlgo ⊠
initial evaluation	enumerate paths	●	●			●			
	materialize subgraph (edges/labels)		●	↯	●	●			
	reachability (shortestPath)			↯					
	reachability+dynamic labels (APOC lib.)				●				
	fixed-point calculation (APOC library)				●				
	materialize subgraph components				●	●			●
	Graph Algorithms library						●	●	●
update	reevaluation						●	●	
	maintain component	n/a		n/a	●	●			●
	dirty flag for Comments		●						

TABLE I: Comparison of strategies for Q2 regarding the techniques (language features and libraries) used in each strategy. Strategies in **bold** are described in more detail. Notation – ●: the technique is used by the strategy, ↯: incompatible strategies. “MatComp” strategies are denoted with ◻, while “GraphAlgo” strategies are denoted with ×.

i.e. groups of users who know each other directly or via friends. The score is defined as the sum of squared component sizes. Finally, the top 3 Comments should be returned.

Updating the graph. After the query execution on the initial graph the case study requires solutions to perform a number of updates on the graph while maintaining the results of the queries. Fig. 2 shows the initial graph and the updated graph with the scores for Q1 and Q2.

Neo4j. Neo4j is a graph database management system using the *property graph* data model. Such graphs consist of labelled entities, i.e. nodes and edges, which can be described with *properties* encoded as key-value pairs. Neo4j uses the Cypher query language [2] which offers both read and update constructs [3]. While the main focus of Neo4j is to run graph queries, it also supports graph analytical algorithms with the recently released Graph Algorithms library [7].

II. Approach

Q1 Batch. Q1 can be expressed with the Cypher query in Listing 1. The Cypher language uses node labels (e.g. Post, Comment, User), edge types (e.g. ROOT_POST, LIKES), node and edges properties to express graph patterns. The query matches every node with label Post, then all its comments via the rootPost edges, then the Users via the likes edges. `OPTIONAL MATCH` denotes an optional pattern, where variables are filled with NULL values if there is no match. `RETURN` can be also used to group and aggregate. The results are grouped by the id and timestamp properties of the Posts, aggregated, then the top 3 scores are returned. The aggregation counts the likes using the number of Users (a User can like more Comments), and counts the number of Comments (`DISTINCT` is used to remove duplicate Comments).

```
MATCH (p:Post)
OPTIONAL MATCH (p)<-[:ROOT_POST]-(c:Comment)
OPTIONAL MATCH (c)<-[:LIKES]-(u:User)
RETURN p.id AS id,
  10*count(DISTINCT c)+count(u) AS score,
  p.timestamp AS timestamp
ORDER BY score DESC, timestamp DESC LIMIT 3
```

Listing 1: Q1 Batch.

Q1 Incremental. To incrementally evaluate Q1, we initially compute the score for each Post as previously and store it in score property (Listing 2). Based on this property the current top 3 scores can be computed using Listing 3. The score property is indexed to improve lookup times.

For every batch of updates Alg. 1 is executed to insert new elements and update the score property, then Listing 3 is used to get the top 3.

```
MATCH (p:Post)
OPTIONAL MATCH (p)<-[:ROOT_POST]-(c:Comment)
OPTIONAL MATCH (c)<-[:LIKES]-(u:User)
WITH p, 10*count(DISTINCT c)+count(u) AS score
SET p.score = score
```

Listing 2: Q1 Incremental – initial evaluation.

```
MATCH (p:Post)
WHERE p.score >= 0 // query hint to use index
RETURN p.id AS id, p.score AS score,
  p.timestamp AS timestamp
ORDER BY score DESC, timestamp DESC LIMIT 3
```

Listing 3: Q1 Incremental – get top 3 results.

Algorithm 1 Maintaining scores for Q1

```
procedure UpdateQ1(updates)
    for all update ∈ updates do
        AddNewElement(update)                      ▷ insert into the graph
        if update is Post then
            update.score ← 0                       ▷ init score for new Post
        else if update is ⟨Comment, Post⟩ then     ▷ new Comment node
            ⟨_, rp⟩ ← update                       ▷ get the root Post
            rp.score ← rp.score + 10               ▷ update score
        else if update is ⟨User, Comment⟩ then     ▷ new likes edge
            ⟨_, c⟩ ← update                        ▷ get Comment vertex of new edge
            rp ← RootPost(c)                       ▷ navigate via rootPost
            rp.score ← rp.score + 1                ▷ update score
        end if
    end for
end procedure
```

Q2 Strategies. Table I compares the different strategies used for Q2 to find connected components in a subgraph and handle updates. Fig. 3 shows a comparison of their runtime during the initial evaluation and after the updates. The *EnumPaths* strategy finds the connected components by enumerating all paths of friends edges between Users who like a comment and filtering to ensure that every User on the path also likes the Comment. The complexity of this operation is intractable, therefore this strategy is not feasible on larger graphs. To only enumerate valid paths in the subgraph, the edges of the subgraph can be materialized for each subgraph, i.e. for each Comment (stored as an edge property). The combination of these techniques (*EnumPaths Subgraph* △) also exhibits poor scalability (later measurements show that it reaches the time limit after graph size 8). This limitation could be resolved by using a Cypher reachability query, but unfortunately such queries cannot perform filtering on properties (which is necessary to ensure we only enumerate valid paths).

