Harnessed Causality

Essays dedicated to Maciej Koutny on the occasion of his 60th birthday

Editor:

Victor Khomenko
Newcastle University
School of Computing
Newcastle upon Tyne, NE4 5TG, UK
Email: Victor.Khomenko@ncl.ac.uk

Cover art:

Ashur Rafiev
Newcastle University
School of Computing
Newcastle upon Tyne, NE4 5TG, UK
Email: Ashur.Rafiev@ncl.ac.uk

Publisher:

Newcastle University
Newcastle upon Tyne, NE1 7RU, UK
Website: www.ncl.ac.uk

ISBN: 978-0-7017-0266-3

Publication date: 3rd September 2018

Preface

Mark with serene impartiality
The strife of things, and yet be comforted,
Knowing that by the chain causality
All separate existences are wed
Oscar Wilde, "*Humanitad"*

This Festschrift is dedicated to Maciej Koutny on the occasion of his 60th birthday. As Maciej's colleagues are undoubtedly aware, *causality* (with Petri nets used as the harness) is a prominent unifying topic of Maciej's research, and it was selected as the topic of this Festschrift.

It turns out that causality is a very complicated phenomenon. E.g. this book is 'caused' by Maciej's 60th birthday – even though its creation had preceded this remarkable occasion. This constitutes an instance of reverse temporal causality, whose formal modelling would undoubtedly require Petri nets with inhibitor arcs, contextual loops, and step semantics, not to mention the heavy mathematical artillery that Maciej uses so fluently. It is left to the reader to complete this formal model. ~~(I didn't get research funding for this.)~~

Attendees of Petri Nets 2018 (Bratislava) on the way to the conference banquet. Maciej is on the left, holding the conference banner. Photo credit: Juraj Mazari.

Apparently, the number 60 has some special significance, which I could not comprehend (being only 42, i.e. having accomplished only 70%). Hence, I have commissioned a personal research project to discover the secrets of 60, using the computational resources at my disposal. The experiment was conducted on a PC with a 64-bit Intel® Core™ i7-6700K 4.00GHz CPU with 4 cores (hyperthreaded) and 64Gb RAM. "60" was entered into Wikipedia's search box, and the results were carefully analysed. It turns out that 60 is:

- the natural number following 59 and preceding 61;
- being three times 20, it is called "three score";
- the number of seconds in a minute, and the number of minutes in an hour (a legacy of the Babylonian number system);

- a *highly composite number* (a.k.a. *anti-prime*), as it has more divisors than any smaller positive integer has;
- the sum of a pair of twin primes (29 + 31), the sum of four consecutive primes (11 + 13 + 17 + 19), adjacent to two primes (59 and 61), and the smallest number that is the sum of two odd primes in six ways (it takes a lot of courage being anti-prime surrounded by primes on all sides!);
- a *unitary perfect number*, as it is equal to the sum of its proper *unitary divisors*, excluding the number itself; a divisor d of a number n is a unitary divisor if d and n/d share no common factors;
- an *abundant number*, as the sum of its proper divisors is greater than the number itself.

There are many more interesting and potentially relevant mathematical and non-mathematical facts about 60, but the penny dropped when I noticed this:

> *The number of miles per hour an automobile accelerates to from rest (0-60) as one of the standard measurements of performance*.

The moment of truth! 60 is the standard performance measurement point, so we can now officially conclude that Maciej has been accelerating up to 60 impressively well (and continues to do so). Wikipedia also tells that the forthcoming *perfect number* is 496, which is the next goal for Maciej.

This book presents a collection of essays and papers written by Maciej's friends, colleagues, and disciples (these categories are not mutually exclusive – in fact, their overlaps are conjectured to be considerable). The contributions include personal essays as well as technical papers – both kinds of ingredients are essential for a balanced Festschrift.

I would like to thank everyone who contributed papers, supported or helped with the production of this Festschrift and with the organization of the presentation event. Finally, on behalf of all these people, I would like to congratulate Maciej on this occasion and wish him many happy returns!

Victor Khomenko
September 2018
Newcastle upon Tyne

Table of Contents

Author Index

Multiparty Session Types in Distributed Systems With Migration

Bogdan Aman and Gabriel Ciobanu

Romanian Academy, Institute of Computer Science, Iaşi, Romania
"A.I.Cuza" University, Blvd. Carol I no.11, 700506 Iaşi, Romania
bogdan.aman@gmail.com, gabriel@info.uaic.ro

Abstract. We propose a multiparty session type system for distributed systems involving timed migration between explicit locations and local communications. We use TiMo (Timed Mobility) as the process calculus over which we define the session types, and investigate certain scenarios in which processes are required to behave according to these types. The typing system ensures some properties including type soundness, communication and migration safety.

1 Introduction

In the current distributed systems there exist a high number of mobile entities moving between different locations and interacting with other entities to exchange data through communication. It is necessary to go beyond the sequential computation of the λ-calculus, even beyond the concurrent processes of the π-calculus. It is also worth to consider timed processes and explicit locations for migration. This is why we use a process calculus named TiMo able to describe migrating processes in a unified framework working with timers, explicit locations and local clocks in distributed systems [10]. We can see TiMo as a prototyping language for multi-agent systems, featuring mobility and local interaction. Multi-agent systems typically consist of a large number of agents which exhibit autonomic behaviour depending on their timeouts and actions. The mobility of agents and interaction between the agents through communication may introduce new and sometimes unexpected behaviours. Components can be highly heterogeneous, each operating at different temporal scales and having different objectives. Analysing these systems is becoming more and more necessary because they are really complex and increasingly used in various critical application domains such as e-commerce and distributed collaborative systems. Thus, it is important to have some modelling techniques which are able to describe such systems, and to reason about their behaviour in both qualitative and quantitative terms. To move towards this goal, we consider that it is important to develop a specific typing system.

In distributed systems, the behavioural types were introduced to secure the compatibility of interaction patterns among processes [23]. The behavioural type of a process specifies its expected interactions by using a simple type language,

and so determining a correct evolution. Building secure mobile environments requires solutions for a lot of problems due to the inherent timed migration and communication of the computing entities. We extend TiMo with session types able to reason over migrating and communicating processes and their timed behaviour. The idea is to use these session types to coordinate the communicating processes inside sessions (represented by the locations of our calculus) by representing the trace of the channels usage as a structured sequences of types. The migration capabilities represent the change of the current session for a user, when different sessions are active and each session is independently characterised by corresponding global types.

Global types [24] are specifications of the interactions between the processes that participate in a session. For each session represented in TiMo by the involved distributed locations, there exist a global type specifying its evolution. Also, for each participant uniquely identified by a natural number, there exist a global type that illustrates the movement of the participant between different sessions during the evolution of the whole system. Each global type associated to a session can be projected onto a set of local types describing the session from the perspective of each single participant. As done in [4], we use timed global types and local session types, but with some differences explained in what follows.

In this paper we consider systems in which processes may be engaged in different sessions during their evolutions, where each session is independently characterised by corresponding global types. Also, in our approach an arbitrary number of participants can dynamically join or leave sessions (represented as locations in our calculus) during their evolutions, and so we do not require session creation as is usually done in session types.

2 TiMo : Syntax and Semantics

In TiMo the processes can migrate between different locations of a distributed environment consisting of a number of explicit distinct locations. Timing constraints over migration and communication actions are used to coordinate processes in time and space. The passage of time in TiMo is described with respect to a global clock, while migration and communication actions are performed in a maximal parallel manner. Timing constraints for migration allow one to specify a temporal timeout after which a mobile process must move to another location. Two processes may communicate only if they are present at the same location. In TiMo, the transitions caused by performing actions with timeouts are alternated with continuous transitions. The semantics of TiMo is provided by multiset labelled transitions in which multisets of actions are executed in parallel (in one step).

Timing constraints applied to mobile processes allow us to specify how many time units are required by a process to move from one location to another. A timer in TiMo is denoted by $^{\Delta t}$, where $t \in \mathbb{N}$. Such a timer is associated with a migration action such as $go^{t}bridge$ then P indicating that process P moves to location *bridge* after t time units. A timer $^{\Delta 5}$ associated with an output

communication process $a^{\Delta 5}!\langle z\rangle$ then P else Q makes the channel a available for communication (namely it can send z) for a period of 5 time units. It is also possible to restrict the waiting time for an input communication process $a^{\Delta 4}?(x)$ then P else Q along a channel a; if the interaction does not happen before the timeout 4, the process gives up and continues as the alternative process Q.

The syntax of TiMo is given in Table 1, where the following are assumed:

- a set *Loc* of locations, a set *Chan* of communication channels, and a set *Id* of process identifiers (each $id \in Id$ has its arity m_{id});
- for each $id \in Id$ there is a unique process definition $id(u_1, \ldots, u_{m_{id}}) \stackrel{def}{=} P_{id}$, where the distinct variables u_i are parameters;
- $a \in Chan$ is a communication channel; l is a location or a location variable;
- $t \in \mathbb{N}$ is a *timeout* of an action; u is a tuple of variables;
- v is a tuple of expressions built from values, variables and allowed operations.

Processes	P, Q	::=	$a^{\Delta t}!\langle v\rangle$ then P else Q ∣	(output)
			$a^{\Delta t}?(u)$ then P else Q ∣	(input)
			$go^t l$ then P ∣	(move)
			0 ∣	(termination)
			$id(v)$ ∣	(recursion)
			$P \mid Q$	(parallel)
Located Processes	L	::=	$l[[P]]$	
Systems	N	::=	L ∣ $L \mid N$ ∣ $\mathbf{0}$	

Table 1. TiMo Syntax

Shorthand notation:

$a\,!\,\langle \boldsymbol{v}\rangle \ \rightarrow P$ will be used to denote $a^{\Delta\infty}\,!\,\langle \boldsymbol{v}\rangle$ `then` P `else stop`
$a\,?\,(\boldsymbol{u}) \ \rightarrow P$ will be used to denote $a^{\Delta\infty}\,?\,(\boldsymbol{u})$ `then` P `else stop`

The only binding constructor is $a^{\Delta t}?(u)$ then P else Q that binds the variable u within P (but *not* within Q). $fv(P)$ is used to denote the free variables of a process P (and similarly for systems); for a process definition, it is assumed that $fv(P_{id}) \subseteq \{u_1, \ldots, u_{m_{id}}\}$, where u_i are the process parameters. Processes are defined up-to an alpha-conversion, and $P\{v/u, \ldots\}$ denotes P in which all free occurrences of the variable u are replaced by v, eventually after alpha-converting P in order to avoid clashes.

Mobility is provided by a process $go^t l$ then P that describes the migration from the current location to the location indicated by l after t time units. Since l can be a variable, and so its value is assigned dynamically through communication with other processes, this form of migration supports a flexible scheme for the movement of processes from one location to another. Thus, the behaviour can adapt to various changes of the distributed environment. Processes are further constructed from the (terminated) process 0, and parallel composition $P \mid Q$. A located process $l[[P]]$ specifies a process P running at location l, and a system is built from its components $L \mid N$. A system N is well-formed if there are no free variables in N.

2.1 Operational Semantics of TiMo

The first component of the operational semantics of TiMo is the structural equivalence $\equiv$ over systems. The structural equivalence is the smallest congruence such that the equalities in Table 2 hold.

(PNULL)	$P \mid 0 \equiv P$
(NNULL)	$N \mid \mathbf{0} \equiv N$
(NCOMM)	$N \mid N' \equiv N' \mid N$
(NASSOC)	$(N \mid N') \mid N'' \equiv N \mid (N' \mid N'')$
(NSPLIT)	$l[[P \mid Q]] \equiv l[[P]] \mid l[[Q]]$

Table 2. TiMo Structural Congruence

Essentially, the role of $\equiv$ is to rearrange a system in order to apply the rules of the operational semantics given in Table 3. Using the equalities of Table 2, a given system N can always be transformed into a finite parallel composition of located processes of the form $l_1[[P_1]] \mid \ldots \mid l_n[[P_n]]$ such that no process P_i has the parallel composition operator at its topmost level. Each located process $l_i[[P_i]]$ is called a component of N, and the whole expression $l_1[[P_1]] \mid \ldots \mid l_n[[P_n]]$ is called a *component decomposition* of the system N.

The operational semantics rules of TiMo are presented in Table 3. The multiset labelled transitions of form $N \xrightarrow{\Lambda} N'$ use a multiset Λ to indicate the actions executed in parallel in one step. When the multiset Λ contains only one action λ, in order to simplify the notation, $N \xrightarrow{\{\lambda\}} N'$ is simply written as $N \xrightarrow{\lambda} N'$. The transitions of form $N \overset{t}{\rightsquigarrow} N'$ represent a time step of length t.

(Stop)	$l[[0]] \overset{\lambda}{\not\rightarrow}$ (DStop) $l[[0]] \overset{t}{\rightsquigarrow} l[[0]]$
(DMove)	if $t \geq t'$ then $l[[go^t l' \text{ then } P]] \overset{t'}{\rightsquigarrow} l[[go^{t-t'} l' \text{ then } P]]$
(Move0)	$l[[go^0 l' \text{ then } P]] \xrightarrow{l \triangleright l'} l'[[P]]$
(Com)	$l[[a^{\Delta t}!\langle v\rangle \text{ then } P \text{ else } Q]] \mid l[[a^{\Delta t'}?(u) \text{ then } P' \text{ else } Q']] \xrightarrow{\{v/u\}@l} l[[P]] \mid l[[P'\{v/u\}]]$
(DPut)	if $t \geq t' > 0$ then $l[[a^{\Delta t}!\langle v\rangle \text{ then } P \text{ else } Q]] \overset{t'}{\rightsquigarrow} l[[a^{\Delta t - t'}!\langle v\rangle \text{ then } P \text{ else } Q]]$
(Put0)	$l[[a^{\Delta 0}!\langle v\rangle \text{ then } P \text{ else } Q]] \xrightarrow{a!^{\Delta 0}@l} l[[Q]]$
(DGet)	if $t \geq t' > 0$ then $l[[a^{\Delta t}?(u) \text{ then } P \text{ else } Q]] \overset{t'}{\rightsquigarrow} l[[a^{\Delta t - t'}?(u) \text{ then } P \text{ else } Q]]$
(Get0)	$l[[a^{\Delta 0}?(u) \text{ then } P \text{ else } Q]] \xrightarrow{a?^{\Delta 0}@l} l[[Q]]$
(DCall)	if $l[[P_{id}\{v/x\}]] \overset{t}{\rightsquigarrow} l[[P'_{id}]]$ and $id(v) \overset{def}{=} P_{id}$ then $l[[id(v)]] \overset{t}{\rightsquigarrow} l[[P'_{id}]]$
(Call)	if $l[[P_{id}\{v/x\}]] \xrightarrow{id@l} l[[P'_{id}]]$ and $id(v) \overset{def}{=} P_{id}$ then $l[[id(v)]] \xrightarrow{id@l} l[[P'_{id}]]$
(DPar)	if $N_1 \overset{t}{\rightsquigarrow} N'_1$, $N_2 \overset{t}{\rightsquigarrow} N'_2$ and $N_1 \mid N_2 \overset{\lambda}{\not\rightarrow}$ then $N_1 \mid N_2 \overset{t}{\rightsquigarrow} N'_1 \mid N'_2$
(Par)	if $N_1 \xrightarrow{\Lambda_1} N'_1$ and $N_2 \xrightarrow{\Lambda_2} N'_2$ then $N_1 \mid N_2 \xrightarrow{\Lambda_1 \cup \Lambda_2} N'_1 \mid N'_2$
(DEquiv)	if $N \equiv N'$, $N' \overset{t}{\rightsquigarrow} N''$ and $N'' \equiv N'''$ then $N \overset{t}{\rightsquigarrow} N'''$
(Equiv)	if $N \equiv N'$, $N' \xrightarrow{\Lambda} N''$ and $N'' \equiv N'''$ then $N \xrightarrow{\Lambda} N'''$

Table 3. TiMo Operational Semantics

In rule (MOVE0), the process $go^0 l'$ then P migrates from location l to location l' and evolves as process P. In rule (COM), a process $a^{\Delta t}!\langle v\rangle$ then P else Q located at location$l$, succeeds in sending a tuple of values v over channel a to process $a^{\Delta t}?(u)$ then P' else Q' also located at l. Both processes continue to execute at location l, the first one as P and the second one as $P'\{v/u\}$. If a communication action has a timer equal to 0, then by using the rule (PUT0) for output action or the rule (GET0) for input action, the generic process $a^{\Delta 0} * $ then P else Q where $* \in \{!\langle v\rangle, ?(x)\}$ continues as the process Q. Rule (CALL) describes the evolution of a recursion process. The rules (EQUIV) and (DEQUIV) are used to rearrange a system in order to apply a rule. Rule (PAR) is used to compose larger systems from smaller ones by putting them in parallel, and considering the union of multisets of actions.

The rules devoted to the passing of time are starting with D. For instance, in rule (DPAR), $N_1 \mid N_2 \stackrel{\lambda}{\not\rightarrow}$ means that no action λ (i.e, an action labelled by $l' \triangleright l$, $\{v/u\}@l$, $id@l$, $go^{\Delta 0}@l$, $a?^{\Delta 0}@l$ or $a!^{\Delta 0}@l$) can be applied in the system $N_1 \mid N_2$. Negative premises are used to denote the fact that the passing to a new step is performed based on the absence of actions; the use of negative premises does not lead to an inconsistent set of rules.

A complete computational step is captured by a derivation of the form:

$$N \stackrel{\Lambda}{\longrightarrow} N_1 \stackrel{t}{\rightsquigarrow} N'.$$

This means that a complete step is a parallel execution of individual actions of Λ followed by a time step. Performing a complete step $N \stackrel{\Lambda}{\longrightarrow} N_1 \stackrel{t}{\rightsquigarrow} N'$ means that N' is directly reachable from N. If there is no applicable action ($\Lambda = \emptyset$), $N \stackrel{\Lambda}{\longrightarrow} N_1 \stackrel{t}{\rightsquigarrow} N'$ is written $N \stackrel{t}{\rightsquigarrow} N'$ to indicate (only) the time progress.

Proposition 1. *For all systems N, N' and N'',*

$$\text{if } N \stackrel{t}{\rightsquigarrow} N' \text{ and } N \stackrel{t}{\rightsquigarrow} N'', \text{ then } N' \equiv N''.$$

Proposition 1 states that the passage of time does not introduce any nondeterminism into the execution of a process.

Proposition 2. *For all systems N, N' and N'',*

$N \stackrel{(t+t')}{\rightsquigarrow} N'$ *if and only if there is a N'' such that* $N \stackrel{t}{\rightsquigarrow} N''$ *and* $N'' \stackrel{t'}{\rightsquigarrow} N'$.

Proposition 2 states that whenever a process is able to evolve for a certain time t, then it must evolve through every time moment before t; this ensures that the process evolves continuously.

3 Example

Time issues and mobility are essential in several systems in which a correct evolution depends not only on the actions taken, but also when and where the actions happen. A system may crash if an action is taken too early or too late, or in an inappropriate location. We illustrate how TIMO works by a *TravelShop* example used also in [12]. In this example, a client process attempts to pay as

little as possible for a ticket to a predefined destination. The scenario involves five locations and six processes. The role of each of the locations is as follows:

- *home* is a location where the client process starts and ends its journey;
- *travelshop* is the main location; it is initially visible to the client;
- *standard* and *special* are two locations of the service where clients can find out about the ticket prices;
- *bank* is a location where the payment is made.

The role of each of the processes is as follows:

- *client* is a process which initially resides in the *home* location, and is determined to pay for a flight after comparing two offers (standard and special) provided by the travel shop. Upon entering the travel shop, *client* receives the location of the standard offer and, after moving there and obtaining this offer, the client is given the location where a special offer can be obtained. After that, *client* moves to the bank and pays for the cheaper of the two offers, and then returns to *home*.
- *agent* first informs *client* where to look for the standard offer and then moves to *bank* in order to collect the money from the till. After that *agent* returns back to *travelshop*.
- *flightinfo* communicates the standard offer to clients as well as the location of the special offer.
- *saleinfo* communicates the special offer to clients together with the location of the bank. *saleinfo* can also accept an update of the special offer by the travel shop.
- *update* initially resides at the *travelshop* location and then migrates to *special* in order to update the special offer.
- *till* resides at the *bank* location and can either receive e-money paid in by clients, or transfer the e-money accumulated so far to *agent*.

Figure 1 describes schematically the evolution of the *TravelShop* system.

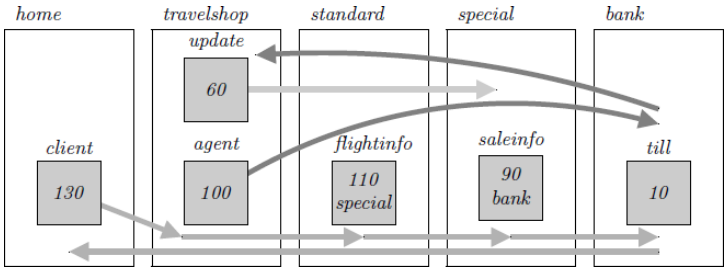


Fig. 1. The agency example

The specification of the running example in TiMo is given in Table 4.

$$
\begin{array}{l}
\mathit{TravelShop} \stackrel{\text{df}}{=} \\
\quad \mathit{home}\,[\![\, \mathit{client}(130) \,]\!] \mid \mathit{travelshop}\,[\![\, \mathit{agent}(100) \mid \mathit{update}(60) \,]\!] \mid \\
\quad \mathit{standard}\,[\![\, \mathit{flightinfo}(110, \mathit{special}) \,]\!] \mid \mathit{special}\,[\![\, \mathit{saleinfo}(90, \mathit{bank}) \,]\!] \mid \\
\quad \mathit{bank}\,[\![\, \mathit{till}(10) \,]\!] \\
\mathit{client}(\mathit{init}) \stackrel{\text{df}}{=} \\
\quad \mathtt{go}^{\Delta 5}\ \mathit{travelshop} \rightarrow \mathit{flight}\,?\,(\mathit{standardoffer}) \rightarrow \mathtt{go}^{\Delta 4}\ \mathit{standardoffer} \rightarrow \\
\quad \mathit{info}\,?\,(\mathit{p1}, \mathit{specialoffer}) \rightarrow \mathtt{go}^{\Delta 3}\ \mathit{specialoffer} \rightarrow \mathit{info}\,?\,(\mathit{p2}, \mathit{paying}) \rightarrow \\
\quad \mathtt{go}^{\Delta 6}\ \mathit{paying} \rightarrow \mathit{pay}\,!\,\langle \min\{\mathit{p1}, \mathit{p2}\} \rangle \rightarrow \\
\quad \mathtt{go}^{\Delta 4}\ \mathit{home} \rightarrow \mathit{client}(\mathit{init} - \min\{\mathit{p1}, \mathit{p2}\}) \\
\mathit{agent}(\mathit{balance}) \stackrel{\text{df}}{=} \\
\quad \mathit{flight}\,!\,\langle \mathit{standard} \rangle \rightarrow \mathtt{go}^{\Delta 10}\ \mathit{bank} \rightarrow \mathit{pay}\,?\,(\mathit{profit}) \rightarrow \mathtt{go}^{\Delta 12}\ \mathit{travelshop} \rightarrow \\
\quad \mathit{agent}(\mathit{balance} + \mathit{profit}) \\
\mathit{update}(\mathit{saleprice}) \stackrel{\text{df}}{=} \\
\quad \mathtt{go}^{\Delta 0}\ \mathit{special} \rightarrow \mathit{info}\,!\,\langle \mathit{saleprice} \rangle \rightarrow \mathtt{stop} \\
\mathit{flightinfo}(\mathit{price}, \mathit{next}) \stackrel{\text{df}}{=} \\
\quad \mathit{info}\,!\,\langle \mathit{price}, \mathit{next} \rangle \rightarrow \mathit{flightinfo}(\mathit{price}, \mathit{next}) \\
\mathit{saleinfo}(\mathit{price}, \mathit{next}) \stackrel{\text{df}}{=} \\
\quad \mathit{info}^{\Delta 10}\,?\,(\mathit{newprice})\ \mathtt{then}\ \mathit{saleinfo}(\mathit{newprice}, \mathit{next}) \\
\quad \mathtt{else}\ \mathit{info}\,!\,\langle \mathit{price}, \mathit{next} \rangle \rightarrow \mathit{saleinfo}(\mathit{price}, \mathit{next}) \\
\mathit{till}(\mathit{cash}) \stackrel{\text{df}}{=} \\
\quad \mathit{pay}^{\Delta 1}\,?\,(\mathit{newpayment}) \\
\quad \mathtt{then}\ \mathit{till}(\mathit{cash} + \mathit{newpayment})\ \mathtt{else}\ \mathit{pay}^{\Delta 2}\,!\,\langle \mathit{cash} \rangle\ \mathtt{then}\ \mathit{till}(0)\ \mathtt{else}\ \mathit{till}(\mathit{cash})
\end{array}
$$

Table 4. TiMo description of the running example.

4 Type System

Let X be a set of clocks ranging over $x_1, \ldots, x_n$ and taking values in $\mathbb{R}^{\geq 0}$, where each participant i has assigned its own clock x_i. A clock assignment $v : X \rightarrow \mathbb{R}^{\geq 0}$ returns the time of the clocks in X. We write $v + t$ for the assignment mapping all $x \in X$ to $v(x) + t$. We denote by v_0 the assignment mapping all the clocks to 0; in our approach, when an action is performed by a participant i, the clock x_i is set to the value 0. The set $\Phi(X)$ of clock constraints over X is given by:

$$\varphi ::= \mathit{true} \mid x > c \mid x = c \mid \neg\varphi.$$

An interaction in a global type consists of a send action together with a receive action, each annotated with a clock constraint. The clock constraint specifies when the action could be executed. A difference from the approach described in [4] is that we do not need to use reset predicates to specify which clocks must be reset, because each participant has only one clock assigned to him and the clock is reset after each consumed action.

The syntax for sorts S, timed global types G, and timed local types T is:

$S ::= bool \mid nat \mid \ldots \mid G_c$

$G_c ::= p \to q : \langle S \rangle \{C\}.G_c \mid G_c \mid G_c \mid \mu t.G_c \mid t \mid \mathsf{end} \qquad C ::= \{\varphi_p, \varphi_q\}$

$G_m ::= l \to l' : \langle p \rangle \{M\}.G_m \mid \mu t.G_m \mid t \mid \mathsf{end} \qquad M ::= \{\varphi_p\}$

$T_c ::= !p : \langle S \rangle \{B\}.T_c \mid ?p : \langle S \rangle \{B\}.T_c \mid \mu t.T_c \mid t \mid \mathsf{end} \qquad B ::= \{\varphi_p\}$

The sorts S include base types (bool, nat, etc.), and G_c is for communication session (for the creation of a new session of type G). In G_c, type $p \to q : \langle S \rangle \{C\}.G$ models an interaction: p sends to q a message of sort S; the session then continues as prescribed by G. The step is annotated with a time assertion $C ::= \{\varphi_p, \varphi_q\}$, where φ_p and φ_q are the clock constraints for the output and input actions, respectively. Parallel type $G_c \mid G_c$ puts in parallel the global types of several sessions. Recursive type $\mu t.G$ associates a type variable t to a recursion body G; we assume that type variables are guarded in the standard way and end occurs at least once in G. We denote by $\mathcal{P}(G)$ the set of participants involved in the sessions of the distributed system that are typed by the global type G.

By G_m we model the movement of the participant between the different sessions of the distributed system. The types have a similar explanation as in G_c, except that each one involves only one participant and not two.

In T interactions are modelled from a participant's viewpoint either as sending types $!p : \langle S \rangle \{B\}.T_c$ or receiving types $?p : \langle S \rangle \{B\}.T_c$. We denote the projection of G_c onto $p \in \mathcal{P}(G)$ by $G \downarrow_p$; the definition is standard (see [24]), except that each $\{\varphi_p, \varphi_q\}$ is projected onto the sender (resp. receiver) by keeping only the output/input part φ_p and φ_q, respectively. E.g., if $G_c = p \to q : \langle S \rangle \{\varphi_p, \varphi_q\}.G'_c$, then $G'_c \downarrow_p = !p : \langle S \rangle \{\varphi_p\}.G'_c \downarrow p$ and $G'_c \downarrow_q = ?q : \langle S \rangle \{\varphi_q\}.G'_c \downarrow q$.

Example 1. The specifications of our example from Figure 1 and Table 4 get several distinct global types. Even if usually we use numbers to represent participants, in this case we use the process names just to be easier to follow. Since we have five locations and six participants, we have two global types G_c and G_m, each one formed out of six types.

$G_{home} = G_{flightinfo} = G_{saleinfo} = G_{till} = \mathsf{end}$

$G_{travelshop} = \mu t.agent \to client : \langle string \rangle \{x_{agent} \geq 0, x_{client} \geq 0\}.t$

$G_{standard} = \mu t.$
$\quad flightinfo \to agent : \langle int, string \rangle \{x_{flightinfo} \geq 0, x_{agent} \geq 0\}.t$

$G_{special} = update \to saleinfo : \langle int \rangle .\{x_{update} \geq 0, x_{caleinf} \leq 10\}.$
$\quad \mu t.saleinfo \to client : \langle int, string \rangle \{x_{saleinfo} \geq 0, x_{client} \geq 0\}.t$

$G_{bank} = \mu t.client \to till : \langle int \rangle \{x_{client} \geq 0, x_{till} \leq 1\}.$
$\quad till \to agent : \langle int \rangle \{x_{till} \leq 2, x_{agent} \geq 0\}.t$

$G_{client} = \mu t.home \to travelshop : \langle client \rangle \{x_{client} = 5\}.$
$\quad travelshop \to standard : \langle client \rangle \{x_{client} = 4\}.$
$\quad standard \to special : \langle client \rangle \{x_{client} = 3\}.$
$\quad special \to bank : \langle client \rangle \{x_{client} = 6\}.$
$\quad bank \to home : \langle client \rangle \{x_{client} = 4\}.t$

$G_{agent} = \mu t.travelshop \to bank : \langle agent \rangle \{x_{agent} = 10\}.$
$\quad bank \to travelshop : \langle agent \rangle \{x_{agent} = 12\}.t$

$G_{update} = travelshop \to special : \langle update \rangle \{x_{update} = 0\}$

The fact that $G_{home} = \mathsf{end}$ means that there is no communication performed at location *home*, while $G_{flightinfo} = G_{saleinfo} = G_{till} = \mathsf{end}$ means that the participants $flightinfo$, $saleinfo$ and $till$ are static (they do not perform any migration). The definition of the global type $G_{travelshop}$ means that at location $travelshop$ always only the *agent* should be able to send a message containing a string to the *client*. In a similar manner, the global types $G_{standard}$, $G_{special}$ and G_{bank} describe the expected patterns of interactions inside their corresponding locations. Note that the last three types, namely G_{client}, G_{agent} and G_{update}, represent the desired behaviour of the moving participants as depicted in Figure 1 by arrows.

The typing system uses a map from shared names to either their sorts $(S, S', \ldots)$ or to a special sort $\langle G_c \rangle$ used to type various communication sessions (represented as locations in our approach). Since a type is inferred for each participating process in a certain session, we use the notation $T@l$ (called located type) to represent a local type T assigned to a participating process p in a communication session. As we deal with multiple sessions, the type of each participant is given by the value of its local clock and the local communication types for all locations of the system, and also by its mobility type. Using all these ingredients, we define the following typing system:

$$\Gamma ::= \emptyset \mid \Gamma, \langle G_m \rangle \mid \Gamma, l : \langle G_c \rangle \mid \Gamma, t : \Delta$$
$$\Delta ::= \emptyset \mid \Delta, p : (v, \{T@l\}_{l \in Loc}, G_p).$$

A sorting $(\Gamma, \Gamma', \ldots)$ is a finite map from names to sorts, and from process variables to sequences of types. Typings $(\Delta, \Delta', \ldots)$ record linear usage of session channels and migration capabilities by assigning a family of located types to a vector of session channels. We use the judgement $\Gamma \vdash P \triangleright \Delta$ saying that "under the environment Γ, process P has typing Δ".

We get some results dealing with the type preservation under structural equivalence and operational reduction. According to these results, if a well-typed process takes a reduction step of any kind, the resulting process will be also well-typed. As processes interact, the types need to follow this evolution. This dynamics is formalised by a type reduction relation $\Rightarrow$ on environments Δ.

The next theorem guarantees that if a process is well-typed, than any process congruent with it (by using the structural equivalence $\equiv$) is still well-typed.

Theorem 1. *(subject congruence)* $\Gamma \vdash N \triangleright \Delta$ *and* $N \equiv N'$ *imply* $\Gamma \vdash N' \triangleright \Delta$.

Subject reduction guarantees that typing is preserved by reductions; if a process is well-typed, then the process obtained after a computational step is also well-typed.

Theorem 2. *(subject reduction)* $\Gamma \vdash N \triangleright \Delta$ *and* $N \to N'$ *imply* $\Gamma \vdash N' \triangleright \Delta'$, *where* $\Delta = \Delta'$ *or* $\Delta \Rightarrow \Delta'$.

5 Conclusion and Related Work

A first version of TiMo was proposed by Ciobanu and Koutny in [10] as a simplified version of timed distributed π-calculus [13]. Several variants were developed during the last decade. The same co-authors defined the access permissions given by a type system in PerTiMo [12], while other authors proposed a probabilistic extension pTiMo in [14], as well as a real-time extension rTiMo [1]. Recently, a calculus for structure-aware mobile systems that combines TiMo and the bigraph model was defined in [27]. Inspired by TiMo , a flexible software platform was introduced in [9] to support the specification of distributed systems involving timed migration and safe communication [8]. Interesting properties of distributed systems described by TiMo refer to process migration, time constraints, bounded liveness and optimal reachability [2, 11]. A verification tool called TiMo@PAT [15] was developed by using Process Analysis Toolkit, an extensible platform for various model checkers. A probabilistic temporal logic called PLTM was introduced in [14] to verify complex properties making explicit reference to specific locations, temporal constraints over local clocks and multisets of actions.

A distributed calculus that provides nested multiparty and dynamically joinable sessions, and use intra-session and intra-site communications was proposed in [5]. Processes of the Conversation calculus are used in [6, 26] to treat dynamic join and leave of the participants. The dynamic join and leaving mechanism based on the multiparty session types was extended in [19] by introducing the notion of roles. In order to obtain more expressivity, a nested, higher-order multiparty session types was proposed in [18].

Multiparty session types usually guarantee progress only within a single session [20, 23, 19]. Progress for interleaved sessions was considered for modelling sessions in Java [16, 22]. However, the guarantee of progress can be given only for one single active binary session. In [21, 7] there are constructions of processes providing missing participants in dyadic sessions, which are simpler than the static interaction type system for global progress in dynamically interleaved and interfered multiparty sessions as developed in [17]. In [4], the global times are enriched with time constraints such that the multiparty session types are able to express temporal properties in a way similar to timed automata. In order to express such temporal properties, Scribble was extended with timed constrains in [25]. General conditions of progress and non-zero properties of timed communicating automata at the top of multiparty compatibility were proposed in [3].

Our approach combines these approaches by treating in an unifying framework the time constraints, dynamical join and leave mechanisms and interleaved sessions. To our knowledge, this is the first work trying to accomplish this.

Acknowledgement We enjoy very much to collaborate and be co-authors with Maciej. We send him many thanks and our best wishes.

References

1. B. Aman, G. Ciobanu. Verification of critical systems described in real-time TiMo . *Int'l Journal on Software Tools for Technology Transfer* **19**(4), 395–408 (2017).
2. B. Aman, G. Ciobanu, M. Koutny. Behavioural Equivalences over Migrating Processes with Timers. *Lecture Notes in Computer Science* **7273**, 52–66 (2012).
3. L. Bocchi, J. Lange, N. Yoshida. Meeting Deadlines Together. In *Proceedings CONCUR, LIPIcs* **42**, 283–296 (2015).
4. L. Bocchi, W. Yang, N. Yoshida. Timed Multiparty Session Types. *Lecture Notes in Computer Science* **8704**, 419–434 (2014).
5. R. Bruni, I. Lanese, H. Melgratti, E. Tuosto. Multiparty Sessions in SOC. *Lecture Notes in Computer Science* **5052**, 67–82 (2008).
6. L. Caires, H.T. Vieira. Conversation Types. *Theoretical Computer Science* **411**(51-52), 4399–4440 (2010).
7. M. Carbone, S. Debois. A Graphical Approach to Progress for Structured Communication in Web Services. *Electronic Proceedings in Theoretical Computer Science* **38**, 13–27 (2010).
8. G. Ciobanu. Finding Network Resources by Using Mobile Agents. *Intelligent Distributed Computing IV. Studies in Computational Intelligence* **315**, 305–313 (2010).
9. G. Ciobanu, C. Juravle. Flexible Software Architecture and Language for Mobile Agents. *Concurrency and Computation: Practice and Experience* **24**, 559–571 (2012).
10. G. Ciobanu, M. Koutny. Modelling and Verification of Timed Interaction and Migration. *Lecture Notes in Computer Science* **961**, 215–229 (2008).
11. G. Ciobanu, M. Koutny. Timed Mobility in Process Algebra and Petri Nets. *Journal of Logic and Algebraic Programming* **80**(7), 377–391 (2011).
12. G. Ciobanu, M. Koutny. PerTiMo : A Model of Spatial Migration with Safe Access Permissions. *Computer Journal* **58**(5), 1041–1060 (2015).
13. G. Ciobanu, C. Prisacariu. Timers for Distributed Systems. *Electronic Notes in Theoretic Computer Science* **164**(3), 81–99 (2006).
14. G. Ciobanu, A. Rotaru. A Probabilistic Logic for pTiMo . *Lecture Notes in Computer Science* **8049**, 141–158 (2013).
15. G. Ciobanu, M. Zheng. Automatic Analysis of TiMo Systems in PAT. In *Proceedings 18th ICECCS*, IEEE Computer Society, 121–124 (2013).
16. M. Coppo, M. Dezani-Ciancaglini, N. Yoshida. Asynchronous Session Types and Progress for Object-Oriented Languages. *Lecture Notes in Computer Science* **4468**, 1–31 (2007).
17. M. Coppo, M. Dezani-Ciancaglini, N. Yoshida, L. Padovani. Global Progress for Dynamically Interleaved Multiparty Sessions. *Mathematical Structures in Computer Science* **26**(2), 238–302 (2015).
18. R. Demangeon, K. Honda. Nested Protocols in Session Types. *Lecture Notes in Computer Science* **7454**, 272–286 (2012).
19. P.-M. Deniélou, N. Yoshida. Dynamic Multirole Session Types. In Proceedings POPL, 435–446 (2011).
20. M. Dezani-Ciancaglini, U. de'Liguoro. Sessions and Session Types: an Overview. *Lecture Notes in Computer Science* **6194**, 1–28 (2010).
21. M. Dezani-Ciancaglini, U. de'Liguoro, N. Yoshida. On Progress for Structured Communications. *Lecture Notes in Computer Science* **4912**, 257–275 (2008).

22. M. Dezani-Ciancaglini, D. Mostrous, N. Yoshida, S. Drossopoulou. Session Types for Object-Oriented Languages. *Lecture Notes in Computer Science* **4067**, 328–352 (2006).
23. K. Honda, V.T. Vasconcelos, M. Kubo. Language Primitives and Type Disciplines for Structured Communication-based Programming. *Lecture Notes in Computer Science* **1381**, 22–138 (1998).
24. K. Honda, N. Yoshida, M. Carbone. Multiparty Asynchronous Session Types. *Journal of the ACM* **63**(1): article 9 (2016).
25. R. Neykova, L. Bocchi, N. Yoshida. Timed Runtime Monitoring for Multiparty Conversations. *Electronic Proceedings in Theoretical Computer Science* **162**, 19–26 (2014).
26. H.T. Vieira, L. Caires, J.C. Seco. The Conversation Calculus: A Model of Service-Oriented Computation. *Lecture Notes in Computer Science* **4960**, 269–283 (2008).
27. W. Xie, H. Zhu, M. Zhang, G. Lu, Y. Fang. Formalization and Verification of Mobile Systems Calculus Using the Rewriting Engine Maude. In *Proceedings 42nd COMPSAC*, IEEE Computer Society, 213–218 (2018).

A Note on Simultaneous Choice-free Synthesis

For Maciej Koutny on the occasion of his 60'th birthday

Eike Best[1⋆], Raymond Devillers[2], Uli Schlachter[1⋆], and Harro Wimmel[1⋆]

[1] Department of Computing Science,
Carl von Ossietzky Universität Oldenburg, D-26111 Oldenburg, Germany
{eike.best,uli.schlachter,harro.wimmel}@informatik.uni-oldenburg.de
[2] Université Libre de Bruxelles,
Boulevard du Triomphe - C.P. 212, B-1050 Bruxelles, Belgium. rdevil@ulb.ac.be

Abstract. A finite set of transition systems $\{TS_1, \ldots, TS_n\}$ shall be called *simultaneously Petri net solvable* if there is a single Petri net N with different initial markings $\{M_{01}, \ldots, M_{0n}\}$, such that for every $i = 1, \ldots, n$, the reachability graph of (N, M_{0i}) is isomorphic to TS_i. We consider the special case that N must be *choice-free*, that is, contains no structural choices, and we explore how the analysis of small cycles allows to discard ill-formed systems in a pre-synthesis phase, and how they can be used to construct an adequate solution when this is possible.

Keywords: Choice-Freeness, Initial Markings, Labelled Transition Systems, Petri Nets, System Synthesis.

1 Introduction

In various papers, Maciej Koutny developed – with some colleagues – some kind of event structures allowing to model, in particular, a *simultaneity* feature. Thus, Maciej seems to like this word. This prompted the second author of this paper to consider introducing a new kind of simultaneity in our area of research. The idea is to allow, in a Petri net synthesis context, an underlying net to have a set of initial markings, rather than only a single one.

Classically, a Petri net synthesis problem starts from a finite labelled transition system (lts) and asks to build an unlabelled Petri net (possibly of some structural subclass) and an initial marking such that the corresponding reachability graph is isomorphic to the given lts. If this is not possible, one would like to have reasons why, as simple as possible, to be able to mend the given lts in order to finally reach a solvable case satisfying a more or less specific aim.

⋆ Supported by DFG (German Research Foundation) through grant Be 1267/15-1 ARS (Algorithms for Reengineering and Synthesis) and Be 1267/16-1 ASYST (Algorithms for Synthesis and Pre-Synthesis Based on Petri Net Structure Theory).

In a simultaneous synthesis, one starts from several lts and one searches for a single net and several initial markings such the corresponding reachability graphs are isomorphic to the given lts's.

As a motivating example, consider the labelled transition systems shown in Figure 1. Note that their label sets are not equal. We shall be asking the question whether or not there exists a single Petri net N with two initial markings M_{01}, M_{02}, such that the two transition systems are implemented by (N, M_{01}) and by (N, M_{02}), respectively. Figure 2 shows that this is indeed possible. In this case, we say that N, together with the two initial markings, *solves* – or *synthesises* – the two given transition systems *simultaneously*.

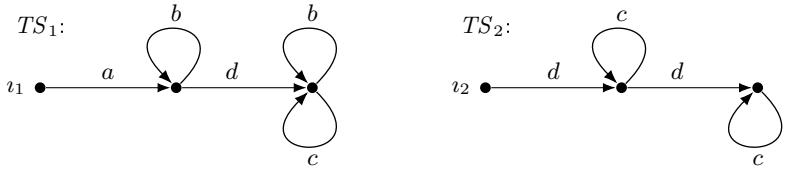


Fig. 1. Two finite transition systems TS_1 and TS_2. A simultaneous Petri net solution is sought.

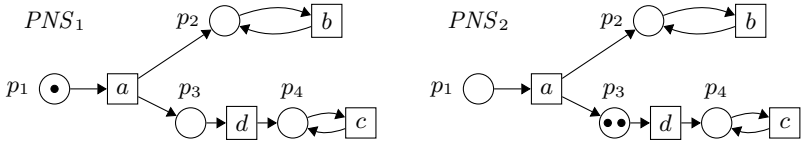


Fig. 2. PNS_1 and PNS_2 simultaneously solve TS_1 and TS_2 (respectively) choice-freely (no place has more than one output transition), since the reachability graph of PNS_i is isomorphic to TS_i $(i = 1, 2)$. Observe that PNS_1 and PNS_2 have the same set of transitions, although not all of them occur in both reachability graphs.

The choice of a specific subclass of nets to be built may considerably change the problem, and its complexity, as illustrated in Figure 3. This example shows that even if TS_3 and TS_4 are individually solvable choice-freely and a simultaneous solution exists, there does not have to be a simultaneous choice-free solution.

Definition 1. (Simultaneous) solvability of *lts* by *PNS*

An (unlabelled) Petri net N and a marking M_0 solve (or synthesise) a labelled transition system TS if the reachability graph of the Petri net system (N, M_0) is isomorphic to TS. A Petri net N and n markings $M_{01}, \ldots, M_{0n}$ solve n labelled

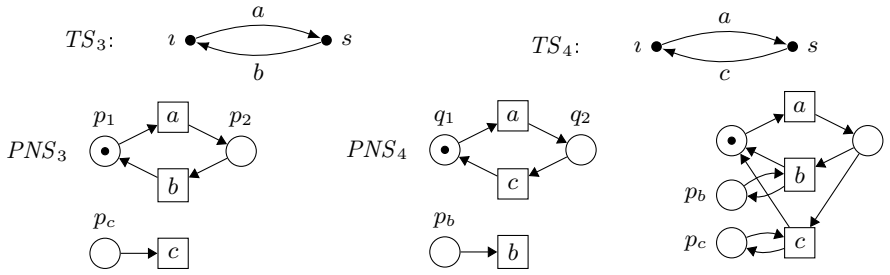


Fig. 3. Two transition systems TS_3 and TS_4, and individual choice-free solutions of them (PNS_3, respectively, PNS_4), both over the transition set $\{a, b, c\}$. There exists a simultaneous solution (bottom right): depending on whether p_b or p_c carries an initial token, TS_3 or TS_4 is solved. Later, we will demonstrate that no choice-free simultaneous solution exists.

transition systems $TS_1, \ldots, TS_n$ simultaneously if the reachability graph of the Petri net system (N, M_{0i}) is isomorphic to TS_i, for $1 \leq i \leq n$. A transition system is solvable (choice-freely solvable, or cf-solvable, for short) if it can be solved by a Petri net system (a choice-free Petri net system, respectively), and similarly for simultaneous solvability. □ 1

In order to solve such a problem, we could use an additional initial state from which n arrows (with fresh labels) lead to the n initial states of $TS_1, \ldots, TS_n$, and ask for the individual solvability of the resulting amended system. Then one considers the markings reached after performing the additional transitions, and drops the latter, in order to obtain a simultaneous solution. However, this general procedure is not very effective because the complexity of the synthesis algorithms usually grows rapidly with the number of states in the lts. Moreover it does not work for searching a choice-free solution if $n \geq 2$.

We shall thus adopt a more "distributed" strategy, considering the various lts separately before consolidating their characteristics.

2 Small cycles, semiflows and residues

Previous works on cf-synthesis exhibited the importance of Parikh vectors of sequences in an lts (counting the number of occurrences of each label), small cycles (no other cycle has a smaller Parikh vector) and short paths between two states (no shorter path exists).

A well-known property of bounded choice-free systems is that any two small cycles in its reachability graph either have the same Parikh vector or disjoint supports. This is illustrated by PNS_1 and TS_1 in Figures 2 and 1, where there are two disjoint Parikh vectors of small cycles (b and c).

This is no longer true for unbounded systems, as illustrated by Figure 4.

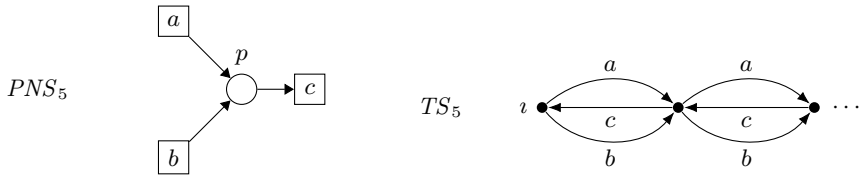


Fig. 4. An unbounded cf-net (on the left) and its reachability graph (on the right) with intersecting small cycles ac and bc.

A first result we obtained is that the disjunction property extends to small cycles in several bounded reachability graphs of a same cf-net.

Theorem 1. SMALL CYCLES IN A SIMULTANEOUS SYNTHESIS

Let (N, M_0^1) and (N, M_0^2) be two bounded systems arising from the same choice-free net N. Let $M_1[\sigma_1\rangle M_1$ be a small cycle in the reachability graph of the former and $M_2[\sigma_2\rangle M_2$ be a small cycle in the reachability graph of the latter. Then σ_1 and σ_2 are either Parikh-equivalent or disjoint.

This allows to rule out the simultaneous solvability of systems TS_3 and TS_4 in Figure 3 since the two small cycles ab and ac are neither Parikh-equivalent nor disjoint.

In fact, it can also be proved that the Parikh vectors of small cycles occurring in the finite reachability graph of some cf-net are minimal semi-flows of the latter, and the next series of new results concern semiflows. In formulating them, we exploit a classic residue operation defined for sequences and extended to T-vectors.

Let $\tau, \sigma \in T^*$ be two sequences over some label set T. The (left) residue of τ with respect to σ, denoted by $\tau \stackrel{\bullet}{-} \sigma$, arises from cancelling successively in τ the leftmost occurrences of all symbols from σ, read from left to right. Inductively: $\tau \stackrel{\bullet}{-} \varepsilon = \tau$; $\tau \stackrel{\bullet}{-} t = \tau$ if $t \notin \mathit{supp}(\tau)$; $\tau \stackrel{\bullet}{-} t$ is the sequence obtained by erasing the leftmost t in τ if $t \in \mathit{supp}(\tau)$; and $\tau \stackrel{\bullet}{-} (t\sigma) = (\tau \stackrel{\bullet}{-} t) \stackrel{\bullet}{-} \sigma$. Said differently, $\tau \stackrel{\bullet}{-} \sigma$ is τ with, for each $t \in T$, the first $\min(\mathcal{P}(\tau)(t), \mathcal{P}(\sigma)(t))$ occurrences of t dropped. Let $\sigma \in T^*$ and $\Phi \in \mathbb{N}^T$: $\sigma \stackrel{\bullet}{-} \Phi$ is the sequence obtained from σ by cancelling the $\min(\mathcal{P}(\tau)(t), \Phi(t))$ leftmost occurrences of t for each $t \in T$. Let $\Phi, \Psi \in \mathbb{N}^T$: $\Psi \stackrel{\bullet}{-} \Phi$ is the T-vector such that, for each $t \in T$, $(\Psi \stackrel{\bullet}{-} \Phi)(t) = \max(\Psi(t) - \Phi(t), 0) = \Psi(t) - \min(\Psi(t), \Phi(t))$. The consistency between these various forms of residues arises from the observation that $\mathcal{P}(\tau \stackrel{\bullet}{-} \sigma) = \mathcal{P}(\tau \stackrel{\bullet}{-} \mathcal{P}(\sigma)) = \mathcal{P}(\tau) \stackrel{\bullet}{-} \mathcal{P}(\sigma)$. Other interesting properties about residues are that $(\sigma \stackrel{\bullet}{-} \sigma_1) \stackrel{\bullet}{-} \sigma_2 = \sigma \stackrel{\bullet}{-} (\mathcal{P}(\sigma_1) + \mathcal{P}(\sigma_2)) = (\sigma \stackrel{\bullet}{-} \sigma_2) \stackrel{\bullet}{-} \sigma_1$ and $\sigma\sigma' \stackrel{\bullet}{-} \sigma = \sigma'$.

In the reachability graph of a cf-system (N, M_0), it is known that all the short paths from M_0 to a reachable marking M have the same Parikh vector, called the distance Δ_M of M from the initial marking.

Theorem 2. PROPERTIES OF SEMIFLOWS IN CF-SYSTEMS

Let (N, M_0) be a cf-system, M any reachable marking and Ψ a semiflow of N. Then,

1. *$\Delta_M \not\geq \Psi$ (short distance property);*
2. *for some reachable marking M', we have $\Delta_{M'} = \Delta_M \stackrel{\bullet}{-} \Psi$ (reduced distance property);*
3. *if there is a cycle around M with Parikh vector Γ disjoint from Ψ, there is also a cycle around M' with Parikh vector Γ when $\Delta_{M'} = \Delta_M \stackrel{\bullet}{-} \Psi$ (early cycle property).*

This may be the base for the development of very discriminating structural checks during an (individual or simultaneous) cf-presynthesis. For instance, Figure 5 illustrates the use of the short distance property, Figure 6 illustrates the reduced distance property, and Figure 7 illustrates the early cycle property

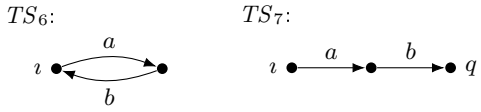


Fig. 5. Two transition systems TS_6 and TS_7 which are individually, but not simultaneously, solvable by a choice-free net. In TS_6, there is a small cycle $\imath[ab\rangle\imath$, while in TS_7, $\Delta_q = \mathcal{P}(ab)$ for a state $q \neq \imath$, so that $\neg(\Delta_q \not\geq \mathcal{P}(ab))$, contradicting the short distance property.

3 Simultaneous cf-synthesis

If a set of transition systems has a simultaneous cf-solution, then necessarily the pre-synthesis, which checks the properties described above, succeeds, and all given transition systems have an individual solution. Conversely:

1. If the pre-synthesis fails, we know there is no solution and we should not enter the proper synthesis phase.
2. If the pre-synthesis succeeds, it may still happen that some (or all) given transition systems have no individual solution.
3. If the pre-synthesis succeeds and all given transition systems have an individual solution, it may still happen that there is no simultaneous solution.

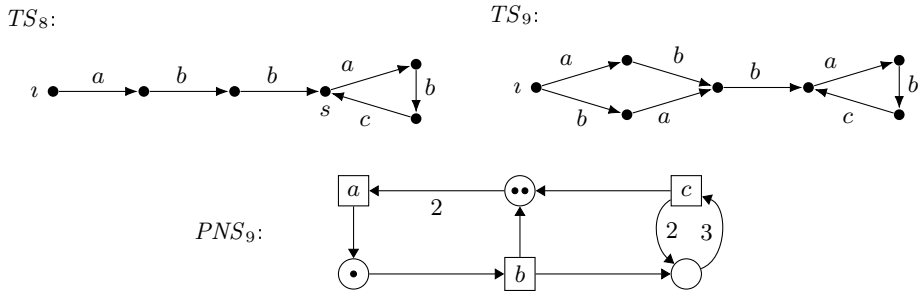


Fig. 6. TS_8 does not satisfy the reduced distance property since $\Delta_s = \mathcal{P}(abb)$, there is a small cycle $s[abc\rangle s$, but there is no state s' such that $\Delta_{s'} = \mathcal{P}(abb \stackrel{\bullet}{-} abc) = \mathcal{P}(b)$. Hence, it is not (individually) cf-solvable. By contrast, TS_9 satisfies the reduced distance property and is cf-solvable, as illustrated by PNS_9.

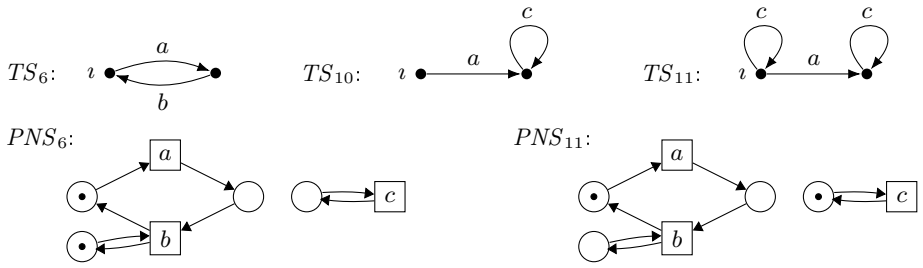


Fig. 7. The two transition systems TS_{10} and TS_{11} are individually cf-solvable. TS_{10} is not simultaneously cf-solvable with TS_6, since there is a loop c after a, but not initially while $a \stackrel{\bullet}{-} ab = \varepsilon$, contravening the early cycle property. By contrast, TS_{11} is simultaneously cf-solvable with TS_6, as shown by PNS_6 and PNS_{11}.

4. If there is one, it is not always trivial to build it from the obtained individual ones. This is why we need to proceed adequately, as explained by means of Figures 8 and 9.

If there is a simultaneous solution, the underlying net must have as (minimal) semiflows the Parikh vectors of all the (small) cycles in all the given lts (possibly more). The classic pre-synthesis procedures developed in previous papers provide for each given lts TS_i the set $\mathcal{G}_i$ of all the Parikh vectors of small cycles. We may then define $\mathcal{G} = \cup_i \mathcal{G}_i$. The synthesis procedures presented in the same papers build for each TS_i (when it is possible) a cf-solution PNS_i which is automatically compatible with all the semiflows in $\mathcal{G}_i$. It is however possible to adapt those synthesis procedures in order to build (when possible) for each TS_i a cf-solution PNS'_i compatible with all the semiflows in $\mathcal{G}$. If this is not possible, we know there is no simultaneous cf-solution. The main result we obtain is then:

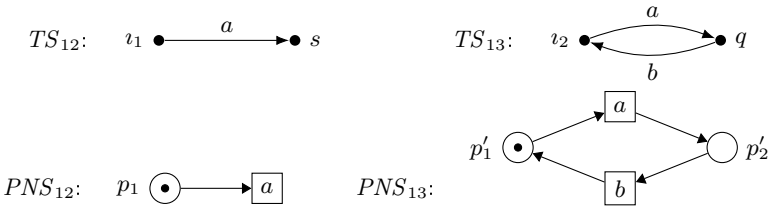


Fig. 8. Two transition systems TS_{12} and TS_{13} and two individual cf-solutions: it is not easy to deduce a simultaneous solution (if possible).