To use reachability queries and avoid the costly enumeration of paths, *MatComp FixedPoint* □ strategy uses Neo4j's APOC library[1]. It materializes subgraph components by converting $\text{User}\circ\xrightarrow{\text{likes}}\circ\text{Comment}$ edges to $\text{Comment}\circ\xrightarrow{\text{component}}\circ\text{Component}$ and $\text{Component}\circ\xrightarrow{\text{user}}\circ\text{User}$ edges where the Component node connects every User who knows each other directly or via friends. The conversion is executed for each component one by one using the fixed-point query execution mechanism of APOC. This strategy has the best performance, which is caused by the use of reachability function and incremental maintenance after updates. Component materialization can be expressed with pure Cypher queries using path operations (*MatComp PathOp* ⊠, which also has limited scalability due to the use of path enumeration.

GraphAlgo strategies (×) use functions provided by the Neo4j Graph Algorithms library[2] [7] to find connected components in a subgraph conveniently. The library loads the subgraph into an in-memory projected subgraph before running the computations. The initial runtimes of these solutions are worse than the runtime of *MatComp FixedPoint*. This can be attributed to the load phase, which is run for every subgraph, i.e. for every Comment.

Strategies can differ in the way they handle the updates: they can fully reevaluate the queries or incrementally maintain the results depending on the update. Repeated reevaluations take a significant amount of time, which causes the execution to time out. *EnumPaths Subgraph* △ strategy uses Comment-level incrementalization with dirty flags, but using path enumerations limits its scalability. The incremental evaluation of materialized components (□) is implemented by merging the components and maintaining their sizes and the scores. These updates have the best performance. (The *MatComp* strategies with worse initial runtime reaches the time limit.)

Q2 Batch. Listing 4 shows the *GraphAlgo* solution for Q2 using Neo4j Graph Algorithms library. The `algo.unionFind.stream` function is used to find connected components of the subgraph given by the node labels and edge types or Cypher queries. For each Comment, the first Cypher query in Lines 3–7 select Users who like the Comment, the second query selects all friend edges as pairs

[1]https://neo4j.com/labs/apoc/
[2]https://neo4j.com/docs/graph-algorithms/

```
1  MATCH (c:Comment)
2    CALL algo.unionFind.stream(
3      'MATCH (c:Comment)<-[:LIKES]-(u:User)
4      WHERE id(c)=' + id(c) + '
5      RETURN id(u) as id',
6      'MATCH (u1:User)-[:FRIEND]->(u2:User)
7      RETURN id(u1) as source, id(u2) as target',
8      {graph: 'cypher'})
9    YIELD setId
10   WITH c, setId, count(setId) AS cSize
11   WITH c, cSize * cSize AS cSize_2
12   RETURN c.id AS id, sum(cSize_2) AS score, c.
       timestamp
13   ORDER BY score DESC, c.timestamp DESC LIMIT 3
14 UNION ALL
15 MATCH (c:Comment)
16   WHERE NOT (c)<-[:LIKES]-(:User)
17   RETURN c.id AS id, 0 AS score, c.timestamp
18   ORDER BY c.timestamp DESC LIMIT 3
```

Listing 4: Q2 batch using the Neo4j Graph Algorithms library.

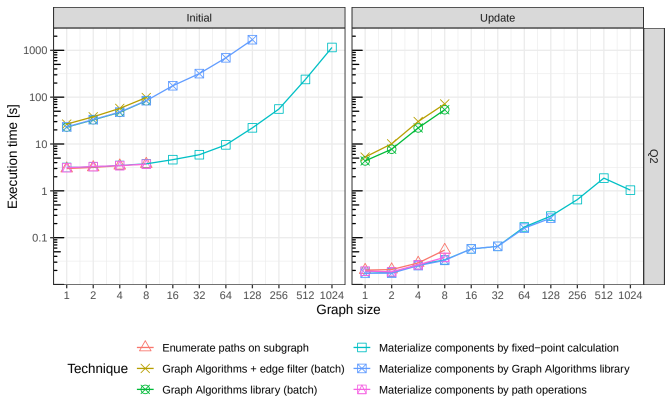


Fig. 3: Performance comparison of Q2 strategies. The symbols denoting techniques correspond to those shown in Table I.

of Users. The function returns the ID of the component containing the User node. Lines 10–14 calculate the squared sum of the component sizes and selects the top 3 scores. The function is invoked only for Comments with likes. The top 3 Comments without likes are enumerated by Lines 16–20.

Q2 Incremental. The incremental solution in this section (*MatComp FixedPoint*) materializes the components of the subgraph using fixed-point query execution of APOC library. To achieve this, the solution materializes subgraph edges as dynamically named labels (Listing 5) and finds reachable nodes using the APOC library (Listing 6). The incremental evaluation is performed by merging the components and maintaining their sizes and the scores (Listing 7).