Proposition 1. TRANSITION HOMOGENISATION AND MERGING

Let $PNS'_1, \ldots, PNS'_n$ be solutions of $TS_1, \ldots, TS_n$ (respectively), all with the same transition set T, while respecting all the semiflows in $\mathcal{G}$; then in their synchronisation it is possible, for all $i \in \{1, \ldots, m\}$, to choose a marking in order to generate TS_i.

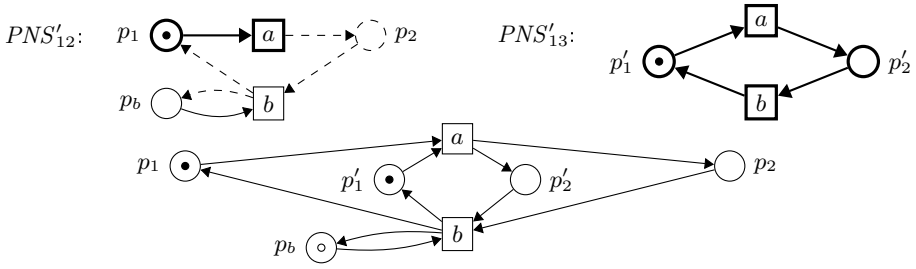


Fig. 9. Homogenised individual cf-solutions of TS_{12} and TS_{13}. Compared with PNS_{12} in Figure 8, the net PNS'_{12} also contains a transition b with a token-free input place p_b (in order to make the label sets equal), as well as dashed arrows and a new place p_2. The latter create a semiflow which corresponds to the small cycle of TS_{13} (and thus to the semiflow (a, b) of PNS_{13} and PNS'_{13}). The simultaneous solution arises by transition synchronisation (with markings according to Proposition 1) and is shown at the bottom of the figure. Depending on whether a token is absent or not on p_b, TS_{12} or TS_{13} is solved.

Acknowledgement The authors would like to thank Professor M. Koutny. It is on the occasion of the preparation of a contribution for celebrating his 60'th birthday that all these results (plus some more) were obtained, and it was fun!

Interval Semantics Generalized

Thomas Chatain[1], Stefan Haar[1], Loïc Paulevé[2], and Stefan Schwoon[1]

[1] LSV, ENS Paris-Saclay, INRIA, CNRS, France
[2] CNRS & LRI UMR 8623, Univ. Paris-Sud – CNRS,
Université Paris-Saclay, 91405 Orsay, France

1 Introduction

This article relates ideas published in [12], and which originated in joint work with Maciej Koutny in [11]. In the latter, we had been investigating the possibilities of joint firing of transitions in contextual Petri nets, finding a firing rule that goes beyond step semantics. The resulting *interval semantics* works under the assumption that for every transition, some time passes between the verification of input tokens, and their actual consumption (+ the production of output tokens): an interval during which the same tokens may authorize firings in parallel. By accounting for this, the interval semantics allows all transitions possible under step and individual semantics, and further firings as well, making new states and behaviours accessible.

Here, we transfer these ideas to the domain of *boolean networks*. Note that all proofs can be found in [12]. Boolean networks model dynamics of systems where several components (or nodes) interact. They specify for each node an update function to determine its next value according to the configuration (global state) of the network. Boolean networks are widely used to model dynamics of biological networks, such as gene networks and cellular signalling pathways.

The scheduling of nodes updates is known to have a strong influence on the reachable configurations of the networks. The relationships between different updating modes received a lot of attention both in transition-centered models of networks such as Petri nets on the one hand, [15,6,8,28,29] (in particular when read arcs are used to model finely the update mechanisms), and function-centered models such as cellular automata [23,5] and Boolean networks [16,26,13,3,19,20], on which this article is focused. Notice that transformations exist from BNs to Petri nets [24,9,10] showing the strong relationship between the two formalisms.

The updating modes usually considered for Boolean networks are the following: the *synchronous* updating, where all nodes are updated simultaneously, generating a deterministic dynamics; the (fully) *asynchronous* updating, where only one node can be updated at a time, this node being chosen non-deterministically. Asynchronous updating generates non-deterministic dynamics due to the different ordering of updates, which can be interpreted as considering in the same model different update speeds. Then, the *generalized asynchronous* updating allows all the combinations of simultaneous updates subsets of nodes, ranging from single nodes (matching asynchronous transitions) to the full set of nodes (matching synchronous transitions). Other updating modes like sequential or

block sequential have also been considered in the literature on cellular automata and Boolean networks [5,3], and usually lead to transitions allowed by the generalized asynchronous updating.

When a Boolean network aims at modelling a dynamical system having time features, as it is typically the case for biological systems, the choice of the update mode is crucial as it determines the set of configurations reachable from a given initial configuration. In applications, it is usual to assess the accordance of a Boolean network with the concrete system by checking if the observed configurations are indeed reachable in the Boolean network. Whenever it is not the case, it typically means that the designed Boolean functions do not model the system correctly, and thus should be modified before further model analysis.

Given that only partial information is available in general on the actual velocity of different nodes and transitions in the concrete system, a common approach is to choose the most general updating mode, i.e., the one entailing as few constraints as possible regarding the unknown scheduling of node updates. In such a setting, and because we abstract away many parameters of the system dynamics, we expect that the Boolean network models an over-approximation of possible transitions, *i.e.*, that any reachable configuration in the concrete system should be reachable in the Boolean network.

In this paper, we show that the generalized asynchronous updating, subsuming synchronous and asynchronous updating, can miss transitions, hence reachable configurations, which correspond to particular, but plausible, behaviours. Thus, the resulting analysis can be misleading on the absence of some behaviours, notably regarding the reachability of attractors (configurations reachable on the long-run), and may lead to reject valid models.

The proposed updating mode for Boolean networks, called *interval semantics*, aims at enabling the reachability of configurations by considering a novel, generalized update scheduling policy. Essentially, the interval semantics considers the possibility of a delay between the triggering of the update of a node, and its actual completion: this models the case of species whose value changes can be slow.

The interval semantics can be expressed as the asynchronous updating over a Boolean network which encodes the decoupling of update triggering and update application. Therefore, our approach allows the definition of an asynchronous Boolean network which simulates the general asynchronous dynamics of the original Boolean network, while including additional and plausible behaviours, and still preserving important dynamical constraints on fixpoints and causality of transitions: the fixpoints of the interval semantics form a one-to-one relationship with the fixpoints of the generalized asynchronous updating.

We illustrate the benefit of the interval semantics on a small example of Boolean network, which is actually embedded in many models of biological networks (e.g., [17,18,27]). Therefore, the analysis of dynamics of these biological models can be substantially impacted by considering the interval semantics.

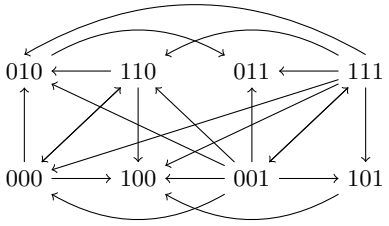
Outline. Sect. 2 gives the definitions of Boolean networks and their synchronous, asynchronous, and generalized asynchronous updating. Sect. 3 gives a motivating

$f_1(x) \triangleq \neg x_2$

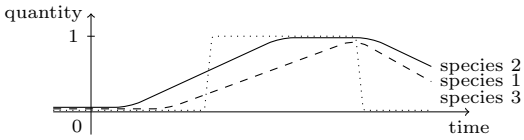
$f_2(x) \triangleq \neg x_1$

$f_3(x) \triangleq \neg x_1 \wedge x_2$

(a)



(b)



(c)

Fig. 1. (a) Example BN f of dimension 3; (b) Transition relations between configurations in $\mathbb{B}^3$ according to the generalized asynchronous updating of f; (c) A possible evolution of the quantities of the species (species 1 in dashed line, species 2 plain, species 3 dotted).

example showing the limit of the generalized asynchronous updating. Sect. 4 introduces the interval semantics for Boolean networks by providing an encoding as an asynchronous Boolean network and by establishing the relation with the generalized asynchronous updating and consistency criteria. Further extensions of the interval semantics are discussed in Sect. 5. Finally, Sect. 6 discusses the relevance of the results for the analysis of biological models, and suggests further work.

2 Definitions

We write $\mathbb{B} = \{0, 1\}$ and $[n] = \{1, \ldots, n\}$. Given a *configuration* $x \in \mathbb{B}^n$ and $i \in [n]$, we denote x_i the i^{th} component of x, so that $x = x_1 \ldots x_n$. Given two configurations $x, y \in \mathbb{B}^n$, the components that differ are noted $\Delta(x, y) \triangleq \{i \in [n] \mid x_i \neq y_i\}$.

Definition 1 (Boolean network). *A Boolean network (BN) of dimension n is a collection of functions $f = \langle f_1, \ldots, f_n \rangle$ where $\forall i \in [n], f_i : \mathbb{B}^n \to \mathbb{B}$.*

Given $x \in \mathbb{B}^n$, we write $f(x)$ for $f_1(x) \ldots f_n(x)$.

Fig. 1 (a) shows an example of BN of dimension 3.

When modelling biological systems, each node $i \in [n]$ usually represents a biochemical species, being either active (or present, value 1) or inactive (or absent, value 0). Each function f_i indicates how the evolution of the value of i is influenced by the current value of other components. However, this description

can be interpreted in several ways, therefore several updating modes coexist for BNs, depending on the assumptions about the order in which the evolutions predicted by the f_i apply.

The *asynchronous updating* assumes that only one component is updated at each time step. The choice of the component to update is non deterministic.

Definition 2 (Asynchronous updating). *Given a BN f, the binary relation $\xrightarrow[\text{async}]{f} \subseteq \mathbb{B}^n \times \mathbb{B}^n$ is defined as:*

$$x \xrightarrow[\text{async}]{f} y \overset{\Delta}{\iff} \exists i \in [n], \Delta(x,y) = \{i\} \wedge y_i = f_i(x) \ .$$

We write $\xrightarrow[\text{async}]{f}{}^*$ for the transitive closure of $\xrightarrow[\text{async}]{f}$.

The *synchronous updating* can be seen as the opposite: *all* components are updated at each time step. This leads to a purely deterministic dynamics.

Definition 3 (Synchronous updating). *Given a BN f, the binary relation $\xrightarrow[\text{sync}]{f} \subseteq \mathbb{B}^n \times \mathbb{B}^n$ is defined as:*

$$x \xrightarrow[\text{sync}]{f} y \overset{\Delta}{\iff} x \neq y \wedge \forall i \in [n], y_i = f_i(x) \ .$$

By forcing all the components to evolve synchronously, the synchronous updating makes a strong assumption on the dynamics of the system. In many concrete cases, for instance in systems biology, this assumption is clearly unrealistic, at least because the components model the quantity of some biochemical species which evolve at different speeds.

As a result, the synchronous updating fails to describe some behaviours that are possible in the asynchronous mode, such as the transition $010 \to 011$ represented in Fig. 1 (b) which represents the activation of species 3 when species 1 is inactive and species 2 is active ($f_3(010) = 1$). There are also transitions which are possible in the synchronous but not in the asynchronous updating, for instance $000 \to 110$. Remark that 110 is not even reachable from 000 in the asynchronous updating.

The *generalized asynchronous updating* generalizes both the asynchronous and the synchronous ones: it allows updating synchronously any nonempty subset of components.

Definition 4 (Generalized asynchronous updating). *Given a BN f, the binary relation $\xrightarrow{f} \subseteq \mathbb{B}^n \times \mathbb{B}^n$ is defined as:*

$$x \xrightarrow{f} y \overset{\Delta}{\iff} x \neq y \wedge \forall i \in \Delta(x,y) : y_i = f_i(x) \ .$$

Clearly, $x \xrightarrow[\text{async}]{f} y \Rightarrow x \xrightarrow{f} y$ and $x \xrightarrow[\text{sync}]{f} y \Rightarrow x \xrightarrow{f} y$. The converse propositions are false in general. It is even false that $x \xrightarrow{f} y$ implies $x \xrightarrow[\text{async}]{f} y \vee x \xrightarrow[\text{sync}]{f} y$.

Note that we forbid "idle" transitions ($x \to x$) whatsoever the updating mode.

For each node $i \in [n]$ of the BN, f_i typically depends only on a subset of nodes of the network. The *influence graph* of a BN (also called interaction or causal graph) summarizes these dependencies by having an edge from node j to i if f_i depends on the value of j. Formally, f_i depends on x_j if there exists a configuration $x \in \mathbb{B}^n$ such that $f_i(x)$ is different from $f_i(x')$ where x' is x having solely the component j different ($x'_j = \neg x_j$). Moreover, assuming $x_j = 0$ (therefore $x'_j = 1$), we say that j has a positive influence on i (in configuration x) if $f_i(x) < f_i(x')$, and a negative influence if $f_i(x) > f_i(x')$. It is possible that a node has different signs of influence on i in different configurations (leading to non-monotonic f_i). It is worth noticing that different BNs can have the same influence graph.

Definition 5 (Influence graph). *Given a BN f, its* influence graph $G(f)$ *is a directed graph* $([n], E_+, E_-)$ *with* positives *and* negatives *edges such that*

$$(j,i) \in E_+ \overset{\Delta}{\iff} \exists x, y \in \mathbb{B}^n : \Delta(x,y) = \{j\}, x_j < y_j, f_i(x) < f_i(y)$$

$$(j,i) \in E_- \overset{\Delta}{\iff} \exists x, y \in \mathbb{B}^n : \Delta(x,y) = \{j\}, x_j < y_j, f_i(x) > f_i(y)$$

A (directed) cycle composed of edges in $E_+ \cup E_-$ is said positive *when it is composed by an even number of edges in E_- (and in number of edges in E_+), otherwise, it is* negative.

The influence graph is an important object in the literature of BNs [25,2]. For instance, many studies have shown that one can derive dynamical features of a BN f by the sole analysis of its influence graph $G(f)$. Importantly, the presence of negative and positive cycles in the influence graph, and the way they are intertwined can help to determine the nature of attractors (that are the smallest sets of configurations closed by the transition relationship) [22], and derive bounds on the number of fixpoints and attractors a BN having the same influence graph can have [21,1,4].

3 Motivating example

Fig. 1 shows an example of BN of dimension 3 and $\xrightarrow{f}$ relation between configurations. The BN shows that the quantity of 3 increases when 1 is absent and 2 is present. In any scenario starting from 000 where 3 eventually increases, 2 has to increase to trigger the increase of 3. Hence, according to the generalized asynchronous updating, the only transition which represents an increase of 3 is $010 \to 011$. After this, no transition is possible.

But, assuming the BN abstracts continuous evolution of quantities, the following scenario, pictured in Fig. 1(c), becomes possible: initially, the absence of species 1 causes an increase of the quantity of species 2, represented in plain line on the figure. Symmetrically, the absence of species 2 causes an increase of the quantity of species 1 (dashed line). This corresponds to the evolution described

by the arrow 000 → 110 in Fig. 1(b) and leads to a (transient) configuration where species 1 and 2 are present.

Assume that 1 and 2 increase slowly. After some time, however, the quantity of 2 becomes sufficient for influencing positively the quantity of 3, while there is still too little of species 1 for influencing negatively the quantity of 3. Species 3 can then increase. In the scenario represented in the figure, 3 (dotted line) increases quickly, and then 1 and 2 continue to increase. In summary, the quantity of species 3 increased from 0 to 1 *during* the increase of 1 and 2, which was not predicted by the generalized asynchronous updating (Fig. 1(b)).

One could argue that in this case, one should better consider more fine-grained models, for instance by allowing more than binary values on nodes in order to reflect the different activation thresholds. However, the definition of the refined models would require additional parameters (the different activation thresholds) which are unknown in general. Our goal is to allow capturing these behaviours already in the Boolean abstraction, so that any refinement would remove possible transitions, and not create new ones.

4 Interval Semantics for Boolean Networks

Interval semantics has been proposed for Petri nets in [11] with the aim at generalizing the notion of steps [14], that are sets of transitions that can be simultaneously fired. The interval semantics adds the possibility to trigger, within a single step, transitions that become enabled by the firing transitions. The motivating example given in the previous section illustrates how this semantics can augment the set of reachable configurations.

In this section, we propose an encoding of the interval semantics for Boolean networks as an asynchronous Boolean network. Essentially, each node $i \in [n]$ is decoupled in two nodes: a "write" node storing the next value $(2i-1)$ and a "read" node for the current value $(2i)$. The decoupling is used to store an ongoing value change, while other nodes of the system still read the current (to be changed) value of the node. A value change is then performed according to the automaton given in Fig. 2: assuming we start in both write and read node with value 0, if $f_i(x)$ is true, then the write node is updated to value 1. The read node is updated in a second step, leading to the value where both write and read nodes are 1. Then, if $f_i(x)$ is false, the write node is updated first, followed, in a second stage by the update of the read node.

Once the write node $(2i-1)$ has changed its value, it can no longer revert back until the read node has been updated. Hence, if $f_i(x)$ become false in the intermediate value 10, the read node will still go through value 1 (possibly enabling transitions) before the write node can be updated to 0, if still applicable.

4.1 Encoding

From the automaton given in Fig. 2, one can derive Boolean functions for the write $(2i-1)$ and read $(2i)$ nodes. It results in the following BN $\tilde{f}$, encoding the interval semantics for the BN f:

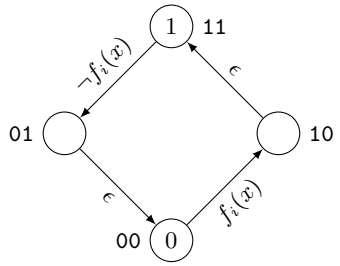


Fig. 2. Automaton of the value change of a node i in the interval semantics. The states marked 0 and 1 represents the value 0 and 1 of the node. The labels $f_i(x)$ and $\neg f_i(x)$ on edges are the conditions for firing the transitions; ϵ indicates that the transitions can be done without condition. The states are labeled by the corresponding values of nodes $(2i-1)(2i)$ in our encoding.

Definition 6 (Interval semantics for Boolean networks). *Given a BN f of dimension n, $\tilde{f}$ is a BN of dimension $2n$ where $\forall i \in [n]$,*

$$\tilde{f}_{2i-1}(z) \stackrel{\Delta}{=} (f_i(\gamma(z)) \wedge (\neg z_{2i} \vee z_{2i-1})) \vee (\neg z_{2i} \wedge z_{2i-1})$$
$$\tilde{f}_{2i}(z) \stackrel{\Delta}{=} z_{2i-1}$$

where $\gamma(z) \in \mathbb{B}^n$ is defined as $\gamma(z)_i \stackrel{\Delta}{=} z_{2i}$ for every $i \in [n]$.
Given $x \in \mathbb{B}^n$, $\alpha(x) \in \mathbb{B}^{2n}$ is defined as $\alpha(x)_{2i-1} = \alpha(x)_{2i} \stackrel{\Delta}{=} x_i$ for every $i \in [n]$.
A configuration $z \in \mathbb{B}^{2n}$ is called consistent *when $\alpha(\gamma(z)) = z$.*

The function $\gamma : \mathbb{B}^{2n} \to \mathbb{B}^n$ maps a configuration of the interval semantics to a configuration of the BN f by projecting on the read nodes. The function $\alpha : \mathbb{B}^n \to \mathbb{B}^{2n}$ gives the interval semantics configuration of a configuration of the Boolean network f, where the read and write nodes have a consistent value.

Example 1. Applied to the BN f of Fig. 1, we obtain the following possible sequence of asynchronous iterations of $\tilde{f}$:

$$00\,00\,00 \xrightarrow[\text{async}]{\tilde{f}} 10\,00\,00 \xrightarrow[\text{async}]{\tilde{f}} 10\,10\,00 \xrightarrow[\text{async}]{\tilde{f}} 10\,11\,00$$
$$\xrightarrow[\text{async}]{\tilde{f}} 10\,11\,10 \xrightarrow[\text{async}]{\tilde{f}} 10\,11\,11 \xrightarrow[\text{async}]{\tilde{f}} 11\,11\,11$$

Therefore, with the interval semantics, the configuration 111 of f is reachable from 000, contrary to the generalized asynchronous semantics. This is due to the decoupling of the update of node 1: the activation of 1 is delayed which allows activating node 3 beforehand.

Any transition of the generalized asynchronous semantics can be simulated by the interval semantics.

Theorem 1. *For all* $x, y \in \mathbb{B}^n$,

$$x \xrightarrow{f} y \Rightarrow \alpha(x) \xrightarrow[\text{async}]{\tilde{f}}{}^{*} \alpha(y) \ .$$

4.2 Consistency

The above theorem shows that the asynchronous semantics of the Boolean network encoding our interval semantics can reproduce any behaviour of the generalized asynchronous semantics. The aim of this section is to show that the interval semantics still preserves important constraints of the BN on its dynamics. In particular, we show the one-to-one relationship between the fixpoints of the BN and its encoding for interval semantics; and that the influences are preserved with their sign.

Lemma 1 states that from any configuration of encoded BN, one can always reach a configuration which corresponds to a configuration of the original BN (i.e., a configuration $z \in \mathbb{B}^{2n}$ such that $\alpha(\gamma(z)) = z$):

Lemma 1 (Reachability of consistent configurations). *For any* $z \in \mathbb{B}^{2n}$ *such that* $\alpha(\gamma(z)) \neq z$, $\exists y \in \mathbb{B}^n : z \xrightarrow[\text{async}]{\tilde{f}}{}^{*} \alpha(y)$.

The one-to-one relationship between fixpoints of f and fixpoints of $\tilde{f}$ is given by the following lemma:

Lemma 2 (Fixpoint equivalence). $\forall x \in \mathbb{B}^n$, $f(x) = x \Rightarrow f(\alpha(x)) = \alpha(x)$; *and* $\forall z \in \mathbb{B}^{2n}$, $\tilde{f}(z) = z \Rightarrow \alpha(\gamma(z)) = z \wedge f(\gamma(z)) = \gamma(z)$.

5 Further Extensions

Our interval semantics decouples the update of a node in order to allow the interleaving of transitions during the interval when the next value has been computed (write node) but not applied yet (read node still with the before-update value). This also implies that, during this interval, the other nodes have access only to the before-update value. A third feature of the interval semantics is the enforcement of the update application: once an update is triggered (write node gets a different value than the read node), no further update on the same node is possible until the update has been applied. Thus, if for instance the update triggers a change of value from 0 to 1, the interval semantics guarantees that the read node will eventually have the value 1.

These two aspects, restricted access to the before-update value of nodes and enforcement of update application, were essentially motivated by our choice that our interval semantics should simulate the synchronous update of nodes used in the classical synchronous and generalized asynchronous semantics, as stated in Theorem 1. However, one could go further and consider extended interval semantics which relax either the restricted access to the before-update value of nodes, or the enforcement of update application, or both. We will see that these relaxations of our interval semantics still preserve the consistency properties stated in Sect. 4.2.

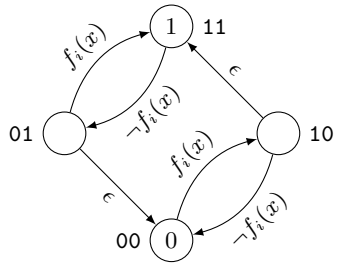


Fig. 3. Automaton of the value change of a node i in the extended interval semantics where the update can be canceled if $f_i(x)$ changes of value during the interval of update. Notations follow the ones of Fig. 2

5.1 Update cancellation

The relaxation of the enforcement of update application can be interpreted as the ability to cancel an ongoing update when f_i changes of value during the interval of update. This can be described by the automaton of Fig. 3, and encoded by removing $\neg z_{2i}$ and z_{2i-1} from the definition of $\tilde{f}_{2i-1}$ in Def. 6.

Theorem 1 and the lemmas in previous section are still verified with update cancellation.

5.2 Reading from either the before-update or after-update values

In terms of modeling, the restriction to before-update values can be seen as an asymmetry in the consideration of transitions: the resource modified by the transition is still available during the interval of update, whereas the result is only available once the transition finished. When modelling biological systems, it translates into considering only species which are slow to reach their activity threshold.

Actually, the choice of whether the before-update, after-update or both values are available during the update may be done according to the knowledge of the modeled system. Our construction can easily be adapted for giving access, depending on the node, to the after-update value instead of the before-update value. For instance, if the node i should follow closely value changes of node i, then node j should access the after-update value (write node) of i, whereas, as in our motivating example, if i is slow to update compared to j, node j should access the before-update value (read node) of i.

Finally, one could also consider a more permissive symmetric version which would allow the access of both before-update and after-update values. This choice may be very reasonable when not much is known about the system, for instance about the relative speed of the nodes.

5.3 Comparison with multi-valued networks

Multi-valued networks [7] are an extension of Boolean networks where the domain of each node $i \in [n]$ ranges over a finite discrete ordered domain $\mathbb{D}_i$. The value changes of the nodes are specified using a function $g_i : \mathbb{D}_1 \times \cdots \times \mathbb{D}_n \to \{-, 0, +\}$ which determines the direction of the value change.

Thus, a strong constraint of this semantics is that value changes are always unitary: a transition will either change the value to the smallest higher one, or the highest smaller one, if it exists. However, one can remark that the automaton modeling the value change with the interval semantics (Fig. 2) does not satisfy such a constraint, and hence cannot be encoded as a single multi-valued node.

6 Discussion

As shown in our motivating example in Sect. 3, the interval semantics can enable the reachability of configurations that are not allowed in other updating modes, notably asynchronous or generalized asynchronous. This can be problematic when expecting Boolean networks to produce an over-approximation of reachable configurations due to the abstraction of parameters related to speed and activity threshold of components, as it is usually assumed when modelling biological networks. It appears that the Boolean network in Sect. 3 is embedded in numerous actual models of biological networks (e.g., [17,18,27]). Therefore, the result of analysis of the transient dynamics of these models may be deeply impacted by using the interval semantics, which had never been considered previously.

The transitions enabled by the interval semantics are due to nodes which update slowly: whenever committed to a value change, in the meantime of the update application, the other nodes of the network still evolve subject to its before-update value. This time scale consideration brings an interesting feature when modeling biological networks which gathers processes of different nature and velocity. Our encoding allows the application of the interval semantics only to a subset of nodes, offering a flexible modelling approach.

Future work consider determining semantics of Boolean networks which guarantee the formal simulation of hybrid and continuous network dynamics.

Acknowledgements

The authors acknowledge the support from the French Agence Nationale pour la Recherche (ANR), in the context of the ANR-FNR project "Algo-ReCell" ANR-16-CE12-0034, from the Labex DigiCosme (project ANR-11-LABEX-0045-DIGICOSME) operated by ANR as part of the program "Investissement d'Avenir" Idex Paris-Saclay (ANR-11-IDEX-0003-02), from the INRIA Associated Team *LifeForm*, and from Paris Ile-de-France Region (DIM RFSI/FormaReBio).