III. Evaluation

To evaluate the performance and scalability of our solution, we have used the benchmark framework of the case study [5]. This executes the queries on graphs of increasing sizes as shown in Table II, then adds new elements to the graph, and maintains the result using the queries. (Similar queries can be formulated for element removal.) As a performance baseline,

```
MATCH (c)<-[:LIKES]-(u:User)
WITH c, collect(u) AS users
CALL apoc.create.addLabels(users, ['Likes_' + c.id])
     YIELD node
RETURN count(*)
```

Listing 5: Q2 – materializing subgraph by dynamic labelling of users liking the comment.

```
CALL apoc.periodic.commit("
  MATCH (c:Comment)<-[:LIKES]-(u1:User)
  WITH c, min(u1) AS u1
  CREATE (c)-[:COMPONENT]->(comp:Component)
  WITH c, u1, comp
  CALL apoc.path.subgraphNodes(u1,
    labelFilter: 'Likes_' + c.id,
    relationshipFilter: 'FRIEND'}) YIELD node AS u2
  CREATE (comp)-[:USER]->(u2)
  WITH c, comp, u2
  MATCH (c)<-[l:LIKES]-(u2)
  DELETE l
  WITH c, comp, count(*) AS componentSize
  SET comp.size = componentSize
  RETURN count(*)")
```

Listing 6: Q2 – grouping components by selecting a single user (u1) per comment and their reachable friends (u2) in the subgraph, then replacing LIKES edges with a Component node and COMPONENT and USER edges until reaching a fixed point where all LIKES edges are replaced.

```
WITH $friendEdge AS friendEdge
MATCH (cp1:Component)-[:USER]->(u1:User)
  -[friendEdge]->(u2:User)<-[:USER]-(cp2:Component)
  <-[:COMPONENT]-(c:Comment)-[:COMPONENT]->(cp1)
WITH c, cp1, cp2,
     cp1.size AS cp1Size, cp2.size AS cp2Size,
     cp1.size + cp2.size AS newCompSize
CALL apoc.refactor.mergeNodes([cp1, cp2],
  {mergeRels: true}) YIELD node AS newComp
SET newComp.size = newCompSize,
    c.score = c.score - cp1Size*cp1Size
    - cp2Size*cp2Size + newCompSize*newCompSize
```

Listing 7: Q2 – for every FRIEND edge inserted where the two users belonged to separate components, merge the components and maintain the size and scores accordingly. (A similar query exists for new LIKES edges.)

we used the reference implementation of the case study, written in the .NET Modeling Framework [4] (NMF Batch) and its incremental version (NMF Incremental). We executed the benchmark on a cloud machine with a 24-core Intel® Xeon® Platinum 8167M CPU with Hyper Threading at 2.00 GHz, 320 GB RAM, and HDD storage. The execution times are shown in Fig. 4. The results show that the batch variant does not scale, as it takes more than 20 minutes for graph size 8. However, the incremental Neo4j variant is able to scale for all graph sizes. Neither of them is competitive against the incremental NMF solution which achieves sub-second reevaluation times for both queries.

IV. Conclusion and Future Work

This paper presented a Neo4j-based solution for the "Social Media" case study of the 2018 Transformation Tool Contest. We discussed a number of techniques that allow incremental

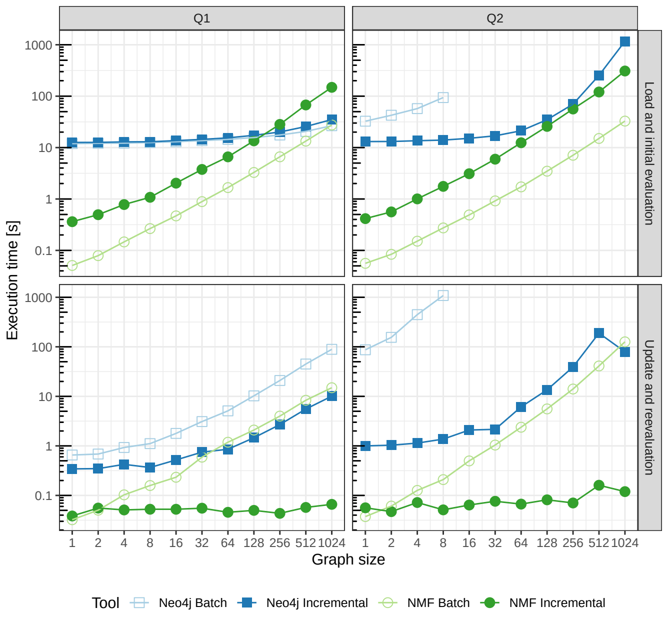


Fig. 4: Execution times of the queries with respect to the graph sizes in the "load and initial evaluation", and the "update and reevaluation" phases. Both the x axis (graph size) and the y axis (execution time) are logarithmic.