References

1. J. Aracena. Maximum number of fixed points in regulatory boolean networks. *Bulletin of Mathematical Biology*, 70(5):1398–1409, 2008.
2. J. Aracena, J. Demongeot, and E. Goles. Positive and negative circuits in discrete neural networks. *IEEE Transactions of Neural Networks*, 15:77–83, 2004.
3. J. Aracena, E. Goles, A. Moreira, and L. Salinas. On the robustness of update schedules in Boolean networks. *Biosystems*, 97(1):1 – 8, 2009.
4. J. Aracena, A. Richard, and L. Salinas. Number of fixed points and disjoint cycles in monotone boolean networks. *SIAM Journal on Discrete Mathematics*, 31(3):1702–1725, 2017.
5. J. Baetens, P. V. der Weeën, and B. D. Baets. Effect of asynchronous updating on the stability of cellular automata. *Chaos, Solitons & Fractals*, 45(4):383 – 394, 2012.
6. P. Baldan, A. Corradini, and U. Montanari. Contextual Petri nets, asymmetric event structures, and processes. *Information and Computation*, 171(1):1–49, 2001.
7. G. Bernot, F. Cassez, J.-P. Comet, F. Delaplace, C. Müller, and O. Roux. Semantics of biological regulatory networks. *Electronic Notes in Theoretical Computer Science*, 180(3):3 – 14, 2007.
8. N. Busi and G. M. Pinna. Non sequential semantics for contextual P/T nets. In *Application and Theory of Petri Nets*, volume 1091 of *Lecture Notes in Computer Science*, pages 113–132. Springer, 1996.
9. C. Chaouiya, A. Naldi, E. Remy, and D. Thieffry. Petri net representation of multi-valued logical regulatory graphs. *Natural Computing*, 10(2):727–750, 2011.
10. T. Chatain, S. Haar, L. Jezequel, L. Paulevé, and S. Schwoon. Characterization of reachable attractors using Petri net unfoldings. In *Computational Methods in Systems Biology*, volume 8859 of *Lecture Notes in Computer Science*, pages 129–142. Springer, 2014.
11. T. Chatain, S. Haar, M. Koutny, and S. Schwoon. Non-atomic transition firing in contextual nets. In *Applications and Theory of Petri Nets*, volume 9115 of *Lecture Notes in Computer Science*, pages 117–136. Springer, 2015.
12. Th. Chatain, S. Haar, and L. Paulevé. Beyond Generalized Asynchronicity. In J. Baetens and M. Kutrib, editors, *Proceedings of the 24th Annual International Workshop on Cellular Automata and Discrete Complex Systems (AUTOMATA'18)*, Lecture Notes in Computer Science, Ghent, Belgium, June 2018. Springer. To appear.
13. A. Garg, A. Di Cara, I. Xenarios, L. Mendoza, and G. De Micheli. Synchronous versus asynchronous modeling of gene regulatory networks. *Bioinformatics*, 24(17):1917–1925, 2008.
14. R. Janicki and M. Koutny. Semantics of inhibitor nets. *Information and Computation*, 123(1):1–16, 1995.
15. R. Janicki and M. Koutny. Fundamentals of modelling concurrency using discrete relational structures. *Acta Inf.*, 34:367–388, 1997.
16. S. A. Kauffman. Metabolic stability and epigenesis in randomly connected nets. *Journal of Theoretical Biology*, 22:437–467, 1969.
17. Z. Mai and H. Liu. Boolean network-based analysis of the apoptosis network: Irreversible apoptosis and stable surviving. *Journal of Theoretical Biology*, 259(4):760 – 769, 2009.
18. P. Martínez-Sosa and L. Mendoza. The regulatory network that controls the differentiation of t lymphocytes. *Biosystems*, 113(2):96 – 103, 2013.

19. M. Noual and S. Sené. Synchronism versus asynchronism in monotonic boolean automata networks. *Natural Computing*, 2017.
20. E. Palma, L. Salinas, and J. Aracena. Enumeration and extension of non-equivalent deterministic update schedules in boolean networks. *Bioinformatics*, 32(5):722–729, 2016.
21. E. Remy, P. Ruet, and D. Thieffry. Graphic requirements for multistability and attractive cycles in a Boolean dynamical framework. *Advances in Applied Mathematics*, 41(3):335 – 350, 2008.
22. A. Richard. Negative circuits and sustained oscillations in asynchronous automata networks. *Advances in Applied Mathematics*, 44(4):378 – 392, 2010.
23. B. Schönfisch and A. de Roos. Synchronous and asynchronous updating in cellular automata. *Biosystems*, 51(3):123 – 143, 1999.
24. L. J. Steggles, R. Banks, O. Shaw, and A. Wipat. Qualitatively modelling and analysing genetic regulatory networks: a petri net approach. *Bioinformatics*, 23(3):336–343, 2007.
25. D. Thieffry and R. Thomas. Dynamical behaviour of biological regulatory networks – II. Immunity control in bacteriophage lambda. *Bulletin of Mathematical Biology*, 57:277–297, 1995.
26. R. Thomas. Boolean formalization of genetic control circuits. *Journal of Theoretical Biology*, 42(3):563 – 585, 1973.
27. P. Traynard, A. Fauré, F. Fages, and D. Thieffry. Logical model specification aided by model-checking techniques: application to the mammalian cell cycle regulation. *Bioinformatics*, 32(17):i772–i780, 2016.
28. W. Vogler. Partial order semantics and read arcs. *Theoretical Computer Science*, 286(1):33–63, 2002.
29. J. Winkowski. Processes of contextual nets and their characteristics. *Fundamenta Informaticae*, 36(1), 1998.

Developing and Applying a Rewriting Framework for Timed Mobility

Gabriel Ciobanu[1] and Jason Steggles[2]

[1] Romanian Academy, Institute of Computer Science, Iaşi, Romania,
gabriel@info.uaic.ro
[2] School of Computing Science, University of Newcastle, UK,
jason.steggles@ncl.ac.uk

Abstract. TiMo (Timed Mobility) is a process algebra developed for modelling mobile based systems with timing constraints. We present a new semantic model for Timed Mobility (TiMo) by mapping its operational semantics rules to corresponding rewrite rules in Maude. We use the meta-programming capabilities of Maude to formulate a rewriting strategy that captures the maximal parallel computational steps of TiMo . We formally prove the correctness of the translation of TiMo into Maude by showing the soundness and completeness with respect to the operational semantics of TiMo . To illustrate the new rewriting framework, we formulate a TiMo specification of a simple robot swarm example, and use Maude to simulate and analyse it.

Keywords: Timed Mobility, Maximal Concurrency, Rewriting Logic.

1 Introduction

In this paper we highlight one of the many contribution made by Maciej Koutny to the field of modelling and reasoning about concurrent systems, namely the development of TiMo (Timed Mobility). TiMo was introduced in [9] as a process calculus able to describe migrating agents using specific features such as explicit locations, timeouts, and timed migration and communication in distributed systems. After this first step, the same authors presented a structural translation of TiMo into behaviourally equivalent high level timed Petri nets in [10]. As a result, it is obtained a formal net semantics for timed interaction and migration which is both structural and allows to deal directly with concurrency and causality. In [11], TiMo is extended with (dynamic) access permissions; a more detailed version is presented in [18]. Overall, the approach is motivated by the 'globally asynchronous/locally synchronous' execution strategy (as found in GALS approach [22]) in a semantic framework based on local maximal concurrency. Processes are viewed as residing within distinct locations, where each location has its own local clock. Processes are allowed to migrate between locations, and this is controlled by timers linked to the local clock of the location containing the process. Timers are also used to control communication between co-located processes.

The operational semantics of TiMo is provided by a transition system labelled with multisets of actions executed in parallel in one step. A complete computational step is captured by a parallel execution of actions followed by a time step. Based on such an operational semantics, it is proved that the passage of time does not introduce any nondeterminism into the execution of a process, and the processes evolve properly in time. The standard notion of bisimilarity is extended to TiMo in [4] to deal with timed transitions and multisets of actions.

In this paper we use *Rewriting Logic* (RL), an algebraic formal modelling approach based on using equations to define static states and rewrite rules to define dynamic state transitions [30]. In previous work [13], a new semantic model for TiMo was developed by using RL and strategies with the aim of providing a foundation for tool support. In particular, rewriting strategies are used to capture the locally maximal concurrent step of a TiMo specification and the RL approach was realised using the support tool Elan [7]. This approach was then extended in [16] with access permissions in order to develop a new semantic model for PerTiMo [18]. These semantical models are formally proved to be sound and complete with respect to the original operational semantics on which they were based. We build on this work here, and develop an updated RL semantic model for TiMo based on using Maude [20], an adaptable formal tool framework for modelling and analysing RL models. Maude provides a range of interesting analysis tools (such as an LTL model checker [24]) and importantly, it has powerful meta–programming capabilities [19] which allow rewriting strategies [25] to be developed to refine an RL model.

We develop a new semantic model for TiMo by mapping its operational semantics rule set to a corresponding set of rewrite rules in Maude. An interesting aspect of this is how to cope with the maximal concurrency captured by the the time progression rule which is based on using negative premises. We make use of Maude's meta–programming capabilities to formulate a *rewriting strategy* [25] that captures this maximal parallel computational step. We formally discuss the correctness of the resulting Maude specification by proving it is both *sound* and *complete* with respect to the original operational semantics of TiMo.

To illustrate the new Maude framework, we consider a case study based on a *Complex Adaptive System (CAS)* [26, 5]. We formulate a new TiMo model of a simple robot swarm example based on robots collaborating to pull up sticks [27, 28], and then use Maude to simulate and analyse this model. This simple example gives useful insight into the flexibility of the proposed RL modelling approach and illustrates the type of interesting analysis that can be done.

The paper is structured as follows. Section 2 describes the syntax and semantics of TiMo. Section 3 briefly introduces RL and the Maude support tool. In Section 4, we develop an RL model of TiMo using Maude and its metaprogramming capabilities. In Section 5 we illustrate the framework we have developed using a robot swarm example based on collaborative stick pulling. Finally, in Section 6 we make some concluding remarks.

2 The Development of TiMo

TiMo (Timed Mobility) [9–11] is a process algebra for mobile based systems where it is possible to add timers to communication and mobility actions. Processes reside within distinct locations and each location runs according to its own local clock. Processes are allowed to migrate between locations and communication can occur between co–located processes; these actions are controlled by timers linked to the local clock of the location the process resides in.

To formalise TiMo we begin by defining a set *Loc* of *locations*, a set *Chan* of *communication channels*, and a set *Id* of process identifiers, where each $id \in Id$ has arity m_{id}. We use $\boldsymbol{x}$ to denote a finite tuple of elements $(x_1, \ldots, x_k)$ whenever it does not lead to a confusion.

The syntax of TiMo is given in Table 1, where P represents *processes* and N represents networks. Moreover, for each $id \in Id$, there is a unique process definition (Def), where P_{id} is a process expression, the u_i's are distinct variables playing the role of parameters, and the X_i^{id}'s are data types. In Table 1, it is assumed that: (i) $a \in Chan$ is a channel, and $t \in \mathbb{N} \cup \{\infty\}$ represents a timeout; (ii) each v_i is an expression built from data values and variables; (iii) each u_i is a variable, and each X_i is a data type; (iv) l is a location or a location variable; and (v) Ⓢ is a special symbol used to state that a process is temporarily 'stalled'.

Processes	$P ::= a^{\Delta t}\,\mathbf{!}\,\langle \boldsymbol{v} \rangle$ `then` P `else` P' ∣	(*output*)
	$a^{\Delta t}\,\mathbf{?}\,(\boldsymbol{u}{:}\boldsymbol{X})$ `then` P `else` P' ∣	(*input*)
	$\mathtt{go}^{\Delta t}\ l$ `then` P ∣	(*move*)
	$P \mid P'$ ∣	(*parallel*)
	$id(\boldsymbol{v})$ ∣	(*recursion*)
	stop ∣	(*termination*)
	ⓈP	(*stalling*)
Networks	$N ::= l[\![\, P \,]\!] \mid N \mid N'$	
Definition	$id(u_1, \ldots, u_{m_{id}} : X_1^{id}, \ldots, X_{m_{id}}^{id}) \stackrel{\text{df}}{=} P_{id}$	(Def)

Table 1. TiMo Syntax. Length of $\boldsymbol{u}$ is the same as $\boldsymbol{X}$, and length of $\boldsymbol{v}$ in $id(\boldsymbol{v})$ is m_{id}.

The only variable binding construct is $a^{\Delta t}\,\mathbf{?}\,(\boldsymbol{u}{:}\boldsymbol{X})$ `then` P `else` P' which binds the variables $\boldsymbol{u}$ within P (but *not* within P'). We use $fv(P)$ to denote the free variables of a process P (and similarly for networks). For a process definition as in (Def), we assume that $fv(P_{id}) \subseteq \{u_1, \ldots, u_{m_{id}}\}$, and so the free variables of P_{id} are parameter bound. Processes are defined up to the alpha-conversion, and $\{v/u, \ldots\}P$ is obtained from P by replacing all free occurrences of a variable u by v, etc, possibly after alpha-converting P in order to avoid clashes. Moreover, if $\boldsymbol{v}$ and $\boldsymbol{u}$ are tuples of the same length then $\{\boldsymbol{v}/\boldsymbol{u}\}P$ denotes $\{v_1/u_1, v_2/u_2, \ldots, v_k/u_k\}P$.

A process $a^{\Delta t}\,\mathbf{!}\,\langle \boldsymbol{v} \rangle$ `then` P `else` P' attempts to send a tuple of values $\boldsymbol{v}$ over the channel a for t time units. If successful, it continues as process P; otherwise

(Eq1-3) $N \mid N' \equiv N' \mid N \quad (N \mid N') \mid N'' \equiv N \mid (N' \mid N'') \quad l[\![P \mid P']\!] \equiv l[\![P]\!] \mid l[\![P']\!]$

(Call) $l[\![id(\boldsymbol{v})]\!] \xrightarrow{id@l} l[\![Ⓢ\{\boldsymbol{v}/\boldsymbol{u}\}P_{id}]\!]$ (Move) $l[\![\mathtt{go}^{\Delta t}\ l'\ \mathtt{then}\ P]\!] \xrightarrow{l'@l} l'[\![Ⓢ P]\!]$

(Com) $$\frac{v_1 \in X_1 \ \ldots \ v_k \in X_k}{l[\![a^{\Delta t}\,!\,\langle \boldsymbol{v} \rangle\ \mathtt{then}\ P\ \mathtt{else}\ Q \mid a^{\Delta t'}\,?\,(\boldsymbol{u}{:}\boldsymbol{X})\ \mathtt{then}\ P'\ \mathtt{else}\ Q']\!] \xrightarrow{a\langle \boldsymbol{v} \rangle @l} l[\![Ⓢ P \mid Ⓢ\{\boldsymbol{v}/\boldsymbol{u}\}P']\!]}$$

(Par) $$\frac{N \xrightarrow{\psi} N'}{N \mid N'' \xrightarrow{\psi} N' \mid N''}$$ (Time) $$\frac{N \not\xrightarrow{}_l}{N \xrightarrow{\surd_l} \phi_l(N)}$$

(Equiv) $$\frac{N \equiv N' \quad N' \xrightarrow{\psi} N'' \quad N'' \equiv N'''}{N \xrightarrow{\psi} N'''}$$

Table 2. Three rules of the structural equivalence (Eq1-Eq3), and six action rules (Call), (Move), (Com), (Par), (Equiv), (Time) of the operational semantics. In (Par) and (Equiv) ψ is an action, and in (Time) l is a location.

it continues as the alternative process P'. A process $a^{\Delta t}\,?\,(\boldsymbol{u}{:}\boldsymbol{X})\ \mathtt{then}\ P\ \mathtt{else}\ P'$ attempts for t time units to input a tuple of values of type $\boldsymbol{X}$ and substitute them for the variables $\boldsymbol{u}$. Mobility is implemented by a process $\mathtt{go}^{\Delta t}\ l\ \mathtt{then}\ P$ which moves from the current location to the location l within t time units. Note that since l can be a variable, and so its value is assigned dynamically through communication with other processes, migration actions support a flexible scheme for moving processes around a network. Processes are further constructed from the (terminated) process stop and parallel composition $P|P'$. Finally, process expressions of the form Ⓢ P are a purely technical device which is used in the subsequent formalisation of structural operational semantics of TiMo; intuitively, Ⓢ specifies that a process P is temporarily (i.e., until a clock tick) *stalled* and so cannot execute any action. A located process $l[\![P]\!]$ is a process running at location l, and a network is composed out of its components $N \mid N'$.

A network N is *well-formed* if: (i) there are no free variables in N; (ii) there are no occurrences of the special symbol Ⓢ in N; (iii) assuming that id is as in the recursive equation (Def), for every $id(\boldsymbol{v})$ occurring in N or on the right hand side of any recursive equation, the expression v_i is of type corresponding to X_i^{id}. We let $Prs(TM)$ and $Net(TM)$ represent the set of well-formed TiMo process and network terms respectively. The first component of the operational semantics of TiMo is the structural equivalence $\equiv$ on networks. It is the smallest congruence such that the equalities (Eq1–Eq3) in Table 2 hold. Using (Eq1–Eq3) one can always transform a given network N into a finite parallel composition of networks of the form $l_1[\![P_1]\!] \mid \ldots \mid l_n[\![P_n]\!]$ such that no process P_i has the parallel composition operator at its topmost level. Each subnetwork $l_i[\![P_i]\!]$ is called a *component* of N, the set of all components is denoted by $comp(N)$, and the parallel composition is called a *component decomposition* of the network N. Note that these notions are well defined since component decomposition is

unique up to the permutation of the components. This follows from the rule (CALL) which treats recursive definitions as function calls which take a unit of time. Another consequence of such a treatment is that it is impossible to execute an infinite sequence of action steps without executing any local clock ticks.

Table 2 introduces two kinds of operational semantics rules: $N \xrightarrow{\psi} N'$ and $N \xrightarrow{\surd_l} N'$. The former is an execution of an action ψ by some process, and the latter a unit time progression at location l. In the rule (TIME), $N \not\rightarrow_l$ means that the rules (CALL) and (COM) as well as (MOVE) with $\Delta t = \Delta 0$ cannot be applied to N for this particular location l. Moreover, $\phi_l(N)$ is obtained by taking the component decomposition of N and simultaneously replacing all the components of the form $l[\![\, \mathtt{go}^{\Delta t}\ l'\ \mathtt{then}\ P \,]\!]$ by $l[\![\, \mathtt{go}^{\Delta t-1}\ l'\ \mathtt{then}\ P \,]\!]$, and all components of the form $l[\![\, a^{\Delta t}\omega\ \mathtt{then}\ P\ \mathtt{else}\ Q \,]\!]$ (where ω stands for $\mathbf{!}\langle \boldsymbol{v} \rangle$ or $\mathbf{?}(\boldsymbol{u}{:}\boldsymbol{X})$) by $l[\![\, Q \,]\!]$ if $t = 0$, and $l[\![\, a^{\Delta t-1}\omega\ \mathtt{then}\ P\ \mathtt{else}\ Q \,]\!]$ otherwise. After that, all the occurrences of the symbol Ⓢ in N are erased.

The above defines executions of individual actions. A complete computational step is captured by a *derivation* of the form $N \stackrel{\Psi}{\Longrightarrow} N'$, where $\Psi = \{\psi_1, \ldots, \psi_m\}$ $(m \geq 0)$ is a finite multiset of l-actions for some location l (i.e., actions of the form $id@l$ or $l'@l$ or $a\langle \boldsymbol{v} \rangle @l$) such that $N \xrightarrow{\psi_1} N_1 \cdots N_{m-1} \xrightarrow{\psi_m} N_m \xrightarrow{\surd_l} N'$. That is, a derivation is a condensed representation of a sequence of individual actions followed by a clock tick, all happening at the same location. Intuitively, we capture the cumulative effect of the concurrent execution of the multiset of actions Ψ at location l. We say that N' is *directly reachable* from N. Note that whenever there is only a time progression at a location, we have $N \stackrel{\varnothing}{\Longrightarrow} N'$.

One can show that derivations are well defined as one cannot execute an unbounded sequence of action moves without time progress, and the execution Ψ is made up of independent (or concurrent) individual executions. Moreover, derivations preserve well-formedness of networks (see [9]).

3 Rewriting Logic and MAUDE

Rewriting logic (RL) [30] is a formal modelling and analysis framework based on an algebraic specification approach. In order to model a dynamic system in RL there are two stages. First the static states of the system are specified using standard equational specification techniques. Secondly, rewrite rules are used to specify the non–deterministic state transitions that represent the dynamic behaviour of the system. The application of rewrite rules can be controlled using *rewriting strategies* [25] and the result is an expressive and versatile formal framework. RL has been applied to model a wide range of different formalisms and systems, including: biological systems [23, 31], Petri nets [34, 32], and process algebras [29, 16].

A range of different tools have been developed to support RL (see [20, 7, 6]). In this paper, we use MAUDE [20], an advanced support tool for RL that provides a range of interesting analysis tools (such as an LTL model checker

[24]) and meta–programming capabilities. MAUDE has been widely used to create executable environments for different languages and models of computation [21].

In order to illustrate using MAUDE consider the following simple RL specification written in MAUDE:

```
mod EXABCD is
  sorts Entity State .
  subsort Entity < State .

  ops A B C D : -> Entity .
  op __ : State State -> State [assoc comm].

  rl [rule1] : A A => B .
  rl [rule2] : A B => C .
  rl [rule3] : C => A A A .
  rl [rule4] : C B => D A .
endm
```

The system contains four entities `A`, `B`, `C`, and `D` which are declared as constants of sort `Entity`. The states of the system are represented as multi–sets of entities and these are modelled by introducing the sort `State`. The sort `Entity` is declared as a subsort of `State` which means that an entity can be viewed as a singleton state. We use an implicit multi-set union operator `__ : State State -> State` (where `_` is denotes the location of an infix argument) and define this to be associative and commutative by adding the flags `[assoc comm]` (this corresponds to adding the appropriate equations for these properties). The dynamic transitions allowed in the system are then defined using the four rewrite rules given in the specification. As an example, consider the following rewrite trace derived from the initial state `C C`:

```
C C => A A A C => B A C => D A A => D B
```

A range of analysis tools are provided by MAUDE, such as the built–in model checking command `search S =>+ P`, which allows us to check if a pattern term `P` can be reached by rewriting an initial ground term `S`. For example, we can use the `search` command to check if we can derive a state containing `D D` from an initial state `C C C`:

```
search C C C =>+ D D s:State .
```

This search returns true (with `s` instantiated with `A`) and we can view a corresponding witness rewrite trace.

One key motivation for using MAUDE is the meta–programming capabilities it offers. This meta–programming allows the definition of rewriting strategies which can be used to control the way in which the rewrite rules are applied. To illustrate this, consider the following meta–programming example which defines a rewrite strategy that prioritises `rule3` over the other rules.

```
ceq rwStrat(T) = if Step? :: Result4Tuple then getTerm(Step?)
    else (if Step2? :: ResultPair then getTerm(Step2?) else T  fi)
    fi
    if Step? := metaXapply(upModule('EXABCD,false),T,'rule3,
                none,0,unbounded,0)
       /\  Step2? := metaRewrite(upModule('EXABCD,false),T,1) .
```

The rewriting strategy `rwStrat` is defined using a conditional equation which allows local definitions (based on `:=`) to be used to combine the various parts of the strategy. It makes use of two built–in meta–level functions: `metaXapply` which allows a given rule to be applied (in this case `rule1`) to a term `T` ; and `metaRewrite` which allows a term `T` to be rewritten. Type checking is used to ensure that the meta–level functions have been successfully applied, e.g. `Step? :: Result4Tuple` is used to check if `rule1` has been successfully applied. For full details of the notation used here and further rewriting strategy examples see the Maude manual [21].

4 Translating TiMo into Maude

In this section we consider how to translate a TiMo specification TM into a semantically equivalent Maude model $Md(TM)$. We begin by defining the sorts required in $Md(TM)$ to model the key TiMo concepts of channels, locations, processes and networks as follows:

```
sorts Chan VLoc ALoc Loc Prs Nets .
subsorts VLoc ALoc < Loc .
```

Note that in order to model the location parameter passing that can occur in communication we defined the sort `Loc` for locations to consist of two subsorts: `VLoc` which represents locations variables; and `ALoc` representing actual location names.

Next we define the function symbols needed in $Md(TM)$ to represent processes and networks.

```
op stop : -> Prs .
op S : Prs -> Prs .
op _|_ : Prs Prs -> Prs [assoc comm] .
op go : Time Loc Prs -> Prs .
op in : Chan Time VLoc Prs Prs -> Prs .
op out : Chan Time Loc Prs Prs -> Prs .
op _[_] : ALoc Prs -> Nets .
op _|_ : Nets Nets -> Nets [assoc comm] .
```

Note that the function symbol `_|_` is overloaded and is used to represent parallel composition of both processes and networks in TiMo. Both instances are defined (equationally) to be associative and commutative using the operator

flags `[assoc comm]`. The function symbol `S` is used to represent the special stall symbol Ⓢ used in TiMo to control time progression.

To model process definitions we need to add a function symbol

```
op id : s1 ... sn -> Prs
```

for each process identifier $id(u_1, \ldots, u_n : s_1, \ldots, s_n)$ in our TiMo specification TM, where `si` is assumed to be a well–defined algebraic data type in $Md(TM)$ representing s_i.

To capture the dynamic semantics of TiMo processes and networks, as defined by the action rules in Table 2, we need to formulate appropriate rewrite rules to define the behaviour of the Maude model. In order to simplify working with network terms we make the simplifying assumption that network components with the same location are always merged into a single network structure (this assumption is clearly valid given Eq 3 from Table 2). We enforce this by adding the following equation to $Md(TM)$:

```
eq (AL[P1]) | (AL[P2]) = AL[P1 | P2] .
```

Each individual network location term `l[P1 | ... | Pn]` will therefore consist of a number of atomic processes `Pi` (where a process term is referred to as *atomic* if it does not have the parallel operator at its topmost level).

When considering mobility we have a non–deterministic choice between executing a `go` command or allowing time to pass. We incorporate this behaviour into $Md(TM)$ by adding the following pair of rules:

```
rl [move] : (AL[go(T,AL1,P1) | P2] | N) =>
              (AL[P2] | AL1[S(P1)] | N) .
rl [move] : (AL[go(T,AL1,P1) | P2] | N) =>
              (AL[S(go(T-1),AL1,P1)) | P2] | N)  if notExp(T) .
```

The idea is that while $\mathtt{T} > 0$ (i.e. `notExp(T)` equates to true) either rule can be applied allowing the process to either wait or to execute the move. However, as soon as the timer `T` has expired (i.e. $\mathtt{T} = 0$ and so `notExp(T)` equates to false) then only the first rule that moves `P1` to the new location `AL1` can be used.

Communication in TiMo is based on output and input processes synchronising on a common channel within a location. We model this synchronisation by using the following rule:

```
rl [com] : (AL[out(C,T1,AL1,P1,P2) | in(C,T2,VL,P3,P4)]) | N) =>
                (AL[S(P1) | S(sub(P3,VL,AL1))] | N) .
```

This rule makes use of a substitution function `sub : Prs VLoc ALoc -> Prs`, where the term `sub(P,VL,AL)` represents the process term resulting from substituting all free occurrences (not bound by an input action symbol) of the location variable `VL` in the process term `P` by the actual location term `AL`. It is straightforward to define `sub` equationally using recursion over process terms.

For each process definition

$$id(u_1, \ldots, u_n : s_1, \ldots, s_n) \stackrel{\text{df}}{=} P_{id}$$

we add the following corresponding rule to $Md(TM)$:

```
rl [call] : (AL[id(u1,...,un) | P] | N) => (AL[S(Pid) | P] | N) .
```

where `Pid` is the process term that results from translating P_{id} into $Md(TM)$ and each `ui` is a variable of sort `si` in $Md(TM)$.

In the sequel we refer to rules `move`, `com` and `call` as *process transition rules*.

Every derivation step in TiMo finishes with the application of the (Time) action rule (see Table 2). This allows time to progress by updating timers and removing all instances of the stall symbol. To model this in $Md(TM)$ we introduce a function `tick : Prs -> Prs` which we define equational using recursion over process terms. The following equations illustrate the approach taken:

```
eq tick(S(P)) = P .
eq tick(in(C,0,VL,P1,P2)) = P2 .
ceq tick(in(C,T,VL,P1,P2)) = in(C,T-1,VL,P1,P2) if notExp(T) .
eq tick(P1 | P2) = tick(P1) | tick(P2) .
```

Note that the `go` command does not feature explicitly in these equations since its timer is handled within the `move` process rules above. We then overload `tick` so that it can be applied to network terms in the obvious way.

We now have to compose the above components to correctly model within $Md(TM)$ a derivation step. This is done by defining a *rewriting strategy* using Maude's metalevel programming capabilities [20, 25]. First, we define a metalevel operation `update : Term -> Term` which allows processes in the chosen location to perform an action if allowable.

```
ceq update(T) =
        if Step? :: ResultPair then getTerm(Step?) else T  fi
    if Step? := metaRewrite(upModule('Md(TM), false), T, unbounded) .
```

We then build on this by defining a metalevel operation `next : Term -> Term` which applies `update` to a term and then allows time to progress by applying the `tick` function.

```
ceq next(T) =
        if Step? :: ResultPair then getTerm(Step?) else T1 fi
    if T1 := update(T) /\
       Step? := metaReduce(upModule('Md(TM), false), 'tick[T1]) .
```

Note that in the above we use the metalevel representation of `tick` as indicated by the backquote and that `metaReduce` is used to apply the defining equations in $Md(TM)$.

Finally, we introduce a conditional rule `step` which allows the rewriting strategy `next` to be automatically applied within Maude.

```
crl [step] : AL[P] => downTerm(T1,NTerm)
               if T1 := nextState(upTerm(AL[P])) .
```

This conditional rule avoids the need to directly use MAUDE's metalevel notation by applying the built–in functions `upterm` and `downTerm` for moving between the module and metalevel representation of terms. Note the second parameter to `downTerm` is a term `NTerm` which sets the expected kind to be returned from the operation (see [21]).

We conclude this section by considering the important question of the correctness of the above semantic translation from TIMO to MAUDE. In order to formally show that the translation is correct we need to show: *soundness* – each step in our MAUDE model represents a derivation step in TIMO; and *completeness* – every derivation step possible in TIMO is represented in our MAUDE model. To formalise the above correctness properties we begin by defining a bijective mapping $\sigma : Net(TM) \rightarrow valTerm(Md(TM))$ from the set $Net(TM)$ of TIMO network terms in TM to their corresponding terms in $Md(TM)$ (this is straightforward to do using the natural translation given by the syntax). Note that not all terms in $Md(TM)$ are well–defined since they may incorrectly contain the stall symbol or may contain unbounded location variables. We therefore let $valTerm(Md(TM))$ denote the set of all well–defined network terms in $Md(TM)$.

The following result shows the `step` rule preserves $valTerm(Md(TM))$ terms.

Theorem 1 For any network term `net1` $\in valTerm(Md(TM))$, if `net1 => net2` by an application of the rule `step` then `net2` $\in valTerm(Md(TM))$.

Proof. Consider a network term `Loc[p1 | ... | pn]` $\in valTerm(Md(TM))$, where `n` > 0 and each process term `pi` is atomic. It can be shown that each process term `pi` can be updated by at most one of the process transition rules and this gives four cases to consider: 1) rule `move` was applied; 2) rule `com` was applied; 3) rule `call` was applied; or 4) no process transition rule was applied and time was simply allowed to progress. It can be shown that each of these cases results in a well-defined term that represents a corresponding TIMO process. (See [13] for an example of this type of proof.) □

We can now show that $Md(TM)$ is a sound and complete model of TIMO specification TM (see Figure 1).

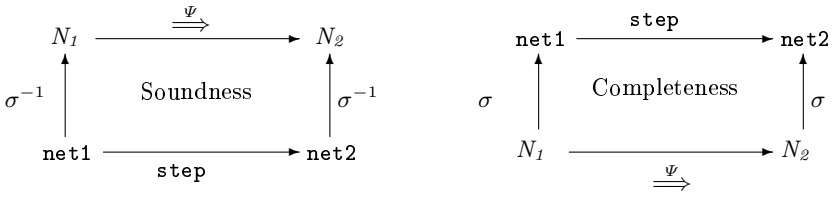


Fig. 1. The properties of soundness and completeness required for $Md(TM)$ to be a correct model of TM.

Theorem 2 (Soundness) Let $\texttt{net1}, \texttt{net2} \in \mathit{valTerm}(\mathit{Md}(\mathit{TM}))$ be valid network terms. Then if `net1 => net2` by an application of the rule `step` then $\sigma^{-1}(\texttt{net1}) \stackrel{\Psi}{\Longrightarrow} \sigma^{-1}(\texttt{net2})$ for some finite multiset $\Psi = \{\psi_1, \ldots, \psi_m\}$ of l-actions and some location l (i. e. the diagram for soundness in Figure 1 commutes).

Proof. By the definition of rule `step` and the notion of a derivation in TIMO it suffices to consider a valid network location term of the form

$$\texttt{l[p1 | ... | pn]} \in \mathit{valTerm}(\mathit{Md}(\mathit{TM})),$$

where $\texttt{n} > 0$ and each process term `pi` is atomic. It can be seen that each process term `pi` is involved in at most one process transition rule application when rule `step` is applied. This gives use four possible cases to consider: 1) rule `move` was applied; 2) rule `call` was applied; 3) rule `com` was applied; or 4) no process transition rule was applied and time was simply allowed to progress. For brevity, we consider only Case 1) in detail here (for a complete example of this type of proof see [13]).

Case 1) Suppose `pi` has the form `go(t,l2,p)` and that a `move` rule is applied. Then there are two possible cases to consider:
i) *Timer is reduced*: Suppose $\texttt{t} > 0$ and the `move` rule applied simply allowed time to progress

$$\texttt{go(t,l2,p) => S(go(t-1,l2,p))}$$

By the definition of strategy `next` and `tick` we know the stall symbol S will be removed resulting in the process term `go(t-1,l2,p)`. By the definition of time progression in TIMO and the assumption $\texttt{t} > 0$ we have

$$l\,[\![\,\texttt{go}^{\Delta t}\ l2\ \texttt{then}\ \sigma^{-1}(\texttt{p})\,]\!] \xrightarrow{\surd_l} at\,[\![\,\texttt{go}^{\Delta t-1}\ l2\ \texttt{then}\ \sigma^{-1}(\texttt{p})\,]\!]$$

as required.
ii) *Process moves*: Suppose applying the `move` rule resulted in the process moving to location `l2` producing the network term `l2[S(p)]`. By the definition of strategy `next` and `tick` we know the stall symbol S will be removed resulting in the network term `l2[p]`. By the action rule (MOVE) (Table 2) we have

$$l\,[\![\,\texttt{go}^{\Delta t}\ l2\ \texttt{then}\ \sigma^{-1}(\texttt{p})\,]\!] \xrightarrow{l2@l} l2\,[\![\,\textcircled{S}\sigma^{-1}(\texttt{p})\,]\!]$$

The result follows since the stall symbol Ⓢ will be removed by the time progression step in TIMO. □

Theorem 3 (Completeness) Let $N_1, N_2 \in \mathit{Net}(\mathit{TM})$ be any well–formed network terms in TM. Then, if $N_1 \stackrel{\Psi}{\Longrightarrow} N_2$, for some location l and some multi-set $\Psi = \{\psi_1, \ldots, \psi_m\}$ of l-actions, then $\sigma(N_1)$ `=>` $\sigma(N_2)$ by applying the `step` rule. In other words, the diagram for completeness in Figure 1 commutes.

Proof. By the definition of a derivation in TiMo and the `step` rule it suffices to consider a well–formed network of the form

$$l[\![P_1 \mid \ldots \mid P_n]\!] \equiv l[\![P_1]\!] \mid \ldots \mid l[\![P_n]\!],$$

where $n > 0$ and each P_i is an atomic process. Suppose

$$l[\![P_1 \mid \ldots \mid P_n]\!] \stackrel{\Psi}{\Longrightarrow} N',$$

for some finite set of l–actions $\Psi = \{\psi_1, \ldots, \psi_m\}$, $m \geq 0$. Then it can be seen that each atomic process P_i is involved in at most one l–action ψ_i. We show that the derivation applied to each process P_i is correctly captured by the `step` rule in $Md(TM)$. We have four possible cases to consider: 1) rule `move` was applied; 2) rule `call` was applied; 3) rule `com` was applied; or 4) no process transition rule was applied and time was simply allowed to progress. For brevity, we consider only Case 3) in detail here (for a complete example of this type of proof see [13]).

Case 3) Suppose the action rule (Com) has been applied to two processes P_i and P_j, for $i \neq j$, i.e.

$$l[\![c^{\Delta t_1}\,!\,\langle l_2\rangle \texttt{ then } P_i^1 \texttt{ else } P_i^2 \mid c^{\Delta t_2}\,?\,(vl : Loc) \texttt{ then } P_j^1 \texttt{ else } P_j^2]\!]$$

$$\xrightarrow{c<l_2>@l} l[\![Ⓢ P_i^1 \mid Ⓢ\{l_2/vl\}P_j^1]\!]$$

where the stall symbols Ⓢ will be removed by the final time step. Then we have

$$\begin{array}{l}\sigma(c^{\Delta t_1}\,!\,\langle l_2\rangle \texttt{ then } P_i^1 \texttt{ else } P_i^2 \mid c^{\Delta t_2}\,?\,(vl : Loc) \texttt{ then } P_j^1 \texttt{ else } P_j^2)) \\ = \texttt{out(c,t1,l2,}\sigma(P_i^1)\texttt{,}\sigma(P_i^2)\texttt{)) | in(c,t2,vl,}\sigma(P_j^1)\texttt{,}\sigma(P_j^2)\texttt{))}\end{array}$$

By applying the `com` rule we have

$$\begin{array}{l}\texttt{out(c,t1,l2,}\sigma(P_i^1)\texttt{,}\sigma(P_i^2)\texttt{)) | in(c,t2,vl,}\sigma(P_j^1)\texttt{,}\sigma(P_j^2)\texttt{))} \\ = \texttt{S(}\sigma(P_i^1)\texttt{)) | S(sub(}\sigma(P_j^1)\texttt{,vl,l2))}\end{array}$$

where all occurrences of the stall symbol `S` will be removed by the `tick` function. It is then straightforward to see that

$$\sigma(P_i^1 \mid \{l_2/vl\}P_j^1) = \sigma(P_i^1) \mid \texttt{sub}(\sigma(P_j^1), \texttt{vl}, \texttt{l2})$$

by definition of σ. □

5 Case Study: Robot Swarm

In this section we investigate applying the developed Maude framework to analyse a simple *Complex Adaptive Systems (CAS)* [26, 5]. We formulate a new TiMo model of a simple robot swarm example based on robots collaborating to pull up sticks [27, 28], and then use Maude to simulate and analyse this model. This simple example gives useful insight into the flexibility of the proposed Maude modelling approach and illustrates the type of interesting analysis possible.

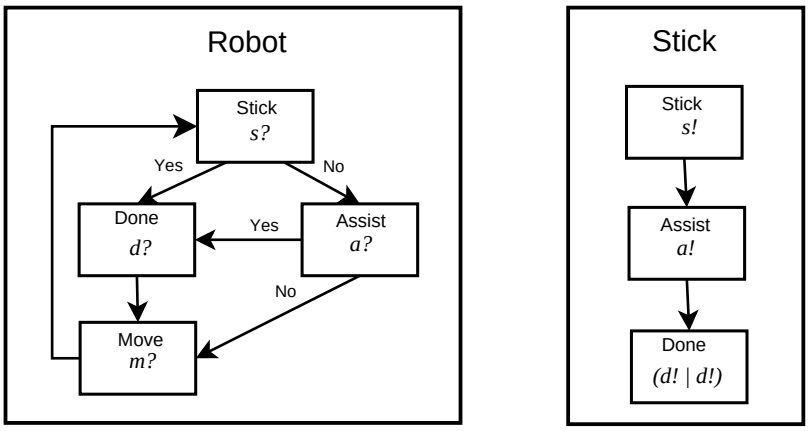


Fig. 2. A graphical representation of the two TiMo processes *stick* and *robot*

5.1 A TiMo Specification of the Stick Pulling Problem

The *stick pulling problem* [27, 28] is a robot swarm example in which a group of robots are given the task of locating and pulling up a set of sticks distributed in a given space. Importantly, no one robot is able to pull up a stick by themselves and so two robots need to collaborate to remove a stick.

To model the problem we define a TiMo specification *SP*. We begin by specifying a simple grid space which is made up of locations labelled $L(i,j)$, where i is the row and j is the column. To indicate the locations reachable from a given grid location we use an output process over a channel m defined as follows:

$$D(l) \stackrel{\text{df}}{=} m^{\Delta 2}\,!\,\langle l\rangle \;\; \texttt{then}\; D(l)\; \texttt{else}\; D(l)$$

As an example of how this can be used, consider defining a 3×3 grid in which robots are able to move vertically and horizontally:

$$\begin{aligned} \mathit{grid} \stackrel{\text{df}}{=}\; & L^{1,1}[\![\, D(L^{1,2}) \,|\, D(L^{2,1}) \,]\!] \;\mid\; L^{1,2}[\![\, D(L^{1,1}) \,|\, D(L^{1,3}) \,|\, D(L^{2,2}) \,]\!] \\ & \mid\; L^{1,3}[\![\, D(L^{1,2}) \,|\, D(L^{2,3}) \,]\!] \;\mid\; L^{2,1}[\![\, D(L^{1,1}) \,|\, D(L^{2,2}) \,|\, D(L^{3,1}) \,]\!] \\ & \mid\; L^{2,2}[\![\, D(L^{2,1}) \,|\, D(L^{1,2}) \,|\, D(L^{2,3}) \,|\, D(L^{3,2}) \,]\!] \\ & \mid\; L^{2,3}[\![\, D(L^{2,2}) \,|\, D(L^{1,3}) \,|\, D(L^{3,3}) \,]\!] \;\mid\; L^{3,1}[\![\, D(L^{2,1}) \,|\, D(L^{3,2}) \,]\!] \\ & \mid\; L^{3,2}[\![\, D(L^{3,1}) \,|\, D(L^{2,2}) \,|\, D(L^{3,3}) \,]\!] \;\mid\; L^{3,3}[\![\, D(L^{3,2}) \,|\, D(L^{2,3}) \,]\!] \end{aligned}$$

We model sticks and robots using the two processes given in Figure 2. The TiMo definitions for these two processes are given below. These definitions make use of three channels to synchronise the stick pulling operation (see Figure 2): channel s is used to check if a free stick needs pulling; channel a is used to check if a robot needs assistance to complete the pulling up of a stick; and channel d is

used to confirm when a stick has been successfully pulled out.

$$\begin{aligned} \mathit{stick} \stackrel{\text{df}}{=} {} & s^{\Delta\infty}\,\texttt{!}\ \texttt{then}\ (a^{\Delta\infty}\,\texttt{!}\ \texttt{then}\ ((d^{\Delta\infty}\,\texttt{!}\ \texttt{then stop else stop}) \\ & \mid (d^{\Delta\infty}\,\texttt{!}\ \texttt{then stop else stop}))\ \texttt{else}\ \mathit{stick})\ \texttt{else}\ \mathit{stick} \end{aligned}$$

$$\begin{aligned} \mathit{robot} \stackrel{\text{df}}{=} {} & s^{\Delta 0}\,\texttt{?}\ \texttt{then}\ (d^{\Delta\infty}\,\texttt{?}\ \texttt{then}\ (m^{\Delta 0}\,\texttt{?}\,(l)\ \texttt{then}\ (\texttt{go}^{\Delta 0}\ l\ \texttt{then}\ \mathit{robot}) \\ & \texttt{else}\ \mathit{robot})\ \texttt{else}\ \mathit{robot})\ \texttt{else}\ (a^{\Delta 0}\,\texttt{?}\ \texttt{then}\ (d^{\Delta\infty}\,\texttt{?}\ \texttt{then} \\ & (m^{\Delta 0}\,\texttt{?}\,(l)\ \texttt{then}\ (\texttt{go}^{\Delta 0}\ l\ \texttt{then}\ \mathit{robot})\ \texttt{else}\ \mathit{robot})\ \texttt{else}\ \mathit{robot})\ \texttt{else} \\ & (m^{\Delta 0}\,\texttt{?}\,(l)\ \texttt{then}\ (\texttt{go}^{\Delta 0}\ l\ \texttt{then}\ \mathit{robot})\ \texttt{else}\ \mathit{robot})) \end{aligned}$$

Given the above TiMo specification *SP* we can define a range of different systems. As an example, consider the system given below which involves three robots and two sticks:

$$\mathit{grid} \mid L^{1,1,}[\![\mathit{robot}]\!] \mid L^{1,3,}[\![\mathit{robot}]\!] \mid L^{3,1,}[\![\mathit{robot}]\!] \mid L^{1,2}[\![\mathit{stick}]\!] \mid L^{2,2}[\![\mathit{stick}]\!]$$

5.2 Applying the Maude Framework

We now apply the techniques developed in Section 4 to translate the TiMo specification *SP* of the stick pulling problem into a Maude model $Md(SP)$. To begin we extend the Maude definitions to incorporate the locations and channels that are used in *SP*. We then need to translate the three process definitions for $D(l)$, *stick*, and *robot* into appropriate Maude rules. As an example, consider the following `call` rule used to model the definition of the *stick* process:

```
rl [call] : (AL[stick | P] | N) => (AL[S(out(s,inf,(out(a,inf,
              ((out(d,inf,stop,stop)) | (out(d,inf,stop,stop))),
              stick)),stick)) | P] | N) .
```

We can now consider analysing $Md(SP)$ using Maude's built–in model checking command `search`. However, as the model stands its use would be very restricted due to the state space explosion problem that arises given the number of locations available to be selected during each derivation step. To address this problem we can go back to the definition of our rewriting strategy captured by the `step` rule and update this strategy so that only locations containing robot processes are considered. This helps to make analysis tractable and illustrates the flexibility of Maude and its metaprogramming capabilities.

Consider the following `search` test that confirms robots are able to collaborate to pull up a stick:

```
search [1] (grid | (L(1,1)[robot]) | (L(2,2)[stick]) |
           (L(3,3)[robot])) =>+ N1 such that not(areSticks(N1)) .
```

The test makes use of a function `areSticks : Nets -> Bool` which captures the property of there being no sticks in a grid. This function is straightforward to define equationally and again this illustrates the flexibility afforded by Maude.

As an example of a further test, consider the more complex test given below which again confirms the robots abilities to collaborate:

```
search [1] (grid | (L(1,1)[robot]) | (L(1,3)[robot]) |
    (L(1,2)[stick]) | (L(2,2)[stick]) | (L(3,1)[robot]))
    =>+ N1 such that not(areSticks(N1)) .
```

A range of interesting analysis can be applied to the model using MAUDE and its various analysis tools, such as the LTL model checker [24]. A detailed analysis of the model (and its potential shortcomings) is beyond the scope of this paper and will be considered in future work.

6 Concluding Remarks

Aiming to bridge the gap between the existing theoretical approach of process calculi and forthcoming realistic programming languages for distributed systems, TIMO was introduced as a rather simple calculus. We can see TIMO as a prototyping language for multi-agent systems, featuring mobility and local interaction. Multi-agent systems typically consist of a large number of agents which exhibit autonomic behaviour depending on their timeouts and actions. The mobility of agents and interaction between the agents through communication may introduce new and sometimes unexpected behaviours. Components can be highly heterogeneous, each operating at different temporal scales and having different objectives. Verifying these systems is becoming increasingly necessary because they are extremely complex and often used in various critical application domains such as e-commerce and distributed collaborative systems. Thus, it is important to have modelling techniques and tools which are able to describe such systems, and to reason about their behaviour in both qualitative and quantitative terms. We see the work on TIMO as an important step towards this goal and this highlights one of the many important contributions to this field made by Maciej Koutny.

After the initial version of TIMO introduced in [9, 10], several variants of TIMO were developed during the last years: a version with access permissions given by a type system [18], a real-time version RTIMO [1], a probabilistic extension PTIMO [14], and a version with costs CTIMO.

Inspired by TIMO, a flexible software platform was presented in [12] to support the specification of agents allowing timed migration in a distributed environment. A verification tool called TIMO@PAT was developed by using an extensible platform for model checkers [15]. A probabilistic temporal logic called PLTM was introduced in [14] to verify properties of PTIMO processes making explicit reference to specific locations, and using temporal constraints over local clocks and multisets of actions. A formal relationship between RTIMO and timed automata allows us to use the model checking capabilities provided by the software tool UPPAAL [2]. TIMO was used to describe a railway control system, and then a new behavioural congruence over real-time systems (named strong open time-bounded bisimulation) was used to check which behaviours are closer to an optimal and safe behaviour [3]. In [17] it is defined a general framework for reasoning about systems specified in TIMO by using the Event-B modelling method and the Rodin platform.

In this paper we have extended existing work [13, 16] to develop a new semantic translation from TiMo to Maude. The aim here was to provide support for analysing TiMo specifications by allowing the range of interesting model checking tools provided by Maude to be applied. We illustrated our approach with a simple CAS robot swarm example based on the stick pulling problem. This case study involved developing a new TiMo specification of the stick pulling problem and illustrated the analysis possible using Maude. In particular, it highlighted the considerable flexibility provided by Maude and its metaprogramming capabilities.

Acknowledgements

We enjoy every moment shared with Maciej (as scientist and as friend), and express our thanks for these moments. *Gabriel Ciobanu*

I would like to sincerely thank Maciej for his support, guidance and friendship over the last 25 years. He is an inspirational researcher and I count myself very luck to have been able to work with him. *Jason Steggles*

References

1. B. Aman, G. Ciobanu. Real-Time Migration Properties of rTiMo Verified in Uppaal. *Lecture Notes in Computer Science* **8137**, 31–45 (2013).
2. B. Aman, G. Ciobanu. Timed Mobility and Timed Communication for Critical Systems. *Lecture Notes in Computer Science* **9128**, 146–161 (2015).
3. B. Aman, G. Ciobanu. Verification of Critical Systems Described in Real-Time TiMo. *Int'l Journal on Software Tools for Technology Transfer*, 1–14 (2016).
4. B. Aman, G. Ciobanu, M. Koutny. Behavioural Equivalences over Migrating Processes with Timers. *Lecture Notes in Computer Science* **7273**, 52–66 (2012).
5. V. Andrikopoulos , A. Bucchiarone, S. Gómez Sáez, D. Karastoyanova, C.A. Mezzina. Towards Modeling and Execution of Collective Adaptive Systems. In: Lomuscio A.R., Nepal S., Patrizi F., Benatallah B., Brandić I. (eds), *Service-Oriented Computing - ICSOC 2013*. LNCS 8377, pages 69–81, Springer, 2014.
6. E.Balland, P.Brauner, R.Kopetz, P.-E.Moreau, and A.Reilles. Tom: Piggybacking rewriting on java. In: RTA'07, LNCS 4533, pages 36–47, Springer Verlag, 2007.
7. P. Borovanský, C. Kirchner, H. Kirchner, P.–E. Moreau and C. Ringeissen: An overview of ELAN. In: C. Kirchner and H. Kirchner (eds), Proc. of *WRLA '98*, *Electronic Notes in Theoretical Computer Science* 15 (1998).
8. G. Ciobanu and C. Prisacariu: Timers for Distributed Systems. *Electronic Notes in Theoretical Computer Science* 164 (2006) 81–99.
9. G. Ciobanu and M. Koutny: Modelling and Verification of Timed Interaction and Migration. Proc. of *FASE'08*, Springer, LNCS 4961 (2008) 215–229.
10. G. Ciobanu and M. Koutny: Timed Mobility in Process Algebra and Petri Nets. *Journal of Algebraic and Logic Programming* 80(7) (2011) 377–391.
11. G. Ciobanu and M. Koutny: Timed Migration and Interaction with Access Permissions. Proc. of *FM'11*, Springer, LNCS 6664 (2011) 293–307.

12. G. Ciobanu, C. Juravle. Flexible Software Architecture and Language for Mobile Agents. *Concurrency and Comput. Practice and Experience* **24**, 559–571 (2012).
13. G. Ciobanu, M. Koutny, and L. J. Steggles: A Timed Mobility Semantics Based on Rewriting Strategies. In: G.Eleftherakis, M.Hinchey, and M.Holcombe (eds), Proc. of *SEFM'12*, *Springer, LNCS 7504* (2012) 141–155.
14. G. Ciobanu, A. Rotaru. A Probabilistic Logic for pTiMo. *Lecture Notes in Computer Science* **8049**, 141–158 (2013).
15. G. Ciobanu, M. Zheng. Automatic Analysis of TiMo Systems in PAT. *Engineering of Complex Comp. Systems (ICECCS)*, IEEE Computer Society, 121–124 (2013).
16. G. Ciobanu, M. Koutny, and L. J. Steggles. Strategy based semantics for mobility with time and access permissions. *Formal Aspects of Computing*, 27(3):525–549, 2014.
17. G. Ciobanu, T.S. Hoang, A. Stefanescu. From TiMo to Event-B: Event-Driven Timed Mobility. *ICECCS 2014* (best paper), IEEE Computer Society, 1–10 (2014).
18. G. Ciobanu and M. Koutny: PerTiMo: A Model of Spatial Migration with Safe Access Permissions. *The Computer Journal* 58(5) (2015) 1041–1060.
19. M. Clavel, *et al.*: Maude as a Metalanguage. In: C. Kirchner and H. Kirchner (eds), Proc. of *WRLA '98*, *Electronic Notes in Theoretical Computer Science* 15 (1998).
20. M. Clavel, *et al*: Maude: Specification and Programming in Rewriting Logic. *Theoretical Computer Science* 285(2) (2002) 187–243.
21. M. Clavel, *et al. Maude Manual (Version 2.7)* http://maude.cs.illinois.edu/ Accessed April 2016
22. S. Dasgupta, D. Potop-Butucaru, B. Caillaud and A. Yakovlev: Moving from weakly endochronous systems to delay-insensitive circuits. *Electronic Notes in Theoretical Computer Science* 146 (2006) 81–103.
23. S. Eker, M. Knapp, K. Laderoute, P. Lincoln, J. Meseguer, K. Sonmez. Pathway Logic: Executable models of biological networks. In: F. Gadducci, U. Montanari (eds.), *Proc. of WRLA 2002*, *Electronic Notes in Theoretical Computer Science*, 71:144–161, 2004.
24. S. Eker, J. Meseguer, A. Sridharanarayananb. The Maude LTL Model Checker. *Electronic Notes in Theoretical Computer Science*, 71:162–187, 2004.
25. S. Eker, N. Martí-Oliet, J. Meseguer, A. Verdejo. Deduction, Strategies, and Rewriting. *Proc. of STRATEGIES 2006*, *Electronic Notes in Theoretical Computer Science*, 174(11):3–25, 2007.
26. J. Hillston. Challenges for Quantitative Analysis of Collective Adaptive Systems. In: Abadi M., Lluch Lafuente A. (eds), *Trustworthy Global Computing. TGC 2013*, LNCS 8358, pages 14–21, Springer, 2014
27. A. J. Ijspeert, A. Martinoli, A. Billard, and L. M. Gambardella. Collaboration Through the Exploitation of Local Interactions in Autonomous Collective Robotics: The Stick Pulling Experiment. *Autonomous Robots*,11(2):149–171, 2001.
28. A. Martinoli, K. Easton, and W. Agassounon. Modeling Swarm Robotic Systems: a Case Study in Collaborative Distributed Manipulation. *The International Journal of Robotics Research*, 23(4-5):415-436, 2004.
29. N. Martí-Oliet and J. Meseguer. Rewriting logic as a logical and semantic framework. In: D.M.Gabbay and F.Guenthner (eds), *Handbook of Philosophical Logic (Second Edition)*, Vol. 9, pages 1–87, Kluwer Academic Publishers, 2002.
30. J. Meseguer: Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science* 96(2) (1992) 73–155.
31. V. Nigam, R. Donaldson, M. Knapp, T. McCarthy and C. Talcott. Inferring Executable Models from Formalized Experimental Evidence. Computational Methods in Systems Biology 9308, pages 90–103, Springer Verlag, 2015.

32. L. J. Steggles. Rewriting Logic and Elan: Prototyping Tools for Petri Nets with Time. *Applications and Theory of Petri Nets 2001*, LNCS 2075, pages 363-381, Springer Verlag, 2001.
33. L. J. Steggles, R. Banks, O. Shaw, A. Wipat. Qualitatively modelling and analysing genetic regulatory networks: a Petri net approach. *Bioinformatics*, 23(3):336-343, 2006.
34. M-O. Stehr, J. Meseguer, and P. C. Ölveczky Rewriting Logic As a Unifying Framework for Petri Nets. In: H. Ehrig, *et al.* (eds), Unifying Petri Nets: Advances in Petri Nets, LNCS 2128, pages 250–303, Springer Verlag, 2001.
35. A. Verdejo and N.Martí-Oliet: Two Case Studies of Semantics Execution in Maude: CCS and LOTOS. *Formal Methods in System Design* 27(1–2) (2005) 113–172.

Can We Ever Stop Them?

Jörg Desel

FernUniversität in Hagen, Germany
joerg.desel@fernuni-hagen.de

This short article is devoted to all Petri net researchers born in the late 1950s (like myself), which are thus around their respective 60th birthday. In particular, it is devoted to Maciej Koutny.

In industry, many people around 60 are already planning their retirement. The situation in academia is different; most professors at this age seem to be as productive as ever and ignore the benefits of retirement age. So the question tackled in this contribution is not whether it is possible to keep them from work *immediately* but to keep them from work *eventually*. But is there a switch which turns people in a state such that they stop working eventually? In terms of Petri nets, we can formulate this problem as follows:

> Assume a Petri net and a transition t of this net. Given any reachable marking m of the net, can we eventually stop the behavior of the net by forbidding occurrences of t, or, equivalently, does any reachable marking m enable an infinite occurrence sequence without occurrences of t?

Apparently, this question is also highly relevant for real applications of Petri nets. For example, given a robot (or any kind of machine) modeled by a Petri net, can some component modeled by a particular transition be used as a cut out? As known from our computers, immediate stops are not always desirable, but rather forced shut down processes.

The problem tackled in this article could be solved by any standard mechanism involving temporal logics. There exist standard model checking procedures for Petri nets and properties stated as a temporal logic formula. Instead, this article provides a solution which is purely based on Petri net analysis techniques.

Throughout this paper we consider place/transition Petri nets without inhibitor arcs. As usual, we assume that the sets of places and transitions of a place/transition net are finite.

Terminating Petri nets

To warm up, we first consider the question whether a place/transition Petri net terminates eventually, i.e., whether all its occurrence sequences are finite.