	1	2	4	8	16	32	64	128	256	512	1024
#nodes	1274	2071	4350	7530	15k	30k	58k	115k	225k	443k	859k
#edges	2533	4207	9118	18k	35k	71k	143k	287k	568k	1.1M	2.3M

TABLE II: Graph sizes w.r.t. to the scale factor.

evaluation during updates. Initial results show that incrementalization techniques provide significant performance benefits and allow the solution to scale for graph sizes orders of magnitude larger than batch solutions. In the future, we plan to perform a detailed performance evaluation of our solution against other query and transformation tools, including traditional relational databases (such as PostgreSQL), EMF-based model query engines (such as VIATRA [8]), and differential dataflow engines [6]. Our Neo4j solution is also subject to further optimizations such as using an incremental connected components algorithm [1].

References

[1] D. Ediger et al. Tracking structure of streaming social networks. In *IPDPS*, pages 1691–1699. IEEE, 2011.
[2] N. Francis et al. Cypher: An evolving query language for property graphs. In *SIGMOD*, pages 1433–1445. ACM, 2018.
[3] A. Green et al. Updating graph databases with Cypher. *PVLDB*, 2019.
[4] G. Hinkel. NMF: A multi-platform modeling framework. In *ICMT*, 2018.
[5] G. Hinkel. The TTC 2018 Social Media case. In *TTC at STAF*, 2018.
[6] F. McSherry et al. Differential dataflow. In *CIDR*, 2013.
[7] M. Needham and A. E. Hodler. *Graph Algorithms: Practical Examples in Apache Spark and Neo4j*. O'Reilly Media, 2019.
[8] D. Varró et al. Road to a reactive and incremental model transformation platform: three generations of the VIATRA framework. *Softw. Syst. Model.*, 2016.

Continuous time modelling of autocatalytic chemical reaction dynamics from sporadic inputs

Edward De Brouwer
ESAT-STADIUS
KU Leuven
Leuven, Belgium
edward.debrouwer@esat.kuleuven.be

Adam Arany
ESAT-STADIUS
KU Leuven
Leuven, Belgium
adam.arany@esat.kuleuven.be

Jaak Simm
ESAT-STADIUS
KU Leuven
Leuven, Belgium
jaak.simm@esat.kuleuven.be

Yves Moreau
ESAT-STADIUS
KU Leuven
Leuven, Belgium
moreau@esat.kuleuven.be

***Abstract*—Real-world time series are usually *sporadically* observed when collected outside of rigorous scientific experiments. The sampling is irregular across time and across dimensions, which breaks the typical assumption of most recurrent networks and dynamical process models. In this work, we propose to use our GRU-ODE-Bayes [1], a recently proposed model that posits a continuous latent process whose dynamics are driven by an ordinary differential equation parametrized with neural networks [2]. We apply this framework in the estimation of reaction dynamics from observational data. In particular, we show that our method is able to learn unstable autocatalytic chemical reactions within the large class of reactions driven by the Brusselator dynamical model [3], such as the Belousov–Zhabotinsky reaction in the presence of very few sporadic observation points.**

***Index Terms*—Neural Networks, Time Series, ODE, Chemical Reactions Dynamics**

I. Introduction and Motivation

Scientific research produces large quantity of time series data from various domains such as chemistry, astronomy, healthcare or climate science. Even though most statistical methods for analyzing this type of data assumes that signals are measured systematically at fixed constant time intervals, this assumption is often not hold in practice.

Indeed, due to some practical contingencies, much real-world data is *sporadic* (*i.e.*, the signals are sampled irregularly and not all signals are measured each time). A typical example is patient measurements, which are taken when the patient comes for a visit (*e.g.,* sometimes skipping an appointment) and where not every measurement is taken at every visit. Another example would be running lab experiments spread over several days when sensor errors can occur.

Modeling then becomes challenging as such data violates the main assumptions underlying traditional machine learning methods (such as recurrent neural networks). The most straightforward way to address this issue is to perform imputation, but this is often unsatisfactory as it results in biased estimates.

Recently, the GRU-ODE-Bayes model [1] proposed a novel way to address this type of sporadic time series for forecasting and classification. Based on the seminal Neural-ODE idea [2], it proposes a filtering approach that posits a *continuous* latent process that generates the observations. By framing the dynamics as an ordinary differential equation (ODE) parametrized by neural networks, the model is able to model complex dynamics and integrate it over arbitrary time intervals, therefore addressing the sporadicity issue in a natural way. Furthermore, it was shown that the continuity prior was providing extra performance when the continuity assumption is verified in practice, such as in patients clinical data prediction or weather forecasting.

This continuity assumption is also valid in chemical reaction dynamics. Indeed, chemical reactions are intrinsically continuous and can most of the time be expressed analytically as ODEs. However, reactions in multiple component mixtures are very complex and the true dynamic is often unknown. In this case, chemists try to approximate the true dynamic of the process by running some temporal experiments and learning the underlying dynamics of the reaction. What is more, as mentioned above, scientific practice sometimes lead to sporadic time series, as some concentration or variables are not measured continuously.