Obviously, a bounded place/transition net terminates if and only if its reachability graph has no cycles. In fact, if the reachability graph has a cycle, then each occurrence sequence from the initial marking to any marking represented by a vertex of the cycle can be extended infinitely, following the arcs of the cycle (remember that each vertex of the reachability graph represents a reachable marking). Conversely, a bounded place/transition net has only finitely many

reachable markings, because the set of places of the net is assumed to be finite. Since each occurrence sequence corresponds to a directed path of the reachability graph, each infinite occurrence sequence corresponds to a directed path that passes through at least one vertex more than once; thus the reachability graph has a cycle.

Unbounded Petri nets do not terminate anyway. To see this, consider the construction of the reachability tree. Since the set of transitions is finite, each vertex of this tree has finitely many immediate successors. By König's lemma, the tree has an infinite path, corresponding to an infinite occurrence sequence.

Hence, an obvious algorithm to check termination of a place/transition net first checks boundedness, for example by the coverability graph construction. In case the considered net is bounded, the algorithm constructs the reachabilty graph and checks whether this graph has a cycle. Actually, this two-step approach is not necessary, because the coverability graph of a bounded place/transition net equals its reachability graph and cyclicity of this graph is implicitly checked during the coverability graph construction. A perhaps more elegant algorithm[1] first adds a place to the net which has all transitions of the net in its pre-set and no transition in its post-set, and then checks boundedness of this place, for example by construction of the coverability graph. Obviously, this additional place is bounded if and only if the length of all occurrence sequences is bounded. Since the set of transitions is finite, this is the case if and only if there is no infinite occurrence sequence.

Termination after stopping a transition – the bounded case

We now come back to the question asked initially: Does a place/transition Petri net terminate if a given transition t of the net is stopped eventually? In other words: Is there an infinite occurrence sequence with only finitely many occurrences of t?

For bounded place/transition nets, there is again a very simple algorithmic solution: Construct the reachability graph and check whether every cycle of this graph contains at least one arc labeled by t. In fact, if there is a cycle without t-labeled arc, then – as above – some infinite occurrence sequence starts with a finite sequence to some vertex of this cycle (which might include occurrences of t) and then runs along the cycle infinitely. Conversely, assume that each cycle has at least one t-labeled arc. Each infinite occurrence sequence passes through some vertex of the reachability graph infinitely often. All (infinitely many) subsequences between two subsequent passes through that vertex correspond to a cycle. By assumption all these subsequences contain an occurrence of t, whence t occurs infinitely often in the sequence.

Algorithmically, we can delete all t-labeled arcs in the reachability graph (which does *not* necessarily lead to a connected graph) and check for cycles.

[1] communicated by Karsten Wolf

Dito – the Unbounded Case

Finally, we consider the case that the considered place/transition net is unbounded. Does it eventually terminate provided a given transition t occurs only finitely often? Unfortunately, the coverability graph does not bring immediate help. Consider the simple example of a place/transition net with only one place, which is initially unmarked, a single input transition i, and a single output transition o.

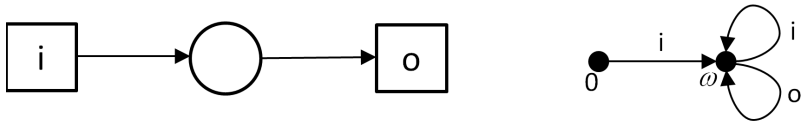


Fig. 1. A simple net and its coverability grap

We say that a transition t eventually stops a net if (and only if) every infinite occurrence sequence contains infinitely many occurrences of t. In the above example, transition i eventually stops the net, whereas transition o does not. However, both transitions occur in the coverability graph in quite the same way, namely as labels of arcs leading from the ω-marking labeled by ω to itself. These are the only cycles of this coverability graph. While the ready coverability graph does thus not lead to an algorithmic solution, we can solve the problem during its construction, as shown below.

Remember that, during the (nondeterministic) construction of the coverability graph, we compare new ω-markings with already constructed ω-markings. An ω-marking is a marking of the places of a net where some places can have the entry ω, meaning that these places can carry arbitrarily many tokens. More precisely, when a new vertex of the coverability graph is constructed, the algorithm compares the ω-marking m corresponding to this new vertex with the ω-markings m' corresponding to vertices which are on paths from the initial vertex to the new one. If, for all places, the new marking m is identical to m', then the new vertex is identified with the vertex corresponding to m'. Otherwise, if $m(s) \geq m'(s)$ for each place s (where $\omega > n$ for every integer n), then m is modified as follows: For each place s with $m(s) > m'(s)$ we set $m(s) := \omega$, because the sequence from the vertex corresponding to m' to the newly constructed vertex can be repeated arbitrarily often, leading to an unbounded token growth on the place s. In the above example, the marking reached by the occurrence of transition i is greater than the initial marking for the only place; hence in the coverability graph this place gets an ω-entry. Further occurrences of transition i are possible, leading to the same ω-marking, because ω already means "arbitrarily many". Notice, however, that transition i can occur infinitely often, no matter if transition o occurs, whereas o cannot occur arbitrarily often without i, and in particular there is no infinite occurrence sequence $o\,o\,o\ldots$, a fact which is not reflected by the coverability graph.

Now we come back to the problem whether some transition t eventually stops its net. To this end, we modify the coverability graph construction as follows. When adding a new vertex and comparing ω-markings with previously reached ω-markings, we also look at the occurrence sequences leading from the previously reached ω-marking to the current one. If all such sequences contain at least one t-transition, we proceed as in the original algorithm. Otherwise we consider the occurrence sequences without t leading from a previously reached ω-marking m' to the actual ω-marking m which satisfy $m'(s) \leq m(s)$ for each place. We define the effect of an occurrence sequence to a place s as the difference between the number of occurrences of output transitions in the sequence and the number of occurrences of input transitions of the sequence. I.e., by the occurrence of the sequence, the token count on s is decreased or increased by the effect of the sequence to s. If $m(s) \neq \omega$ then the effect of the occurrence sequence to s must not be negative by construction. However, if $m(s) = \omega$, then the occurrence sequence might actually decrease the number of tokens on s, as it happens in our example by the short occurrence sequence o.

If we find a sequence (without t) from some suitable previously reached marking m' to m with non-negative effect to all places s, then we stop the algorithm with output *no*, i.e., the algorithm found out that transition t does not stop the net. Otherwise we proceed as in the usual construction of the coverability graph. If the construction algorithm reaches its regular end, i.e., if it never answered *no*, then it delivers the output *yes*, thus identifying that t actually stops the net.

If we apply our modified algorithm to the above trivial example and ask whether o stops the net, then we immediately identify the occurrence sequence $m_0 \xrightarrow{i} m$ which neither contains o nor has a negative effect on any place (but a positive effect on the only existing place). So the algorithm terminates with output *no*. If we apply it with respect to transition i, then the only relevant cycle is given by the arc labeled o, which is actually a loop. The short occurrence sequence o decreases the token count of the only existing place. So it has a negative effect to this place. Therefore, the algorithm finally constructs the complete coverability tree and ends with the output *yes*.

To prove the algorithm correct, we first observe that it proceeds like the usual coverability graph construction algorithm, except that it might terminate earlier. So it terminates eventually, as the unmodified coverability graph construction algorithm terminates eventually.

If the algorithm terminates with ouput *no*, then there is an ω-marking in the coverability graph constructed so far which enables an occurrence sequence without occurrences of t and with non-negative effect to all places. Remember that an ω-marking enables a finite occurrence sequence if the regular marking constructed by replacing all ω-entries by the length of the sequence enables the occurrence sequence (this replacement ensures that none of the transitions of the sequence lacks tokens on places marked by ω). By construction of the coverability graph, we can actually reach such a regular marking by pumping up the tokens on all ω-marked places. Since the occurrence sequence has no negative effect to any place, the marking reached by the sequence assigns at least as many tokens

to each place as the marking enabling the sequence. Therefore, the occurrence sequence can be repeated infinitely often. Thus the transition t does not stop the net eventually.

Conversely, assume that a transition t does not stop the net eventually. We proceed indirectly and assume that the algorithm stops with output *yes*, thus constructing the full coverability graph. Since t does not stop the net eventually, there exists a reachable marking m that enables an infinite occurrence sequence without t. In this occurrence sequence, we reach markings m' and m'' (reached after m') such that $m''(s) \geq m'(s)$ for each place s (this is the core of the proof of finiteness of the coverability graph, based on Dickson's Lemma). Let σ be the occurrence sequence leading from m' to m''. Clearly, σ also does not contain t, and it has a non-negative effect to all places. It is known that the ω-markings of the coverability graph cover all reachable markings. Hence some ω-marking m'_ω covers m', i.e., $m'_\omega(s) \geq m'(s)$ for each place s. During the construction of the coverability graph the algorithm will find out that m'_ω enables σ, which leads to another ω-marking m''_ω covering m''. However, comparing m''_ω with m'_ω and considering the occurrence sequence σ would lead to an earlier termination of the algorithm with output *no* – a contradiction.

Conclusion

We have shown how to decide whether a single transition is able to stop an entire net evetually. The proposed algorithm can easily be generalized to sets of transitions (if we forbid all transitions of this set at some time, will the net eventually stop?). Another obvious generalization refers to arc weights; the procedure works for nets with arc weights with only very little changes.

Another tool for identifying transitions that stop a net is given by transition invariants, which are closely related to cyclic occurrence sequences, or by transition sur-invariants, which are related to occurrence sequence with non-negative effect to all places. Both types of invariants can be derived by linear algebraic means. These techniques lead to much more efficient algorithms, but unfortunately provide only sufficient criteria for termination problems.

Please notice that we want to stop nets *eventually*. Applied to human researchers, this means that they should find an end in some years, not immediately. In any case, it is important to find the right transition, i.e., the right way to allow the researchers to concentrate on other beautiful things in live. In the case of Maciej, it might be Martha's job to find the right switch of Maciej, corresponding to such a transition early enough – and vice versa.

Detecting and Mitigating Cyberattacks against Microservice-Based Controllers for the Digital Factory (Extended Abstract)

Gabriele Gualandi, Emiliano Casalicchio,
Emanuele Gabrielli, and Luigi Vincenzo Mancini

Department of Computer Science, Sapienza Università di Roma, ITALY
lv.mancini@di.uniroma1.it

Keywords: Cybersecurity · Digital Industry · Cyber-phyisical systems

Evolution of the industry comes with the need of redefining the organization and management of the entire value chain along the lifecycle of products. The main goal is to improve flexibility in the production, for increasing the quality of the final products while using fewer resources. The improvements expected in the industry are supported by the recent advances in wireless sensor networks, artificial intelligence, big data and cloud computing. Such advances allow Cyber-Physical Systems (CPSs) to reduce the distance between digital computing and the physical world. By letting a digital system to perceive the environment and to actuate decisions (i.e. to "control"), the concept of feedback can be ported at many different levels of a value chain.

As an industry is a complex and heterogeneous eco-system, one of the main challenges is to integrate different domains of information which are historically seen as separated. For example, information related to business policies, market demand and supply, productivity trends, just to name a few, has to be integrated in a harmonious way with data sources of any kind (e.g. from data of sensors to decisions of human operators). Concrete scenarios and models have already been proposed under the name of Industry 4.0[1] or Digital Factory in what follow. What emerges from these studies is that a digital factory is indeed a complex system from many points of view. The complexity of the controller responsible for a whole Digital Factory increases with the multiplicity of heterogeneous data sources and actuators. Experts from different fields have to cooperate when developing the logic of the controller(s) and the relative integration with the systems or processes to be controlled. Moreover, there is the need for a continuous refinement and evolution of all the digital systems within a Digital Factory, including the logic of the controller and in related aspects (e.g. integration, deployment). As a complication, in a CPS the physical computational resources used by the digital systems are typically distributed, and there is the need to tolerate unexpected loads or failures which result from changes in the environment. Finally, CPS and the controller are targets for cyber attacks intended to leak information, compromise or destroy the Digital Factory.

The usage of a Service Oriented Approach has been proposed for realizing CPSs[2]. Among the Service Oriented approaches, the microservice approach was recently proposed for implementing CPSs [3] with the goal of solving limitations of current architectures in the scenario of Digital Factory where complexity, speed of evolution and required resources of applications is expected to increase in the coming years.

Microservices[4] is one of the latest architectural trends in software engineering, promising to address several open issues in software development. A microservice architecture is suitable to realize complex, distributed architectures, in which multiple teams of developers cooperate in all steps of software construction, from integration, testing, releasing to deployment and infrastructure management. This is reasonably in line with the requirements of a Digital Factory, where a complex, distributed application (in our case a controller - more likely a set of controllers hierarchically organized) has to evolve while remaining continuously available. Recently, has been proposed the integration of IoT technologies with a microservice architecture [5] with the goal of solving limitations of current architectures in the scenario of a Digital Factory. The IEC 61499 is an industrial standard proposing component-oriented models for developing distributed control systems. Given the modular nature of the microservices approach, we consider possible the implementation of an architecture following industrial standards such as the IEC 61499 into a microservice architecture.

Different microservices approach comes with different guidelines and patterns, having as a common point that of carrying advantages for developing, evolving and deploying complex applications. In this work we consider a Digital Factory adopting the Command Query Responsibility Segregation (CQRS) pattern [6] to implement the control plane. A CQRS application, the controller in our case, is split into two parts: command processing (i.e. inputs to the controller) and query processing (i.e. outputs of the controller). A CQRS application contains an Event Store, which can be used to determine the current and past state of the system, i.e the state of the controller.

Our contribution is twofold. First, we propose an architecture for implementing the controller as a set of containerized microservices that can be automatically deployed and orchestrated on cloud platforms. Compared to a monolithic design, our proposal has advantages when the controller is complex, constantly evolving and requires many computational resources.

Then, we focus on how to detect and mitigate cyber attacks to the controller. In our CQRS based controller, if the query or command services are compromised, the control system could be compromised as well, with catastrophic consequences for the Digital Factory.

The proposed solution consists of a detection and a mitigation mechanism. The detection technique involves the usage of redundant replicas with a voting scheme applied both to input and to the outputs of the controller. The mitigation is implemented by destroying and restoring the compromised microservices from their original image, together with a safe state taken from the Event Store. In the paper, we describe the CQRS-based architecture of the controller enhanced

with the detection and mitigation schema and we show, by means of simulation, the effectiveness of the detection and mitigation mechanisms.

References

1. Zhong, Ray Y., et al. Intelligent manufacturing in the context of industry 4.0: a review. Engineering, 2017, 3.5: 616-630.
2. Feljan, A. V., Mohalik, S. K., Jayaraman, M. B., & Badrinath, R. (2015, December). Soa-pe: A service-oriented architecture for planning and execution in cyber-physical systems. In Smart Sensors and Systems (IC-SSS), International Conference on (pp. 1-6). IEEE.
3. Lee, J., Bagheri, B., & Kao, H. A. (2015). A cyber-physical systems architecture for industry 4.0-based manufacturing systems. Manufacturing Letters, 3, 18-23.
4. Kang, H., Le, M., & Tao, S. (2016, April). Container and microservice driven design for cloud infrastructure devops. In Cloud Engineering (IC2E), 2016 IEEE International Conference on (pp. 202-211). IEEE.
5. Thramboulidis, K., Vachtsevanou, D. C., & Solanos, A. (2018, May). Cyber-physical microservices: An IoT-based framework for manufacturing systems. In 2018 IEEE Industrial Cyber-Physical Systems (ICPS) (pp. 232-239). IEEE.
6. Betts, D., Dominguez, J., Melnik, G., Simonazzi, F., & Subramanian, M. (2013). Exploring CQRS and Event Sourcing: A journey into high scalability, availability, and maintainability with Windows Azure.

Causality's Revenge

Werner A. Hofer

School of Natural and Environmental Sciences,
Newcastle University,
Newcastle upon Tyne NE1 7RU, UK
werner.hofer@ncl.ac.uk

Abstract. I revisit the Bohr-Einstein controversy of 1935. Bohr's assertion that there are no causes in atomic scale systems is, as a closer analysis reveals, not in line with the Copenhagen interpretation since it would contain a statement about reality. What Bohr should have written is that there are no causes in mathematics, which is universally acknowledged. The law of causality requires physical effects to be due to physical causes. For this reason any theoretical model which replaces physical causes by mathematical objects is creationism, that is, it creates physical objects out of mathematical elements. I show that this is the case for most of quantum mechanics.

1 A different kind of research

The article you are reading was initially published under a different title on the internet archive [1]. It was the final assessment of quantum mechanics, distilled itself from a book I had been writing in 2016 and 2017, showing that the theory was flawed because it lacked physical causes for physical effects. Alex Yakovlew, who knew the book and had read the article, thought it perfect for the Festschrift at Maciej's sixtieth birthday, who, after all, had extensively worked on causality. As a Dean, I had worked extensively with Maciej, so this article is dedicated to his birthday. It shows that the problem in modern theoretical physics is a dysfunctional relationship with causality.

This article, which covers the last two years of my research, also did not result from the usual working practices of a theoretical physicist. Such practices typically involve the scribbling of mathematical symbols on a whiteboard or a piece of paper, the rearranging and replacing of these symbols and their evolution until a final set of symbols is reached which seems to make sense and can then be put between the text lines in a scientific article.

This research evolved more or less out of a deep meditation on a few words in Bohr's original paper from 1935, where he tried to prove Einstein wrong, who had accused quantum mechanics of being incomplete [2, 3]. The seven words in question are 'renunciation of the classical ideal of causality', and it took me quite some time to make sense of them. Renunciation to me sounded ominous, and it is perhaps no coincidence that an emissary of the Pope is a Nuntius, which could explain my slight trepidation, since I was raised a Catholic. The

other two words which took a long time to settle in my thinking were 'classical' and 'ideal'. It was not immediately clear to me what these two words had to do with causality. In fact, the more I meditated on them, the less sense they made. In my understanding, which had been philosophically trained on Kant, Schopenhauer and Nietzsche, causality was due to the fact that events happened in space and time and that one event could cause a subsequent one. What this had to do with 'classical', which I understand to mean, in music, the period from about Haendel to Beethoven, was not obvious. If the 'classical' posed problems, it became even worse for the 'ideal', because since when were causes conditioned by something 'ideal'? Things happen, because something causes them, which seems to be pretty much accepted across all disciplines in science and engineering. Why this should pose a problem for atomic physics, remained a mystery.

That is, it remained a mystery, until I began looking for causes in atomic physics and did not find any. The theory describing it only took you so far with causes, and at this point it all became mathematics. Call me suspicious, but this is exactly what a magician would do: he would lure you into a comfortable feeling telling you how he progresses with his magic trick, until at some point the trick is done and you don't know how it happened. This was, more or less, the case in all events in atomic physics I analysed. But as a consequence I did not, like most of my colleagues did over the years, sigh and get on with my mathematical calculations [4], but started to ask: Why are there no causes in atomic physics? and What causes physical effects in atomic physics? It took me quite some time to find simple answers to these two questions.

The answers, and the whole story how physics came to lose its causes are written down in a popular science book, which is about to be published [5]. However, I thought that my colleagues, and those too busy to read a reasonably priced book of 300 pages, might appreciate a free and much shorter executive summary. This is what the rest of the article delivers.

2 Mathematics and other languages

There are two famous statements about the relationship of mathematics and reality. The first is from Galileo Galilei, when he said that "Nature is written in the language of mathematics"[6]. His statement is based on the acceleration of mass in gravity fields and the observation that acceleration is constant and that the path a mass covers is proportional to the time interval squared. This, of course, is described very accurately in the laws of classical mechanics, developed by Newton after Galilei's death. One can interpret the statement in two possible ways. The first would be that mathematics is a language like any other. The difference being that it is not organised along the laws of a particular grammar, which govern languages, but according to the laws of logic, which govern mathematics. In principle, however, there is no difference between mathematics or any other language. The other interpretation would be that mathematics has a special relationship with Nature. This interpretation, as will be seen, has led to some confusion in the scientific community, as it often is taken to mean that

mathematics somehow provides a closer match with reality than other languages could.

Fortunately for scientists in the 21st century, not even mathematicians claim that their mathematical objects are physical objects in real space, possibly with one exception, see below. If any mathematician would make this claim, then she would have to explain why, for example, nobody has ever seen a perfect triangle float in space. So it is fairly safe to assume that mathematics is a particular form of language.

This has been the topic of a heated debate in the middle ages which has, at the beginning of the modern age, led to a clear solution. The debate in the middle ages was between factions which were then called the 'realists' and the 'nominalists'. Realists believed that the elements of language were real. So they searched, quite understandably, for the original language, the one that would reflect Nature closest, which they thought might have been the language Adam and Eve used in paradise. In their view, elements of this original language are real, and they exist in the real space of our everyday life. Realists traced their belief back to the Greek philosopher Plato. Nominalists, on the other hand, did not believe that language was real. For them the elements of a language were a social construct and due to a common agreement about their exact meaning. This faction can be traced back to William of Ockham, of Occam's razor fame.

Nominalism is the common principle agreed by modern science in the 17th century. Language, according to this agreement, is a social construct to enable communication between humans and does not exist in reality. Mathematics, according to this understanding, is also not part of reality.

This consensus has been violated by physics in the 20th century. Initially, this was probably clearest expressed by Eugene Wigner when he said that: "The first point is that the enormous usefulness of mathematics in the natural sciences is something bordering on the mysterious and there is no natural explanation for it. Second, it is just this uncanny usefulness of mathematical concepts that raises the question of the uniqueness of our physical theories"[7]. The two points Wigner raised are very profound. Because it is indeed the question why ready-made mathematical concepts like Riemann geometries, Hilbert spaces, Hermitian matrices, Lie algebras and the like would be useful to describe reality in physics.

It will be seen in the following sections that the reason for this feature of physics, in particular quantum mechanics, is that it is largely devoid of causality and ascribes the ability to have physical effects to elements of the mathematical language. The belief that mathematical objects, which do not exist in real space, can cause physical effects in the real world, is very similar to the belief that a God, who does not exist in the real world, created this world. Philosophically, there is no difference between a Bible-belt Christian, who wishes fire and brimstones onto the infidels, and a University quantum theorist, who wishes mathematical symbols to change reality: they are both, at heart, creationists.

Let me finish this section with a quote from MIT physicist Max Tegmark, who seems to have completely lost the ability to differentiate between a language describing Nature and Nature herself. Tegmark's main contribution to the de-

bate is the Mathematical Universe Hypothesis (MUH), which says that "Our external physical reality is a mathematical structure"[8]. Interestingly, his argument is based on the assumption that there exists an external physical reality independent of us humans. The present article is based on exactly the same assumption. Unfortunately, Tegmark's hypothesis that the objects of mathematics are in fact objects in the real world is exactly the same as the assumption of medieval realists. This hypothesis does not agree with the nominalist view, which is the basis of modern science. Linguists will probably also have a word to say about such an idea.

The point to remember from this section is that language is a description of Nature or reality, which in itself has no reality and depends on common consent. Mathematics, it should also be remembered, is only one particular kind of language.

3 The case against Bohr

To an unbiased observer, a mechanical engineer, say, or a sociologist, it must seem strange that there exists a distinct difference between experiments and theory in modern physics. While experiments at the atomic scale have improved and the precision and abilities to manipulate systems reached dizzying heights which really seem to allow us to create new materials and new structures from the bottom up, theory seems to be stuck in a time warp, which always reverts back to the 1930s. A historian would probably conclude that something happened in the 1930s, which theoretical physics still has problems to overcome. It is hard to overlook the similarities in the political sphere where, for example, the members of the Orange Order in Northern Ireland still march on the 1st of July every year to reassure themselves by a commemoration of the Battle of the Boyne, which happened in 1690 [9]. The reason for this, I would suggest, is the controversy between Einstein and Bohr, and the fact that this controversy transformed theoretical physics from a tool to describe reality by mathematical means to an ideology, which henceforth sought by every means possible to police the opinion that there is no mathematical theory beyond quantum mechanics. Here is, what happened.

We shall come to the case against wavefunctions in the next section, but wavefunctions, by about 1930, became a problem for the logical analysis of events at the atomic scale. Einstein, with his colleagues Podolsky and Rosen, pointed this out in an article in 1935 [2]. The following is a slightly simplified description of the measurements and processes Einstein, Podolsky, and Rosen (EPR) considered. They assumed that two electrons are emitted from an atom in two opposite directions at very high speed. They still retain their common wavefunction as they fly along. Their wavefunction has one component, which is spin-up and one component, which is spin-down. But since these two electrons still have a common wavefunction, we do not know, which electron is spin-up and which electron is spin-down. To find out, we position magnets in the path of the electrons, at equal distance from the atom, which measure their spin. Magnet A, along the

right path at a very large distance from the atom emitting the electrons. And there is magnet B, along the left path also at a very large distance. As magnet A measures the spin of the electron, it is observed that the electron is pushed upwards: this, says a physicist observing the result, is proof that the spin state of the electron arrived at A is spin-up. But here comes the rub: since the two electrons together have zero spin, the spin of the electron measured at magnet B must be spin-down. The physicist at magnet A could either measure the spin or not. If she does not, then her colleague measuring at B could obtain both results, either spin-up or spin-down. If she does, then her colleague can only measure the opposite spin to the one measured at A. So clearly, the measurement at magnet A does change the measurement at magnet B. However, the two measurements are close to simultaneous, as the two electrons have the same speed. So, there is no time for any information from A to reach B in the available interval.

This, said EPR, was proof that either quantum mechanics violated the principle, that nothing is faster than the speed of light, or that there was additional information, which was not contained in the wavefunction and which made the two measurements related.

This problem has now been solved within the model of extended electrons, the solution was recently published [10]. The correct answer is that each electron will carry a phase information, which is revealed at the moment of measurement and which connects the two measurement events. The same is true for photons. So wavefunctions do indeed contain additional information, their phase, which is not considered a physical property in standard quantum mechanics, but nevertheless included in the mathematical description of the problem. The additional information EPR referred to was there all along, but not considered as such.

Bohr, of course, did not know this when he answered that Einstein's arguments "would hardly seem suited to affect the soundness of quantum-mechanical description, which is based on a coherent mathematical formalism covering automatically any procedure of measurement like that indicated" [3]. And, of course, reality "must be founded on a direct appeal to experiments and measurements" [3]. These statements are the statements of a creationist. It is helpful to first analyse the main components of the statement, and then translate the statement into another language to see the problem.

From the viewpoint of a logical analysis, two terms are problematic in the first sentence. These two terms are "coherent" and "automatic". What they indicate, without any formal proof, is that a mathematical construct exists, quantum mechanics, which is a comprehensive description of reality (this is what coherent implies), and that this construct inevitably leads to all possible measurements one can think of (this is the meaning of the word automatically). Apart from the quite stunning arrogance of the sentence, considering the development of experimental methods since Bohr's statement, which have nothing to do with the experiments which could be undertaken in 1935 (the electron microscope, for example, was only invented in 1936), this statement is not science, but close to religion, as a translation into another language reveals.

Assume that this is not the statement of a physicist, but a novelist, who claims that her book is "a coherent story of the world which covers automatically every possible future, humans can experience." If you ask yourself what sort of book this would be then you reach the conclusion that it probably is not a scientific textbook of any discipline, but rather the Holy Book of one of the great religions, the Bible, or the Quran, for example. So Bohr claimed, to be clear about this point, that a mathematical construct would, for all eternity, be a correct description of all possible aspects of reality in physics. And then he went even further.

"Indeed, the finite interaction between object and measuring agencies conditioned by the very existence of the quantum of action entails - because of the impossibility of controlling the reaction of the object on the measuring instruments ... – the necessity of a final renunciation of the classical ideal of causality and a radical revision of our attitude towards the problem of physical reality" [3]. Let us forget for a moment that causality is and has been a fundamental principle of all sciences for at least three hundred years. Because this is, where Bohr falls foul even of his own Copenhagen interpretation. One of the key statements of this interpretation is captured in the two sentences: "It is wrong to think that the task of physics is to find out how Nature is. Physics concerns what we say about Nature" [11]. But if this is the case then a "renunciation of causality" is not possible, because it would explicitly say that there are no causes in Nature. If this is not possible, then the question remains why there are no causes in quantum mechanics. This will be the topic of the next two sections.

To sum up this section, Bohr's reply to EPR is not only outdated today because the EPR problem has been solved by a causal model in real space. It also contains dodgy logic if compared to his own Copenhagen interpretation, is based on creationism when it assumes that a mathematical construct generates all aspects of reality, and contains assertions which are clearly not science, but religion. I suggest we bury his statements in the history books and get on with the science.

Einstein's view, by contrast, that quantum mechanics is an incomplete theory of atomic physics is vindicated. But the omission goes far beyond what even Einstein thought. It does not concern single elements of reality, which are missing in quantum mechanics, but a whole class of physical objects, which would allow to refer events in real space to physical causes. We shall see, how this works in practice in the next sections, but the final score sheet of the Bohr-Einstein controversy then reads: Einstein one, Bohr nil.

4 The case against wavefunctions

One has to be very clear what wavefunctions are, and what they are not. They are not, as Schrödinger thought initially, physical objects in real space like electromagnetic fields. This seems sometimes confusing, because the formalism looks very similar to the formalism in Electrodynamics, in particular if the wavefunction is written as a function of location, like $\psi(\mathbf{r})$. There is a simple way to

distinguish physical objects in space from mathematical objects, and the key question to differentiate is: Does this object contain energy? Every electromagnetic field contains energy, as does mass via the energy-mass relations. The wavefunction, by contrast, does not contain energy. Therefore it is not a physical object in space, but a mathematical object. This is very clear in the abstract formalism, where wavefunctions are objects in their own mathematical space, Hilbert space.

There are two fundamental problems with wavefunctions. The first problem is widely recognized in the community, and it has refused to go away, despite years of hard work by a large number of theoretical physicist. The second problem, which has not been recognized at all so far, is probably the much more important one. These two problems, combined, make wavefunctions not only contradictory entities, but elements of creationism.

The first problem is called the measurement problem and considered by physicists who really think hard about their science one of the fundamental problems in modern physics. The publications and various attempts to solve it are well documented in the literature, and the number of articles trying to account for it probably goes into the thousands. The problem is due to the fact that in a measurement the wavefunction is thought to collapse to its measured state. While this can be stated, it cannot be described consistently in the mathematical framework of quantum mechanics. There is, fundamentally, neither a cause, nor a physical model which would describe how this happens. This problem has led to increasingly weirder speculations about the relationship between the act of measurement, and physical reality, the weirdest one probably the assumption that every measurement creates a new universe. Physicists of this persuasion no longer talk about reality or the universe, but a multiverse which, according to some estimates, contains about 10^{100} universes. For an engineer this would probably indicate that physicists have lost their mind and that they are hallucinating weirder and weirder theories to account for a problem that their science seems incapable of solving. But the problem can actually be turned on its head by two simple question: What if there is no collapse? How do I measure what I measure without a collapse? My colleague Thomas Pope and I have developed a model of spin measurements based on these two questions, and it turns out that the problem can be solved with two simple assumptions: (1) The electron is an extended object in space, and (2) a magnetic field rotates the spin of electrons, which turns out to be a vector field. The solution has recently been published and presented at various conferences in 2017 [10]. This solves the first problem, and it shows that not the wavefunctions, but the densities are the crucial physical variables leading to a solution.

The second problem, which so far has been completely ignored, is the following little equation, which is due to Max Born [12] (I ignore the various physical units that will usually be added):

$$\psi^{\dagger}(\mathbf{r})\,\psi(\mathbf{r}) = \rho(\mathbf{r}) \tag{1}$$

Let us be clear about the meaning of this equation. The two objects on the left, the wavefunction and its dual, do not contain energy, they are objects in mathematical space. The object on the right, the density of electrons, contains energy, because electrons have mass, and it is an element not of mathematical space, but of real space. So this equation says that one can take two elements of mathematical space, multiply them, and one creates an object in real space which contains energy. Every time, says the equation, a theoretical physicist takes the square of the wavefunction, energy magically pops out of Hilbert space and appears in real space. This is, without doubt, creationism in its purest form.

Now a traditionalist might try, at this point, to stall the analysis by claiming that the density is not really a physical object, because, after all, it only shows up in statistics. This might have been a valid argument in 1935, but it is no longer relevant in 2018. There are two reasons for this change. The first is that every year thirty thousand scientific papers are published which are based on a theoretical method called density functional theory (DFT) [13]. In DFT the only physical variable, which determines all physical properties of an atomic scale system is the density of electrons. This firmly roots the density in real space and as a continuous variable. The second reason is that density itself cannot be a statistical property, because this assumption is in conflict with high-precision experiments on metal surfaces, as shown in 2012 [14]. The electron density, not the wavefunction, is the primary physical variable of atomic scale systems. And it is a physical object in real space, not a mathematical object in Hilbert space. So the conclusion remains: the equation describes an impossible relationship of mathematical objects in Hilbert space and physical objects in real space. It is creationism, not science.

The measurement problem, which also is a fundamental obstacle to understanding what happens in atomic scale systems and within the framework of quantum mechanics, can be ignored for most applications of the theory, by following Mermin's recipe to shut up and calculate. This problem, however, cannot be ignored. Because it says, in a nutshell, that quantum mechanics is fundamentally not a causal theory. Not because there are no causes in Nature, which is what Bohr had tried to argue, but because at the point where one commonly expects a cause in a theoretical framework, one gets a mathematical object in Hilbert space.

Within the standard framework, and contrary to the framework of DFT, there is no cause that would make the density attain a particular value. This problem, it turns out, is not only unsolvable, it also makes quantum mechanics unscientific. Creationism is not science, rather the opposite, whether this is within the context of a religion, or within the context of mathematics.

5 The case against spin

The spin angular momentum, or spin as I will call it in this section, is probably the most difficult concept introduced in quantum mechanics. The difficulty arises from the fact that it cannot in any way be captured by an image in real space.

The usual image, a vector which points up or down, which is used in many scientific papers, does not do it justice, as spin is isotropic in the absence of a measurement, therefore not a vector. The problem, in a nutshell, is the following.

In a Stern-Gerlach experiment silver atoms are detected after an inhomogeneous magnetic field at two distinct points: one point off centre in the direction of higher field strength, one point in the direction of lower field strength[15]. The result indicates that the outermost electron of a silver atom has exactly two possible magnetic dipoles, none of them due to the electron orbit. The original experiments were done with silver atoms, but were later repeated with hydrogen atoms with identical results. What actually happens to the electron in silver or hydrogen has until recently been unknown and it poses quite an interesting logical challenge. Assume that the electron's magnetic dipole points in a particular direction, and assume that it is random, then the experimental result must be an extended blob. One could now assume, that the direction is not random, but that one class of electrons has a vector which points up, another class has a vector which points down. This would agree with the experiments.

But if the magnet, which determines the trajectory of the atoms, is turned by a quarter rotation, one would measure the exact same result: two points where the atoms impinge on the detection plate, now the points are offset left and right. If the vectors have a particular direction, then every possible direction would make the experimental results different for different rotations of the magnet. This means the vector cannot have a particular direction. Since it is a fundamental property of every vector that it points into a specific direction, a vector which does not point into a specific direction is a contradiction: so whatever determines the magnetic properties of a hydrogen electron is not a vector, but isotropic. So the electron seems to have a magnetic dipole, which is not only not a vector, but a magnetic dipole which only expresses itself as a vector if it is measured. Both problems have remained profound difficulties for the understanding of the electron until very recently.

The main problem becomes obvious if one asks a simple question: What pushes the silver atom up (down)? The only answer to this question, which is physically possible, is: A magnetic moment, which interacts with the inhomogeneous field of the magnet. But as spin is isotropic, it cannot be a magnetic moment which is a vector. Therefore one has to ask, how an isotropic object becomes a vector, and by what physical process? Described in this way it is obvious that there is no physical process. Instead, there is a similar transformation from mathematical space to real space and from a mathematical object to a physical one. Only in this case it does not involve, as it did for wavefunctions and densities, the creation of energy from Hilbert space, but the creation of a magnetic moment from Hilbert space. In the Pauli equation the relevant term which accomplishes this creation is the so called Stern-Gerlach term [16] (the last term on the right).

$$i\hbar\frac{\partial}{\partial t}|\psi\rangle = \left(\frac{(\mathbf{p}-q\mathbf{A})^2}{2m} - q\phi\right) I|\psi\rangle - \frac{q\hbar}{2m}\sigma\mathbf{B}|\psi\rangle \tag{2}$$

Again, there is no physical cause for the magnetic moment to have a particular direction, there only is a mathematical object with the necessary properties, in this case a Hermitian matrix of a two-dimensional Hilbert space, which accomplishes the transformation. It is, fundamentally, an act of vector-creation from Hilbert space.

One can trace back the logical difficulty to account for these measurements to a single property, tacitly assumed in all of quantum mechanics: the assumption that electrons are point particles [17]. If the electron is a point, then its spin cannot be a vector, because it would have to point into one specific direction. If, however, the electron is an extended object, then spin can be a vector, or rather a vector field. In this case it can also be an isotropic vector field, for example pointing into the radial direction of a sphere. Then the question what pushes the silver atom up or down, is easy to answer: the rotation of the vector field into the direction of the magnetic field vector. Such a process automatically aligns the spin direction with the direction of the external magnetic field, and it will lead to trajectories which are influenced by the field gradient. The model has recently been introduced and it has been shown that it is free of the usual paradoxes [10].

To sum up the result of this section we find that accounting for spin measurements in quantum mechanics also entails an act of creation, the creation of a vector from a Hermitian matrix in Hilbert space. Again, this is creationism and not science.

6 The case against Bohm

On the face of it, the antagonism against Bohm's reformulation of the Schrödinger equation in the 1950s seems quite stunning [18]. Because all he really did, was to rewrite the Schrödinger equation using a particular form of a wavefunction $\psi = R\exp(iS/\hbar)$, where both R and S are real-valued variables. This decouples the real and the imaginary components of the Schrödinger equation. If one now compares the real part of the equation with the Hamilton-Jacobi equation of classical mechanics, one finds an additional potential, which is commonly called the quantum potential Q [19]:

$$\frac{\partial S}{\partial t} = -\left[\frac{|\nabla S|^2}{2m} + V + Q\right] \qquad Q = -\frac{\hbar^2}{2m}\frac{\nabla^2 R}{R} \tag{3}$$

It has the dimension of a potential like the electrostatic potential V. Contrary to a conventional potential like V it does not depend on the physical environment of an electron, but on the shape of its wavefunction via the second derivative of the amplitude R. Note that at this point the equations Bohm derived are not different from the original Schrödinger equation, because they have been obtained by a general ansatz for the complex valued wavefunction ψ, and a simple analysis of the real and imaginary parts of the ensuing equations. The imaginary part can be linked to the continuity equation.

There is quite a large community of physicists who consider themselves Bohmians, and it is indeed tempting to assume that all that makes quantum

mechanics different from, say, classical mechanics, is a special potential which only shows up in atomic scale systems. Since this picture is, intuitively, much more satisfying than simply following the agreed recipe and calculating things without being able to picture them in the mind, it is hard to reject out of hand. However, if one accepts that this potential is what makes quantum mechanics different from our everyday environment, then one will have to accept the properties of the potential also as something which belongs to the quantum domain and is not found in an everyday environment. This is, where things become difficult intellectually.

The first problem arises, if one considers the elements which make up the quantum potential Q. A potential, which changes the energy content of space where it exists, is always a physical object in real space. This is, why Bohm originally called his theory the "causal" interpretation of quantum mechanics. The point-like electron, he thought, would change its trajectory according to the value of the potential. However, the components making up the potential are the amplitude of the wavefunction and its derivative. The wavefunction is, as already emphasized in previous sections, not an element of real space, but an element of mathematical space, Hilbert space, and it does not contain any energy. So the first problem is again a problem of creationism: the potential is created from elements of mathematical space, energy magically pops out of Hilbert space and into real space. This is not the only problem, though.

Because the relationship between the amplitude of the wavefunction and the quantum potential means that this potential exists throughout the whole space, where the wavefunction exists, and it will change immediately if, for example, the physical environment changes at one point of the system. The wavefunction then will not only change at this point but, via the second derivative and the normalization contained in the quantum potential, it will change throughout the system. Bohm's quantum potential is also non-local. If one now thinks of interactions between electrons via electromagnetic fields, then the electromagnetic fields will only interact with electrons to the extent that they have time to propagate to the point of interaction. The quantum potential, however, will interact with an electron instantly.

So while on the one hand the image that a special potential is what makes quantum mechanical systems different from other physical systems is intellectually satisfying, it is on the other hand hard to accept that in this case one will have to give up causality, because this potential is created from mathematical space and there is no physical mechanism which would allow me to understand how this quantum potential actually operates in space. Bohm's reformulation of the Schrödinger equation leads to exactly the same problem as in the original theory: there are no causes.

This could be the endpoint of the analysis and one could then conclude that Bohm's theory is probably not a way to regain causality in quantum mechanics. But one could also go one step further. If it is accepted that Bohm's theory is non-local and a-causal, then one could ask what this means for the original theory described by the Schrödinger equation. Formally, Bohm's equations are

not different from the original equation, because his ansatz for the wavefunction is generally valid.

It is then hard to see how one set of equations makes a theory manifestly non-local, while the logically equivalent set of equations does not have this deficiency. So the question remains: Is wave mechanics itself non-local? Tentatively, I would suggest that the answer to this question is yes. Then one has to understand where this non-locality would come into the original theory. The best candidate, I think, for this is the fact that one obtains physical properties of electrons, which are considered point particles in the conventional theory, only if the operator equations are integrated over the whole space of the wavefunction. So, for example, for the hydrogen electron this means an integration over infinite space, since the wavefunction exists as an exponentially decaying function over the whole space. I suggest that this procedure, the integration over infinite space, makes wave mechanics as non-local as Bohm's theory, where non-locality is made explicit in the form of the quantum potential. It is understood that non-locality also makes a theory a-causal. If this argument is correct, then what Bohm did was not to invent a new theory which would allow us to better understand atomic scale systems, he rather revealed that there is no way one can make quantum mechanics a framework based on causality.

7 Predictions and correlations

If there are no causes in quantum mechanics and if all physical effects are created from mathematical objects, that is elements of the mathematical language, what does this entail for the science described by quantum mechanics? A brief recourse to history will make it clear.

During Galilei's lifetime the conventional wisdom in astronomy was that the solar planets move around the Earth on trajectories described by epicycles on top of circles. The observations of Mars, for example, would show exactly such a behaviour. Also in this case the physical cause for its motion was unknown. So the mathematical model did not connect physical causes with physical effects, it connected mathematical objects (circles) with physical effects. Logically, what this theory describes is not a set of mathematically formulated predictions how the planet moves, but only a correlation between an observation (the position of the planet) and a mathematical model (circles). One part of the astronomical data could have, even before Kepler's observations, given away the fact that all was not well with epicycles, and this part concerned the velocities of planets along their trajectories, which were not constant. In atomic physics, the change of the wavelength of electrons as they change their velocity is also a fact that remained unexplained in quantum mechanics and could have alerted physicists long ago that all was not well in theoretical physics.

Predictions can only be made, if a mathematical model relates a physical cause to motion or other physical effects, and hence observations. Only Newton's theory of gravitation, which provided these causes forty years after Galilei's death, is capable of making these predictions.

The same applies to quantum mechanics. The mathematical models do not connect physical causes to physical effects, because there are no physical causes. The theory is therefore in principle unable to make predictions. All it provides are correlations between mathematical models and experimental observations. Historically, it involved the invention of mathematical objects to account for experiments. First, the invention of matrices in Heisenberg's matrix mechanics, then the invention of wavefunctions and spins in wavemechanics. Not one of these objects is a physical object in real space. A similar case in point, to be analysed in the future, will probably be provided by particle physics, which was forced to add a plethora of new mathematical elements, called particles, as the experiments progressed. Given that it is always possible to add new mathematical elements to the description if a model is not in line with observations, there is also no way to falsify such a theory. Quantum mechanics would, logically speaking, also fail Popper's test for a valid scientific theory.

Quantum mechanics, in short, is not science.

8 Conclusions

The inevitable conclusion from the analysis in the preceding sections is that a major part of modern physics, quantum mechanics, is creationism, in principle not falsifiable, and not science. This generates an interesting set of problems for theoretical physicists. One way to deal with the problems is to ignore the findings and to try to discredit the author of the paper. This is, what the establishment in physics did rather successfully with David Bohm, and it is probably safe to say that this will be the first reaction.

But will theoretical physicists be able to keep a straight face and the necessary authoritative demeanor when they teach quantum mechanics 101 in the future? Will they be able to stifle a snigger when they write down Born's equation or multiply a Pauli matrix with a field vector to obtain a magnetic moment? Not to mention the awkward possibility that students might start to call their Physics professors colloquially professors of Creationism, and rightly so. If this situation is already quite difficult to handle for a real scientist, the second problem is even worse. Because what will biologists think, who had to fight against creationism ever since Charles Darwin published his book? One can predict that physics, as a science, will lose much of the respect it currently enjoys in the scientific community. This leads to the third problem, which is finding a way to make physics a real science again. Here, the question is how much will have to be changed, and how much of the current conventional wisdom will have to be discarded for a future, strictly scientific, physics.

There is, unfortunately, no easy way out. The whole problem of creationism should have been addressed eighty years ago and not swept under the carpet by the faithful followers of Bohr. It should never have been allowed to fester and to impact on all subsequent theory.

An interesting question, which historians might want to investigate, is how much of the analysis in the current article had been understood by Bohr himself.

And if he understood the consequences, how did he think that a lack of causes could lead to a theory which describes physical effects in a scientific manner? How did he think he could keep divine mathematical interventions at bay which, as shown here, permeate the very foundations of the theoretical framework? My suspicion is that, in the end, it might come down to nothing more than personal arrogance, a lack of scientific humility, and a conviction to always be right.

For me the most frightening aspect of this analysis is what it says about us physicists. If quantum mechanics, which is one of the corner stones of modern physics, is actually not science but creationism, then how can we justify teaching our students the same nonsense? What they signed up to, when they entered University, was to get an education in a science discipline which gives them the expertise to understand and to work with the reality they are living in. Teaching them creationism, and calling it science, is irresponsible. So I would urge all of my colleagues in theoretical physics to analyse their own field along the same lines. Not by mindlessly heaping mathematical symbols onto a whiteboard and then, at some point, magically finding physical objects, but by analysing whether what their theory does is actually compatible with the laws of causality. If it is not, it has no place in science. My feeling is that this will probably apply to most of modern physics, not just quantum mechanics. Time, I would think, for a big bonfire of theoretical tradition.

References

1. The title of the original paper: Is quantum mechanics creationism, and not science? alludes to the titles of the articles in the famous Bohr-Einstein controversy. It was published in February 2018; arXiv: 1802.00227
2. A Einstein, B Podolsky, N Rosen, Can Quantum-Mechanical Description of Physical Reality Be Considered Complete? Physical Review 47, 777 (1935)
3. N Bohr, Can Quantum-Mechanical Description of Physical Reality Be Considered Complete? Physical Review 48, 696 (1935)
4. N D Mermin famously considered the Copenhagen interpretation to be equivalent to the order: Shut up and calculate! in What is wrong with this pillow? Physics Today 42, 9 (1989)
5. The book has the title: Mathematical Creationists - How Physics became a Religion and Chemistry conquered the World.
6. Galileo Galilei, Il Saggiatore (1620)
7. Eugene Wigner, The unreasonable effectiveness of mathematics in the natural sciences, Communications in Pure and Applied Mathematics 13, 1 (1960)
8. Max Tegmark, Shut up and calculate, arXiv: 0709.4024 (2007)
9. A colleague from Northern Ireland has made me aware that the Orange Order actually marches on the 12th of July every year, even though the battle happened, historically, on the 1st of July. The reason for the discrepancy is a change from the old Julian calendar to the Gregorian calendar in the 18th century. Interestingly, Guy Fawkes night, which commemorates an event on the 5th of November according to the Julian calendar, is still celebrated on the 5th of November, possibly because the 17th of November would not rhyme that well.
10. T Pope and W A Hofer, Spin in the extended electron model, Frontiers of Physics 12, 128503 (2017).

11. N Bohr, according to Wikiquote said after the Solvay Conference (1927)
12. M Born, Zur Quantenmechanik der Stossvorgänge, Zeitschrift für Physik 37, 863 (1926).
13. P Hohenberg and W Kohn, Inhomogeneous Electron Gas, Physical Review 136, B864 (1964).
14. W A Hofer, Heisenberg, uncertainty and the scanning tunneling microscope, Frontiers of Physics 7, 218 (2012).
15. W Gerlach, O Stern, Der experimentelle Nachweis der Richtungsquantelung im Magnetfeld, Zeitschrift für Physik. 9, 349 (1922).
16. W Pauli, Zur Quantenmechanik des magnetischen Elektrons, Zeitschrift für Physik 43, 601 (1927).
17. W A Hofer, Unconventional Approach to Orbital-Free Density Functional Theory Derived from a Model of Extended Electrons, Foundations of Physics 41, 754 (2011).
18. O Freire Jr., Science and exile: David Bohm, the cold war, and a new interpretation of quantum mechanics, see www.controversia.fis.ufba.br/index-arquivos/Freire-Bohm-HSPS.pdf
19. D Bohm, A Suggested Interpretation of the Quantum Theory in Terms of "Hidden" Variables, Physical Review 85, 166 (1952).

Complexity of Checking Output-Determinacy in General Petri Nets*

Petr Jančar[1] and Victor Khomenko[2]

[1] Faculty of Science, Palacký University Olomouc, Czech Republic
pj.jancar@gmail.com
[2] School of Computing, Newcastle University, Newcastle upon Tyne, UK
Victor.Khomenko@ncl.ac.uk

Abstract. Output-determinacy is an important soundness notion in the theory of non-deterministic asynchronous concurrent systems whose alphabet is partitioned into input and output actions. Intuitively, it postulates that the set of enabled outputs of a system must be fully determined by the visible history of its interactions with the environment, as otherwise the specification is contradictory in the sense that it simultaneously requires and forbids some output.
In the case of safe or k-bounded Petri nets checking output-determinacy was known to be PSPACE-complete, but the complexity (and even decidability) of this problem for general Petri nets was open. In this paper we show that the problem of checking whether output-determinacy is violated is decidable and equivalent to reachability in general Petri nets.

Keywords: Output-determinacy · Petri nets · Reachability · Computational complexity

1 Introduction

Labelled Petri nets (LPNs) with the alphabet (of transition labels) partitioned into input and output actions are often used for specifying concurrent systems (see, e.g., [2, 3, 7–9]). When a specification is deterministic (in the sense of automata and formal language theory), its semantics could be the set of its possible traces, i.e. its *language*. As the final implementation must be deterministic, it may seem reasonable to confine oneself to deterministic specifications only. However, often this turns out to be too restrictive in practice. There are several situations which naturally give rise to non-deterministic specifications which still can be implemented:

Silent transitions For convenience of modelling, the designers often use *silent* transitions, which are not included into visible traces of the system. Such transitions make the Petri net non-deterministic.

* Devoted to Prof Maciej Koutny on the occasion of his 60^{th} birthday.

OR-causality When a safe LPN is used for modelling a situation where the system has to respond to any of several possible stimuli in the same way, non-determinism naturally arises. (OR-causality can also be modelled as a non-safe (2-bounded) LPN without non-determinism [3, 9], but in practice safe Petri nets are preferable as they are much easier to analyse.)

Hiding signals Non-determinism naturally arises when in a deterministic LPN some of the signals are hidden (by turning some visible transitions into silent ones) – hiding signals is essential in many applications, e.g. the decomposition algorithm of [2, 7, 8].

It is thus natural to consider a non-deterministic model for which we could verify that there is a deterministic implementation.

We use the following formal model. An LPN $N = (P, T, F, I, O, \ell, M_N)$ is a structure comprising finite disjoint sets P, T of *places* and *transitions*, respectively; the *flow relation* $F \subseteq (P \times T) \cup (T \times P)$; disjoint sets I, O of *input actions* and *output actions*, respectively; the labelling function $\ell : T \to I \cup O \cup \{\varepsilon\}$ mapping each transition either to an action or to the empty word ε – such transitions are called *silent*; and the *initial marking* $M_N \in \mathbb{N}^P$ (for $\mathbb{N} = \{0, 1, 2, \ldots\}$).

We use the usual notation $M[\sigma\rangle$ to denote that marking M enables a sequence of transitions σ, and write $M[\sigma\rangle M'$ if σ is finite and enabled by M, and firing σ from M yields marking M'. We lift this notation to labels as follows, for ℓ extended to sequences of transitions. For a sequence of actions ν we write $M[\nu\rangle\rangle$ (resp. $M[\nu\rangle\rangle M'$) iff $M[\sigma\rangle$ (resp. $M[\sigma\rangle M'$) for some sequence σ of transitions such that $\ell(\sigma) = \nu$. In particular, a label l is considered enabled by a marking M, written $M[l\rangle\rangle$, if one can fire a finite sequence of silent transitions to directly enable a transition labelled by l, i.e. we have $M[\sigma\rangle M'[t\rangle$ where σ contains only silent transitions and $\ell(t) = l$.

In our constructions we often use pairs of arcs (p, t) and (t, p) (i.e. $(p, t) \in F$ and $(t, p) \in F$) for some place p and transition t. Such a pair will be called a *read arc* and depicted by a line without arrowheads (between a place-circle and a transition-box).

An LPN N specifies the behaviour of a system in the sense that the system must provide *all and only* the specified outputs and that it must allow *at least* the specified inputs. As a consequence, the system must be able to perform at least all traces of N. In fact, N also describes assumptions about the environment the system will interact with; namely, the environment will only produce the inputs specified by N. A correct implementation of N may allow additional input events (and traces), but these events and subsequent behaviour will never occur in the envisaged environment. In other words, when the system is running in a proper environment, only traces of N can occur.

The implementation may actually have fewer input signals than N, keeping only those that are relevant for producing the required outputs. In this case, the environment may provide irrelevant inputs, but the implementation simply ignores them — and in this sense, they are always allowed.

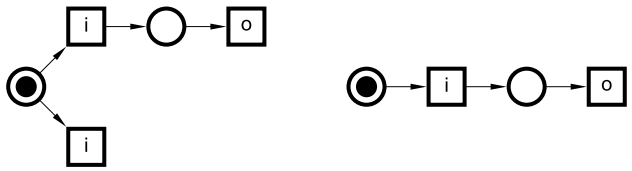


Fig. 1. The LPN on the left is not output-determinate; the result of determinisation is shown on the right. The latter LPN, though implementable and having the same language, is not a correct implementation of the original LPN: it can break the environment by producing o when the environment does not expect it.

Intuitively, assuming a deterministic implementation and possibly non-deterministic specification, the correctness of the implementation can be defined as follows (see [2] for a formal definition):

- the implementation must be able to perform all traces of the specification, maybe dropping some irrelevant input signals;
- after any trace, all the inputs allowed by the specification must be allowed (or ignored) by the implementation;
- after any trace, the implementation must enable exactly the specified outputs.

A non-deterministic specification can perform the same trace in two different ways, reaching different states (markings) M_1 and M_2. Assuming that the only information available to the system is the execution history, i.e. the trace performed, an implementation cannot determine whether its current state corresponds to state M_1 or M_2 of the specification. Hence, a deterministic implementation must behave consistently with the specification *no matter in which of these states the specification is*.

The above definition of correctness requires that the implementation must allow *at least* the inputs enabled by M_1 and *at least* the inputs enabled by M_2; this is easy to achieve even if these sets of inputs differ – i.e. the implementation may allow the union of these sets (or any superset thereof). However, the situation with outputs is different: The implementation must provide *exactly* the outputs enabled by M_1 and *exactly* the outputs enabled by M_2. This is only possible if M_1 and M_2 enable the same outputs.

This in particular implies that *the language is not an adequate semantics for non-deterministic specifications:* M_1 and M_2 may enable different sets of outputs, but the language cannot distinguish between such a system and its determinised version, yet the former has no deterministic implementations whereas the latter has (e.g. itself). For example, Fig. 1 illustrates a dangerous scenario when determinisation hides an error.

One possibility would be to define semantics based on some notion of bisimulation. However, it turns out that *bisimulation is unnecessarily strong* – it is

possible for two non-bisimilar systems to have exactly the same deterministic implementations. This has negative consequences for applications, e.g. some useful transformations preserving the possible implementations would be rejected if they do not yield a bisimilar system. For example, the decomposition algorithm of [8, 7] used semantics based on a variant of bisimulation (called 'angelic bisimulation'), and the semantics based on the notion of output-determinacy helped to improve it significantly [2].

In [2] a formal semantics of non-deterministic LPNs was proposed and justified. For this, the concept of *output-determinacy* (OD), which is a relaxation of determinism, was introduced. In particular, it was shown that for OD LPNs the language *is a sufficient semantics*, and this also holds in the case of distributed LPNs. Moreover, it was proved that an LPN cannot have deterministic implementations unless it is OD.

Definition 1 (Output-Determinacy). *An LPN N is called* output-determinate *(OD) if $M_N[\nu\rangle\rangle M_1$ and $M_N[\nu\rangle\rangle M_2$ implies for every output o that $M_1[o\rangle\rangle$ iff $M_2[o\rangle\rangle$.*

Therefore, OD is a useful correctness property that can be formally verified. Hence, the questions of decidability and computational complexity of this verification problem for various PN classes becomes relevant.

2 The complexity of checking output-determinacy

In [2] it was shown that the coverability problem is easily reducible to OD for safe/k-bounded/general PNs, which immediately yields the $\mathcal{PSPACE}$ lower bound for safe and for k-bounded PNs, as well as the $\mathcal{EXPSPACE}$ lower bound for general PNs. Moreover, for safe and k-bounded PNs an algorithm that runs in polynomial space was proposed. Hence checking OD is $\mathcal{PSPACE}$-complete for safe and k-bounded PNs. However, the upper bound and even decidability of this problem for general PNs were left open.