In this work, we propose to use GRU-ODE-Bayes to learn complex chemical reactions dynamics from sporadic data. We show that using very few data points, our model is able to learn the true chemical dynamics with high accuracy. The rest of

Edward De Brouwer is funded by a FWO-SB grant. Yves Moreau is funded by (1) Research Council KU Leuven: C14/18/092 SymBioSys3; CELSA-HIDUCTION, (2) Innovative Medicines Initiative: MELLODY, (3) Flemish Government (ELIXIR Belgium, IWT, FWO 06260) and (4) Impulsfonds AI: VR 2019 2203 DOC.0318/1QUATER Kenniscentrum Data en Maatschappij. Computational resources and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by the Research Foundation - Flanders (FWO) and the Flemish Government – department EWI. We also gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research. The research has also been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013, Thematic Fundamental Research Collaborations Grounding Innovation in Informatics and Infocommunications).

the paper is organized as follows. In section II, we review the related works relevant to ours. Section III presents succinctly the GRU-ODE-Bayes method and the main assumptions behind the model. Finally, results of our method on the broad class of Brusselator reactions are shown in Section IV.

II. Related works

Our work naturally builds upon the machine learning and statistics literature on (sporadic) time series and on the learning of dynamical systems in chemistry and life sciences in general.

A. Temporal modelling of sporadic time series

The most straightforward approach to handle sporadic time series consists in data imputation. One then feed the observation mask and times of observations jointly to the recurrent network [4]–[6]. Despite some promising experimental results, this approach relies too much on the ability of the model to distinguish imputed and true data points. Some works have tried to address this limitation by introducing more meaningful data representation for sporadic time series [7], [8], like tensors [9], [10].

Gaussian processes (GP) are the most popular generative approach to deal with sporadic observations. They have been used for smart imputation before a RNN or CNN architecture [11], [12], or to derive informative representations of time series [8]. However, they usually suffer from high uncertainty outside the observation support, are computationally intensive, and learning the optimal kernel is tricky.

Most recently, the seminal work of Neural ODEs [2], [13] suggested a continuous version of neural networks that overcomes the limits imposed by discrete-time recurrent neural networks. Coupled with a variational auto-encoder (VAE) architecture, it proposed a natural way of generating irregularly sampled data. However, the VAE nature of the limits its expressivity when it comes to forecasting.

Our method also has connections to the Extended Kalman Filter (EKF) that models the dynamics of the distribution of processes in continuous time. However, the practical applicability of the EKF is limited because of the linearization of the state update and the difficulties involved in identifying its parameters.

B. Learning dynamics in chemistry and life sciences

Mathematical modeling with ordinary differential equations (ODEs) has a very long tradition in biology and ecology. Efforts to apply ODEs to understand population dynamics started already in the 18th century (see, e.g., Malthus's growth model) [14]. More recently, several optimization methods have been proposed to fit ODE models to observational data in domains such as metabolic models and more general biological pathways [15]–[17]. However, those methods require a heavily parametrized formulation of the ODE, motivated by a lot of prior assumptions and expert knowledge. In contrast, the method we propose here allows very flexible representation of the ODE, which alleviates the risk of misparametrization, at the expense of more data samples required for training the model. To the best of our knowledge, this is the first attempt to use ODE parametrized by neural networks to fit biological/chemical processes from observational data.

III. Methods

In this work we are interested in uncovering the dynamics of a D dimensional process $\mathbf{Y}(t) \in \mathbb{R}^D$ driven by an unknown (stochastic) differential equation (SDE) which is observed sporadically :

$$d\mathbf{Y}(t) = \mu(\mathbf{Y}(t))dt + \sigma(\mathbf{Y}(t))d\mathbf{W}(t), \tag{1}$$

where $d\mathbf{W}(t)$ is a Wiener process. The distribution of $\mathbf{Y}(t)$ then evolves according to the Fokker-Planck equation. We refer to the mean and covariance parameters of its probability density function (PDF) as $\mu_{\mathbf{Y}}(t)$ and $\Sigma_{\mathbf{Y}}(t)$. Our goal is to estimate those time evolving parameters conditioned on previous observations.

In practice we have access to N realizations of the SDE but the parameters of the SDE can change from one realization to another. Furthermore, for each realization $\mathbf{Y}(t)_i$ with $(i = 0, ..., N-1)$, we only observed sporadic measurements. That is, we have a vector $\mathbf{t_i^*} \in \mathbb{R}^{T_i}$ of times where the process is observed along with a mask $\mathbf{m} \in \{0,1\}^{T_i \times D}$ that indicates which dimension is observed at each observation time. We then have missingness both across time and across dimensions.

A. GRU-ODE-Bayes

GRU-ODE-Bayes [1] was recently proposed as a new filtering method to deal with *sporadic* time series. It assumes a continuous latent process $\mathbf{h}(t)$ that generates the observations $\mathbf{Y}(t)$ through some mapping $\mathbf{Y}(t) \sim f_{obs}(h(t))$.

It consists of two modules: GRU-ODE, responsible for learning the continuous dynamics of the latent process that generates the observations and GRU-Bayes, responsible for dealing with incoming observations and update the conditional current estimate of the latent process.

1) GRU-ODE: The GRU-ODE module parametrizes the dynamics of the latent process $\mathbf{h}(t)$ with an Neural-ODE inspired from the classical GRU module. We use the following parametric ODE:

$$\frac{d\mathbf{h}(t)}{dt} = (1 - \mathbf{z}(t)) \odot (\mathbf{g}(t) - \mathbf{h}(t)), \tag{2}$$

Where $\mathbf{z}(t)$ and $\mathbf{g}(t)$ are given as in the GRU equations :

$$\begin{aligned}
\mathbf{r}_t &= \sigma(W_r \mathbf{x}_t + U_r \mathbf{h}_{t-1} + \mathbf{b}_r) \\
\mathbf{z}_t &= \sigma(W_z \mathbf{x}_t + U_z \mathbf{h}_{t-1} + \mathbf{b}_z) \\
\mathbf{g}_t &= \tanh(W_h \mathbf{x}_t + U_h (\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_h),
\end{aligned} \tag{3}$$

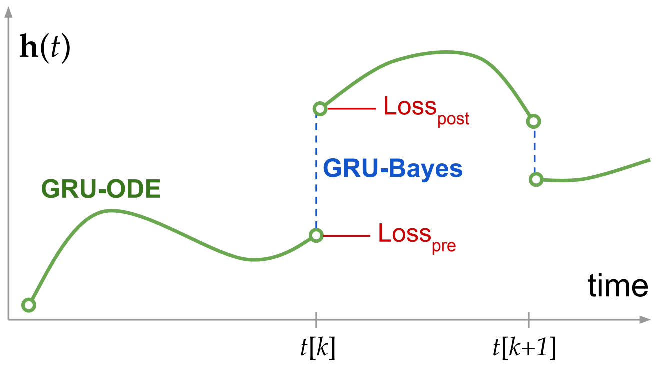


Fig. 1. GRU-ODE-Bayes uses GRU-ODE to evolve the hidden state between two observation times $t[k]$ and $t[k+1]$. GRU-Bayes processes the observations and updates the hidden vector $\mathbf{h}$ in a discrete fashion, reflecting the additional information brought in by the observed data.

2) GRU-Bayes: GRU-Bayes module is responsible for the update of the hidden state when new measurements are observed. As data comes in in packets, we allow the hidden process to jump to a new point in hidden space where it reflects more the newly observed data point as shown on Figure 1.

This update is performed by using a GRU cell that takes as input the previous hidden state and the current observation and then mimics a Bayesian update to set the hidden to a new value that matches the current observations :

$$\mathbf{h}(t_+) = \mathbf{GRU}(\mathbf{h}(t_-), f_{\text{prep}}(\mathbf{y}[k], \mathbf{m}[k], \mathbf{h}(t_-))), \tag{4}$$

3) End-to-end: At test time, the model performs predictions as suggested on Figure 1. We integrate the hidden process according to the GRU-ODE dynamics until the next observation (done with numerical integration). When an observation is reached, we process it with GRU-Bayes and update the hidden state. We then resume to GRU-ODE integration from the new initial point and continue until a next observation is reached. At each point in time, we can use $f_{obs}(.)$ to predict the distribution of the measurements.

B. The Brusselator

In order to demonstrate the capabilities of our approach, we choose to learn the dynamics of a range of reactions that can be modelled by the *Brusselator*. The Brusselator is a theoretical model for a type of autocatalytic chemical reactions. The type of chemical reactions it's modeling consists of two initial reactants A and B whose products X and Y are also catalysts of coupled reactions as shown below:

$$\begin{aligned} \mathrm{A} &\longrightarrow \mathrm{X} \\ 2\,\mathrm{X} + \mathrm{Y} &\longrightarrow 3\,\mathrm{X} \\ \mathrm{B} + \mathrm{X} &\longrightarrow \mathrm{Y} + \mathrm{D} \\ \mathrm{X} &\longrightarrow \mathrm{E} \end{aligned}$$

In the case of A and B being in large excess compared to products of above reactions, their concentration can be assumed constant in the solution and the following dynamics are obtained :

$$\frac{d}{dt}[X] = [A] + [X]^2[Y] - [B][X] - [Y] \tag{5}$$

$$\frac{d}{dt}[Y] = [B][X] - [X]^2[Y] \tag{6}$$

where the brackets stand for the concentration of the given chemical in the solution (*e.g.* in moles/l).

Equations 5 and 6 suggest that a fixed point is located at $[X] = A$ and $[Y] = [B]/[A]$. Importantly, this fixed point becomes unstable when:

Unstability condition :

$$B > 1 + A^2 \tag{7}$$

When the instability condition is satisfied, the concentrations X and Y start oscillating over time. An example of such oscillation is shown on figure 2.

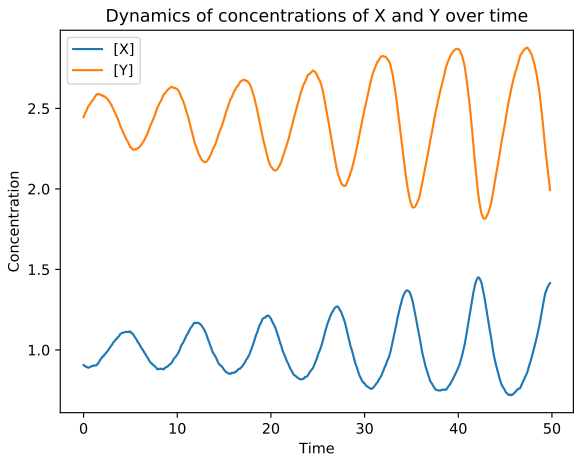


Fig. 2. Example of an unstable dynamic evolution of concentrations of products X and Y over time when A = 0.7 and B = 1.7.

The most well known example of this type of unstable autocatalytic reactions is the Belousov-Zhabotinsky reaction which can be created using a reducing agent (malonic acid) an oxidiser (bromate), and appropriate redox catalyst (eg.: manganese ion, cerium ion, or ferroin).

C. Dataset Generation

Because there is usually some measurement error and because of the uncertainty about the true nature of the hidden driving ODE, we generate trajectories of concentrations $[X]$ and $[Y]$ from a modified stochastic differential equation:

$$\begin{aligned} \frac{dx}{dt} &= 1 + (b+1)x + ax^2y + \sigma dW_1(t) \\ \frac{dy}{dt} &= bx - ax^2y + \sigma dW_2(t) \end{aligned} \tag{8}$$

Where $dW_1(t)$ and $dW_2(t)$ are correlated Brownian motions with correlation coefficient ρ. We simulate 1,000 trajectories driven by the dynamics given in Eq. 8 with parameters $a = 0.3$ and $b = 1.4$ such that the ODE is unstable.

We simulate those trajectories over a time period of 50 seconds from which we sample sporadically N_{samp} observations with $N_{samp} \sim \mathcal{U}niform(0.38, 0.42) * T$ and $T = 50$. For each of those observations, we then randomly sample across dimensions to generate missing value dimension wise. We select both dimensions with probability 0.8 and only one of them with probability 0.2. The resulting dataset is then a sporadic time series of the the generative SDE process from Equation 8.

IV. Results

We train the GRU-ODE-Bayes model on the data described in Section III. We use 700 trajectories for training, 200 for validation and leave 100 out for testing. We used DOPRI-5 for the numerical integration of the ODE. Prediction of our model on a test trajectory is displayed on Figure 3.

We observe that at the start of the trajectory, as no measurement is observed, the model is uncertain about the future of the trajectory, which is illustrated by the very large confidence intervals before the first observations (shaded areas). However, as soon as an observation is reached, the model can update its latent state more accurately. The uncertainty remains high until more observations are reached, when the model can condition the expected trajectory on more information. Furthermore, we observe that the model is able to capture correlations between both dimensions of the process as indicated by the red arrow on the figure, where dimension 2 is updated even if only information about dimension 1 is gathered.

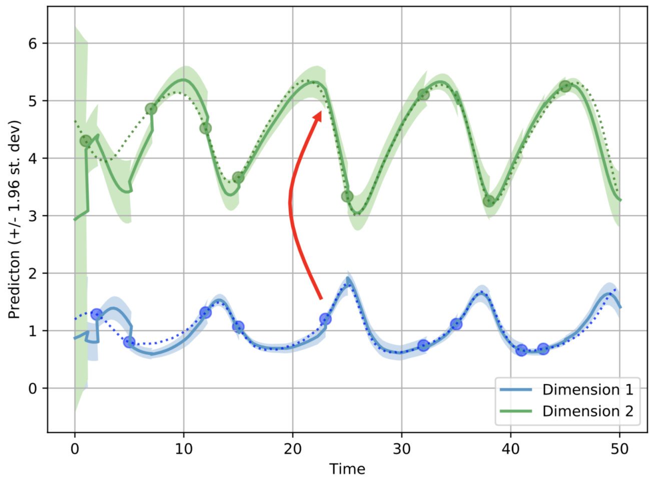


Fig. 3. Predictions of GRU-ODE-Bayes on an unknown realization of the BXLator process of Equation 8. Green and Blue are the two different concentrations we try to model over time. Solid lines represent the predicted means, shaded areas are the confidence intervals (1.96 standard deviations). Red arrow points out the update of dimension 2 when only dimension 1 is observed, showing that the model is able to detect meaningful correlation between both dimensions.

Table I further displays mean square errors (MSE) and negative log-likelihoods obtained on the test sets. They are computed by using the first 35 seconds of the processes as observable data and predictions are made on the remaining hidden observations between $t = 35$ and $t = 50$. We compare it with the Neural-ODE model as well as a Gaussian Process. We observe that our method clearly outperforms the competitors both in terms of MSE and log-likelihood. We motivate this performance by 1) the information bottleneck imposed in the Neural-ODE and 2) the lack of strong extrapolation capabilities in Gaussian processes. Furthermore, compared to a Gaussian process approach that would scale quadratically with the number of observations and dimensions, our model is much more efficient and scales linearly with the number of observations.

TABLE I
Test performance results

Model	**MSE**	**Neg-Loglik**
GRU-ODE-Bayes	0.11	-0.62
Neural-ODE	0.28	0.91
Gaussian Process	0.33	1.09

V. Conclusion

In this paper, we showed the fitness of the GRU-ODE-Bayes method to assess complex chemical reactions dynamics from sporadic observational data. We showed that our method was able to accurately learn the dynamics of the broad range of reaction dynamics modeled by the Brusselator equation. In order to make the problem more realistic, we added Browian motion noise to the dynamics, making every realization of the temporal process different from one another. Our approach outperformed a selected subset of baselines and seem to be suited for long term prediction of the concentrations (see Figure 3).

Those encouraging results motivate us to make this method more useful for the chemical community. In particular, an interesting application would be to be able to predict the constants a and b from a single experiment time series, allowing the chemist to infer the concentration of reactants A and B from the temporal measurements of X and Y. From preliminary experiments we expect that more time series realizations would be required to learn this task accurately. This is left for future work.

References

[1] E. De Brouwer, J. Simm, A. Arany, and Y. Moreau, "Gru-ode-bayes: Continuous modeling of sporadically-observed time series," *arXiv preprint arXiv:1905.12374*, 2019.

[2] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," in *Advances in Neural Information Processing Systems, 2018*, 2018.

[3] I. Prigogine, *From being to becoming*. Freeman, 1982.

[4] Z. Che, S. Purushotham, K. Cho, D. Sontag, and Y. Liu, "Recurrent neural networks for multivariate time series with missing values," *Scientific reports*, vol. 8, no. 1, p. 6085, 2018.

[5] E. Choi, M. T. Bahadori, A. Schuetz, W. F. Stewart, and J. Sun, "Doctor ai: Predicting clinical events via recurrent neural networks," in *Machine Learning for Healthcare Conference*, 2016, pp. 301–318.

[6] Z. C. Lipton, D. Kale, and R. Wetzel, "Directly modeling missing data in sequences with rnns: Improved classification of clinical time series," in *Machine Learning for Healthcare Conference*, 2016, pp. 253–270.

[7] A. Rajkomar, E. Oren, K. Chen, A. M. Dai, N. Hajaj, M. Hardt, P. J. Liu, X. Liu, J. Marcus, M. Sun *et al.*, “Scalable and accurate deep learning with electronic health records,” *npj Digital Medicine*, vol. 1, no. 1, p. 18, 2018.

[8] M. Ghassemi, M. A. Pimentel, T. Naumann, T. Brennan, D. A. Clifton, P. Szolovits, and M. Feng, “A multivariate timeseries modeling approach to severity of illness assessment and forecasting in icu with sparse, heterogeneous clinical data.” in *AAAI*, 2015, pp. 446–453.

[9] E. De Brouwer, J. Simm, A. Arany, and Y. Moreau, “Deep ensemble tensor factorization for longitudinal patient trajectories classification,” *arXiv preprint arXiv:1811.10501*, 2018.

[10] J. Simm, A. Arany, P. Zakeri, T. Haber, J. K. Wegner, V. Chupakhin, H. Ceulemans, and Y. Moreau, “Macau: Scalable bayesian factorization with high-dimensional side information using mcmc,” in *2017 IEEE 27th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, 2017, pp. 1–6.

[11] J. Futoma, S. Hariharan, and K. Heller, “Learning to detect sepsis with a multitask gaussian process rnn classifier,” *arXiv preprint arXiv:1706.04152*, 2017.

[12] M. Moor, M. Horn, B. Rieck, D. Roqueiro, and K. Borgwardt, “Temporal convolutional networks and dynamic time warping can drastically improve the early prediction of sepsis,” *arXiv preprint arXiv:1902.01659*, 2019.

[13] Y. Rubanova, R. T. Chen, and D. Duvenaud, “Latent odes for irregularly-sampled time series,” *arXiv preprint arXiv:1907.03907*, 2019.

[14] D.-E. Gratie, B. Iancu, and I. Petre, “Ode analysis of biological systems,” in *International School on Formal Methods for the Design of Computer, Communication and Software Systems*. Springer, 2013, pp. 29–62.

[15] S. M. Baker, K. Schallau, and B. H. Junker, “Comparison of different algorithms for simultaneous estimation of multiple parameters in kinetic metabolic models,” *Journal of integrative bioinformatics*, vol. 7, no. 3, pp. 254–262, 2010.

[16] C. G. Moles, P. Mendes, and J. R. Banga, “Parameter estimation in biochemical pathways: a comparison of global optimization methods,” *Genome research*, vol. 13, no. 11, pp. 2467–2474, 2003.

[17] P. Mendes and D. Kell, “Non-linear optimization of biochemical pathways: applications to metabolic engineering and parameter estimation.” *Bioinformatics (Oxford, England)*, vol. 14, no. 10, pp. 869–883, 1998.