In this paper we show that finding a violation of OD for general Petri nets is polynomially equivalent to checking Reachability. Hence:

- The problem is decidable.
- The $\mathcal{EXPSPACE}$ lower bound in [2] is likely not tight, as Reachability is conjectured to be much harder.

We proceed by first establishing the improved lower bound, and then deriving a matching upper bound. Below, we denote by cOD the complement of OD, i.e. the problem of checking whether OD is violated.

2.1 The lower bound

We now show that Petri nets Reachability Problem (RP) is easily reducible to cOD, i.e. cOD is RP-hard. For simplicity, we use the special case of Zero

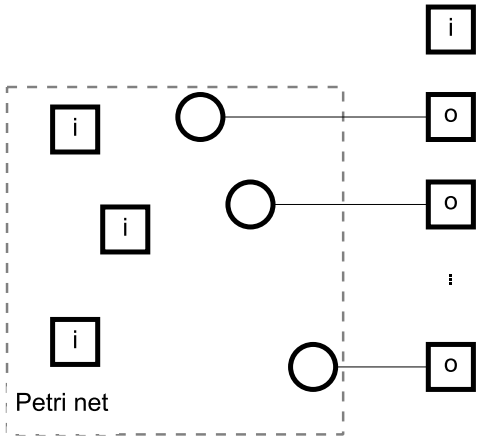


Fig. 2. Reduction from ZRP to OD.

Reachability Problem (ZRP), i.e. the reachability of the zero marking $\mathbf{0} = \{0\}^P$, that is known to be equivalent to RP [1].

Suppose one has to check whether the zero marking is reachable in a given (unlabelled) PN. We build an LPN as follows: If the initial marking of the PN is already $\mathbf{0}$, we return a fixed LPN violating OD. Otherwise we construct the LPN as follows, see Fig. 2:

- Each transition in the PN is labelled with the same input action i.
- For each place of the PN, a new o-labelled transition is created and connected to this place with a read arc; hence, o is always enabled in the resulting LPN as long as the marking is not $\mathbf{0}$.
- A new isolated transition $\hat{t}$ labelled with i is created; hence, i is always enabled in the resulting LPN.

Lemma 1. *Marking* $\mathbf{0}$ *is reachable in the original PN iff the constructed LPN violates OD. Moreover, the resulting LPN is safe/k-bounded/general if the original PN was safe/k-bounded/general.*

Proof. If the initial marking M_N of the PN is zero then the result is trivial, so we assume it is not and consider two cases:

(1) Suppose the zero marking is reachable in the original net via some non-empty execution σ, $M_N[\sigma\rangle\mathbf{0}$. The same execution can be performed in the constructed LPN yielding the trace terminating at the zero marking, $M_N[i^{|\sigma|}\rangle\rangle\mathbf{0}$, and by construction $\mathbf{0}$ does not enable o. Moreover, the LPN has another execution $M_N[\hat{t}^{|\sigma|}\rangle M_N$ yielding the same trace $i^{|\sigma|}$ but finishing at the initial marking that is not zero and thus enables o. Hence the LPN is not OD.

(2) Suppose the zero marking is not reachable in the PN, and hence in the LPN. This means that every reachable marking of LPN enables o, and so OD cannot be violated.

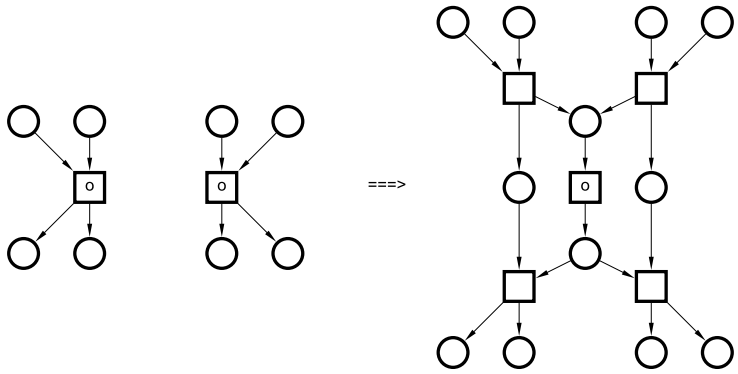


Fig. 3. Replacing multiple o-labelled transitions by a single one. Intuitively, the o-labelled transition is used as a sub-routine, with some silent transitions used to remove the respective input tokens before performing o, and to produce the respective output tokens after performing o.

The bound on the number of tokens follows from the fact that executing the added transitions does not alter the current marking. □

This result immediately implies that cOD is RP-hard (i.e., RP is polynomially reducible to cOD) for the corresponding class of PNs (safe/k-bounded/general).

2.2 The upper bound

OD can be checked by considering each output o in isolation, converting the other output labels to inputs: Indeed, OD holds iff each of these single-output LPNs is OD. Furthermore, multiple o-labelled transitions can be replaced by a single one using the transformation shown in Fig. 3. This transformation preserves the enabledness of o and thus the OD property. Hence, below we assume that the LPN has a single output o, and a single o-labelled transition t_o.

We follow the approach of [2] and compute the synchronous product of the LPN with itself (we thus get two copies of the net that evolve independently, except that any visible action in one copy can be only performed synchronously with the same action in the other copy). Then any violation of the OD is witnessed by the following trace of the product net: $(M_N, M_N)[\nu\rangle\rangle(M_1, M_2)$, with $M_1[t_o\rangle$ (i.e. t_o is immediately enabled by M_1) and $\neg M_2[o\rangle\rangle$ in the original LPN. Unfortunately, this property is difficult to decide for general PNs, since we must verify that M_2 does not enable o even via a very long sequence of silent transitions (on which we cannot bound the intermediate markings a priori). This is not a problem for safe and k-bounded Petri nets, as their markings are bounded and can be represented in polynomial space, but the decidability and complexity in case of general PNs was left open in [2].

We observe that the set En_o of markings of the LPN that enable o (perhaps, via a sequence of silent transitions), i.e.

$$En_o \stackrel{\text{df}}{=} \{M \in \mathbb{N}^P \mid M[o\rangle\rangle \text{ in the original LPN}\},$$

is upward closed, and so can be represented by the finite set of its minimal elements $E_1, E_2, \ldots, E_k \in \mathbb{N}^P$:

$$En_o = \bigcup_{i=1}^{k} \{M \in \mathbb{N}^P \mid M \geq E_i\}.$$

Furthermore, the natural numbers constituting vectors E_i are at most double-exponential in the size of the LPN. Indeed, consider the PN obtained from the LPN by removing all the non-silent transitions. Then these vectors are the minimal initial markings of this PN from which it is possible to cover the preset of t_o. Due to Rackoff's famous result [6], it is sufficient to consider firing sequences of double-exponential length, and they can consume only double-exponential number of tokens.

Here we need to consider the markings *not* enabling o (via silent transitions); hence we are interested in the complement of En_o, i.e. in the set

$$Dis_o \stackrel{\text{df}}{=} \overline{En_o} = \{M \in \mathbb{N}^P \mid \neg M[o\rangle\rangle \text{ in the original LPN}\}.$$

Hence Dis_o is a downward closed set, and it can be represented by the finite set of its maximal elements $D_1, D_2, \ldots, D_{k'}$ in the standard extension of $\mathbb{N}^P$ to the set $(\mathbb{N} \cup \{\omega\})^P$ of *generalised markings*; i.e.

$$Dis_o = \bigcup_{i=1}^{k'} \{M \in \mathbb{N}^P \mid M \leq D_i\}.$$

One can observe that the finite (i.e. non-ω) numbers in vectors D_i are smaller than the largest number occurring in vectors E_j. (Suppose some D_i contains $x \in \mathbb{N}$ that is greater than or equal to the largest number in vectors E_j. For every E_j we have $D_i \not\geq E_j$, i.e. some component of D_i is smaller than the corresponding component of E_j, which cannot be the component with x. But then D'_i arising from D_i by incrementing x also satisfies $D'_i \not\geq E_j$ for all E_j, which contradicts the maximality of D_i.)

Hence the non-ω elements of D_i are at most double-exponential in the size of the LPN, and so any D_i can be represented in exponential space using (e.g.) binary encoding of natural numbers, with a special code for ω.

We now define a non-deterministic algorithm that uses Reachability as an oracle and solves cOD. All operations except Reachability can be performed in exponential space; since Reachability is $\mathcal{EXPSPACE}$-hard, its complexity will dominate the overall complexity of the algorithm. The algorithm takes an LPN with single output o and a single o-labelled transition t_o and proceeds as follows.

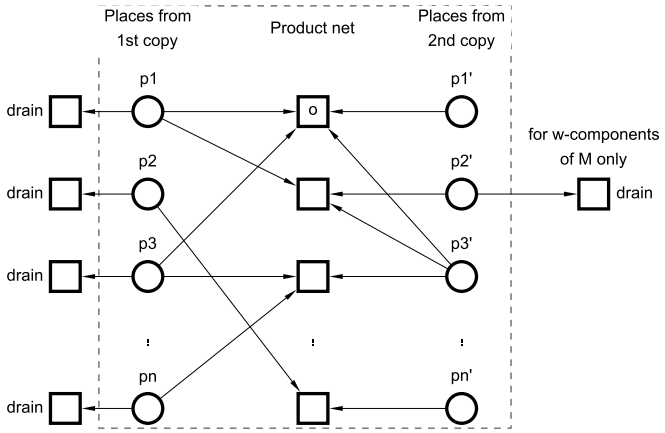


Fig. 4. Reduction from OD to RP.

- Non-deterministically generate a generalised marking M with either ω or at most double-exponential number of tokens per place (thus M can be stored in exponential space).
- Build a PN by removing all the non-silent transitions from the LPN, and set M as its initial marking while also removing the places corresponding to the ω-components of M. If the set of places corresponding to the preset of t_o (without the removed ones) is coverable in this PN then **reject**. (If the algorithm gets past this reject statement, then any finite marking $M' \leq M$ is in Dis_o. Moreover, in any finite $M' \in Dis_o$ we can possibly turn some components to ω and get a generalised M that satisfies the double-exponential restriction and is not rejected; this follows due to the bound on the size of maximal elements in $D_1, \ldots, D_{k'}$ derived above.)
- Build the instance of Reachability as follows (see Fig. 4):
  - Construct the product of the LPN with itself.
  - For each place in the first sub-net of the product, create a 'drain' transition consuming tokens from this place.
  - For each place in the second sub-net of the product that corresponds to an ω component of M, create a 'drain' transition consuming tokens from this place.
  - The marking to be reached puts a single token on each place corresponding to the preset of t_o in the first subnet, no tokens on the other places of the first subnet, and marking M on the places of the second subnet, with no tokens on the places corresponding to the ω components of M.
- If the constructed instance of Reachability is positive (as told by the oracle) then **accept** else **reject**.

Lemma 2. *The original LPN violates OD iff the above algorithm can accept the LPN.*

Proof. (1) Suppose the above algorithm has an accepting run; we fix one. Let M be the respective generalised marking that passed the test (of non-coverability of the preset of o). We consider an execution in the constructed PN that demonstrates the positive answer of Reachability; it yields the marking with tokens on the preset of t_o and no tokens elsewhere in the first subnet, and some tokens in the second subnet forming a finite marking $M' \leq M$. Removing all the 'drain' transitions from this execution still yields a valid execution of the PN, as these transitions can only remove tokens. This modified execution is an execution of the product net, still marks all the places in the preset of t_o in the first subnet, and in the second subnet yields some finite marking M'' that may have some extra tokens on places corresponding to the ω components of M; hence $M' \leq M'' \leq M$. The execution can thus be projected to two executions of the original LPN with the same visible trace, where one execution directly enables t_o and the other ends in a marking $M'' \in Dis_o$ (thus not enabling o even via a sequence of silent transitions); this constitutes a violation of OD.

(2) Suppose the LPN violates OD. Hence there are two executions with the same trace, one of them directly enabling t_o and the other ending in a finite marking M' not enabling o even via a sequence of silent transitions, i.e. $M' \in Dis_o$; thus $M' \leq D_i$ for some maximal element D_i of the extension of Dis_o. Let M be the marking that puts no tokens on places corresponding to the ω-entries of D_i, and coinciding with M' on other places. These two executions yield an execution of the product net leading to a marking enabling t_o in the first subnet and coinciding with M' in the second subnet of the product. By executing the 'drain' transitions as necessary, one can ensure that the marking of the first subnet has a single token on each of the places in the preset of t_o and no tokens elsewhere and the marking of the second subnet coincides with M — this marking corresponds to the one whose reachability is checked by the algorithm. Hence the algorithm can accept by first guessing M (that fits in exponential space due to the bound on the maximal non-ω elements of the extension of Dis_o) and then solving Reachability for the constructed reachable marking. □

A problem with the above construction is that the input to the Reachability sub-routine is large, as M might need exponential space (in the size of the original LPN) to be represented.

However, this problem can be solved by shifting the computation performed by the above algorithm into the constructed PN, thus polynomially reducing a cOD instance to a small instance of Reachability (in fact, to an instance of ZRP). A crucial ingredient is handled by Lipton's construction [4] (strengthened in [5]) enabling to simulate an exp-space-bounded automaton by a net of polynomial size. One can thus construct a polynomial-size PN (w.r.t. the size of the original LPN) which has the following behaviour (see Fig. 5):

– By a (polynomial-size) "Lipton module" it first non-deterministically generates a marking M where, moreover, each place p also gets its complementary place p' marked so that the sum of tokens in p and p' is $2^{2^{pol(n)}}$ for a suitable

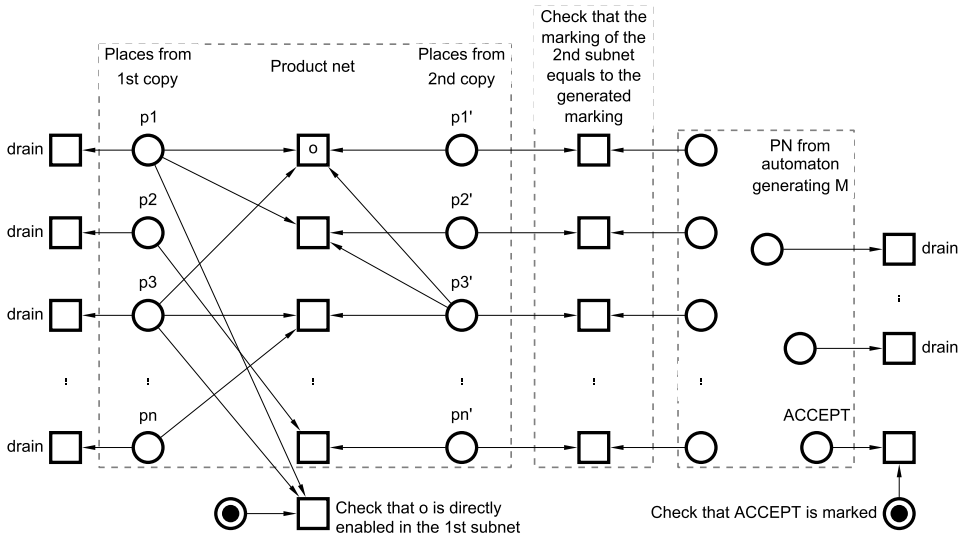


Fig. 5. Reduction from OD to ZRP.

polynomial *pol* (where n is the size of the original LPN). The polynomial *pol* is chosen so that the finite components in the maximal elements D_i of the extension of Dis_o are smaller than $2^{2^{pol(n)}}$; such polynomial *pol* follows from Rackoff's results [6]. We can view the value $2^{2^{pol(n)}}$ as ω. In fact, the module creates two copies of M, one copy (with complementary places) for the following test that $M \in Dis_o$, and the other copy for later comparing with the marking reached in the second part of the product net.

- Now another "Lipton module" checks that $M \in Dis_o$, i.e., the module has a possibility to get a token on a designated ACCEPT place precisely when $M \in Dis_o$. This module is again of polynomial size (it simulates checking if M can cover the preset of t_o within $2^{2^{pol(n)}}$ moves).
- (If there is a token on ACCEPT, then) an execution of the product net follows, reaching some marking (M_1, M_2).
- We want M_1 to enable o while M_2 to belong to Dis_o. The former condition can be handled by the drain transitions and asking that only one token is left in each place in the preset of t_o (in the final marking of the constructed Reachability instance). The other condition is established by comparing M_2 with (the stored copy of) M. This is achieved by special transitions consuming tokens from places of M_2 and corresponding places of M synchronously, asking that zero is reached for all of them. The only issue are "ω-places" in M, i.e. those having $2^{2^{pol(n)}}$ tokens. We first let such places (checked by a "Lipton module") to be adjusted anyhow by special transitions.

By adding several transitions and places and providing the possibility to drain any left over tokens from the "Lipton modules" we construct an instance of ZRP where the reachability of **0** marking guarantees that:

- the generation of M was successful, i.e. ACCEPT is marked;
- the marking reached in the second subnet equals to M (or to adjusted M which also belongs to Dis_o)
- the preset of t_o is marked.

Furthermore, guessing the output o for which OD is violated can also be implemented in the constructed PN in a straightforward way. Hence, we have a polynomial reduction from cOD to ZRP.

3 Conclusion

We have affirmatively answered the question of the decidability of Output-Determinacy in general Petri nets, and sketched the proof to determine its complexity: the complement of Output-Determinacy is polynomially interreducible with Reachability (or ZRP).

References

1. Hack, M.H.T.: Decidability Questions for Petri Nets. PhD thesis, MIT, 1975.
2. V. Khomenko, M. Schaefer and W. Vogler: *Output-Determinacy and Asynchronous Circuit Synthesis.* Special Issue on Best Papers from ACSD'07, IOS Press, *Fundamenta Informaticae*, 88(4) (2008) 541–579.
3. M. Kishinevsky, A. Kondratyev, A. Taubin, and V. Varshavsky. *Concurrent Hardware: The Theory and Practice of Self-Timed Design.* John Wiley & Sons Ltd., 1994.
4. R. Lipton. *The Reachability Problem Requires Exponential Space.* Technical Report 62, Yale University, 1976.
5. E.W. Mayr and A.R. Meyer. *The Complexity of the Word Problems for Commutative Semigroups and Polynomial Ideals. Advances in Mathematics*, 46:305–329, 1982.
6. C. Rackoff. *The Covering and Boundedness Problems for Vector Addition Systems. Theoretical Computer Science*, 6:223–231, 1978.
7. W. Vogler and B. Kangsah. Improved decomposition of signal transition graphs. *Fundamenta Informaticae*, 76:161–197, 2006.
8. W. Vogler and R. Wollowski. Decomposition in asynchronous circuit design. In J. Cortadella et al., editors, *Concurrency and Hardware Design*, Lect. Notes Comp. Sci. 2549, 152 – 190. Springer, 2002.
9. A. Yakovlev, M. Kishinevsky, A. Kondratyev, L. Lavagno, and M. Pietkiewicz-Koutny. On the models for asynchronous circuit behaviour with OR causality. *Formal Methods in System Design*, 9:189–233, 1996.

Optimal Simulation and Maximally Concurrent Evolution - My Early Papers With Maciej Koutny

Ryszard Janicki

Department of Computing and Software, McMaster University,
Hamilton, Ontario, L8S 4K1 Canada
janicki@mcmaster.ca

1 Beginnings

I have met Maciej somehow around 1979/80, when he took my undergraduate course 'Introduction to Switching and Automata Theory'. The course was based on then popular textbook by M. A. Harrison, with the same title, which was also known for having plenty of very difficult problems as exercises. In the middle of term, I gave one of these problems to the students as optional homework. The next class was almost a week later and only Maciej said that he had a solution. His solution was very clever, much more elegant and shorter than the one I had, and he presented it in such nonchalant way, as he did in less that twenty minutes just before the class. I was very impressed as it took me some substantial time to produce my inferior and longer solution. Only about fifteen years later, when we were friends and research collaborators, he admitted that during this almost a week between classes, practically the only thing he thought of, was to try to solve this problem. But his presentation suggested otherwise.

Both him and, then his girlfriend, Marta Pietkiewicz, become interested in my research, Marta did Master Thesis under my supervision, Maciej did his Master Thesis under supervision of Antoni Mazurkiewicz and later PhD under my supervision.

Nevertheless I do not have any common paper with Maciej related to his PhD thesis. In fact I have no common paper related to their theses with any of my former Polish PhD and Masters students. I also have no common paper related to my PhD thesis with Antoni Mazurkiewicz, my PhD supervisor. In seventies and at least first half of eighties last century having papers with students, for most mathematicians and theoretical computer scientists, were considered a breaking of some well established code of honour. The code of honour of Polish School of Mathematics was still very observed, and most of people from theoretical computer science graduated mathematics. According to this code of honour, having papers with students (with rare well justified exceptions) was considered dishonourable. Sounds weird today but this was true then.

It was only in Canada, where to my great surprise, I have learned that having common papers with students is not only a virtue but almost a necessity for a successful grant application or promotion process. Since swimming against the current is seldom successful in long term, right now I have plenty papers with my graduate students, but our first common paper with Maciej was written few years after his PhD, and it dealt with the problem of maximal concurrency [13].

2 Maximal Concurrency and Optimal Simulations/Executions

Since 1986 I have published with Maciej a few dozens of papers on different subjects, the most known are our results on *relational structure semantics of concurrent systems*, which started with [7] and the most representative papers on this subject are still probably [11] and [12]. Nevertheless our first period of collaboration, roughly from 1985 to 1991 was devoted to maximally concurrent and optimal executions (simulations) of concurrent systems under step sequence observational semantics paradigm [4–6, 8, 9, 13], and additional assumption that concurrent behaviours are entirely specified by partial orders.

Among various semantics of executions in non-sequential systems, we can distinguish three widely accepted approaches. The first one, standard in Petri net based models [17], COSY paths expressions [10, 14], Asynchronous Automata [21] and many others [11], may intuitively be expressed as: '*execute as possible*'. This encompasses the whole range of possibly concurrent evolutions; from the sequential ones to the maximally concurrent ones through all the intermediate case, any possible execution is allowed. For example, assuming that observational semantics is defined in terms of step sequences, if the events a,b,c are independent, the following step sequences are considered as equivalent: $\{a,b,c\}$, $\{a,b\}\{c\}$, $\{a,c\}\{b\}$, $\{b,c\}\{a\}$, $\{a\}\{b,c\}$, $\{b\}\{a,c\}$, $\{c\}\{a,b\}$, $\{a\}\{b\}\{c\}$, $\{b\}\{a\}\{c\}$, $\{a\}\{c\}\{b\}$, $\{c\}\{a\}\{b\}$, $\{b\}\{c\}\{a\}$ and $\{c\}\{b\}\{a\}$.

The second one can be called '*execute as possible, but in sequence*', is called *interleaving semantics* and is widely used in all kinds of Process Algebras [1]. In this case for three independent events a,b,c, we have sequences: *abc*, *bac*, *acb*, *cab*, *bca*, *cba*.

The third one is usually expressed as '*execute as much as possible in parallel*' or '*execute as quick as possible*'. In this case we require that processes should not be lazy, and at each step of computation, the set of events executed must be a maximal non-conflict set. This greedy semantics is often used for some kind of Timed Petri Nets [20], some temporal logic applications [3], it is often easier to implement and has many nice algorithmic properties [18]. For our three independent events a,b,c, it will be represented by just one step $\{a,b,c\}$.

Our first common paper [13] (with Peter Lauer and Raymond Devillers as the remaining co-authors) was a successful attempt to answer the question: "*When is the semantics 'execute as much as possible in parallel' equivalent to the semantics 'execute as possible'?*". Various necessary and sufficient conditions for this property were provided and proved. We used COSY Path Expressions (first proposed by Peter Lauer and R. H. Campbell in [14], fully presented in [10]) to represent concurrent systems, and Vector Firing Sequences (proposed by Mike Shields in [19]) to represent concurrent behaviours, however the results can easily be translated into Elementary Petri Nets [16] and Mazurkiewicz Traces [15] extended to step sequences (cf. [4, 8]).

Employing maximal concurrency is an attractive idea, both conceptually and from the point of view of implementation. Unfortunately, as pointed out in [13], there are cases in which the greedy maximally concurrent execution is not sufficiently expressive. To show this case we take the net from Figure 1. The maximally concurrent execution can find only one deadlocked marking of the net, by following the step sequence $\{a,b\}\{d\}$. The other deadlocked marking, which might be reached by following the step sequence $\{b\}\{c\}$, is left undetected.

The process of verification of dynamic properties of non-sequential systems often involves some kind of reasoning about the complete state-space of the system, e.g. proving deadlock-freeness requires showing that it is not possible to reach a state in which no transition is enabled. Reasoning about the complete state-space of non-sequential systems has one very serious drawback, which is a combinatorial explosion of state-space. Even a simple concurrent system can generate many hundreds or thousands of states. To cope with this problem a number of sophisticated techniques have been developed, for example induction, reduced stat-space analysis, etc.

In [4] and [5] a possibility of defining a fully expressive reachability relation on the system's executions which would be a 'small' subset of the complete reachability relation. Such reduced reachability relation was called the *optimal simulation* (it was later renamed to *optimal execution* in [10]). Conceptually, the optimal simulation can be considered as a generalized version of maximal concurrency and, in some, but not so infrequent, cases it is just the maximal concurrency [4, 5]. The concept of *canonical step sequence*, a simple extension of canonical sequence for Mazurkiewicz traces [2, 13] plays crucial role. Figure 1 shows both the full reachability graph of a simple elementary net [16] and the reachability graph of the optimal simulation. The latter is smaller, but because in this case only two transitions a and b can be fired concurrently, the difference in size is not significant. However, in the case of net in Figure 2, the reachability graph of the optimal simulation is isomorphic to that in Figure 1, while (as one may easily check) the full reachability graph would hardly fit on a single page.

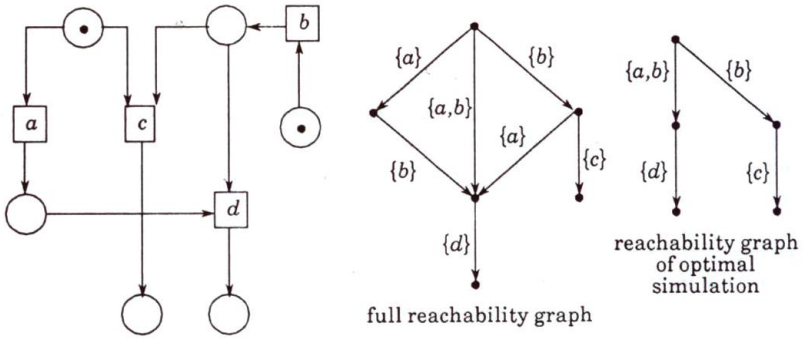


Fig. 1: Full reachability graph versus reachability graph of optimal simulation. Incompleteness of maximally concurrent semantics - the right deadlocked marking, after the step sequence $\{b\}\{c\}$ is not detected.

Optimal simulation enables reasoning about a number of dynamic properties of concurrent systems, and at the same time requires some minimal computational effort. In [4, 5], the optimal simulation had been defined in a very general step sequence setting which made it applicable to different models of concurrency, such as Petri nets, various process algebras or automata-based models.

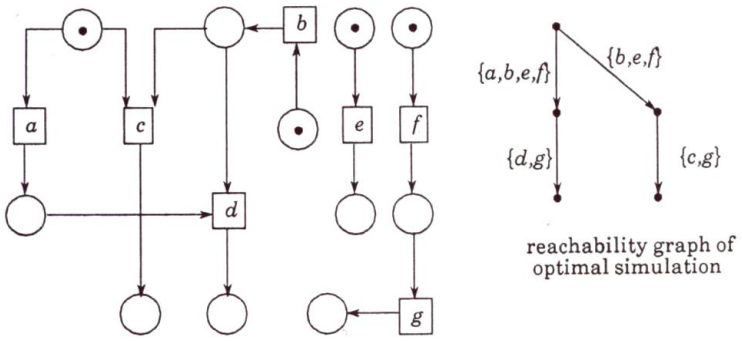


Fig. 2: A net with small reachability graph of optimal simulation and huge (not shown) full reachability graph.

The reachability graphs of finite state systema ca be regarded as *finite representations of reachability relations*. Since the optimal simulation provides the same information about relevant dynamic properties of the system as the full reachability relation [4, 5], the reachability graph of optimal simulation (i.e. it finite representation) and the full reachability graph may be considered equivalent. The optimal simulation is always a subset of full reachability, but of course it does not mean that the reachability graph of optimal simulation is always much smaller than the full reachability graph. However, in the case of concurrent systems exhibiting high degree of concurrency (i.e. those with many sequential components), the reachability graph of optimal simulation is much smaller that the full reachability graph. Thus it is advantageous to use optimal simulation as a tool to reduce the size of reachability graphs of concurrent systems.

Unfortunately, as opposed to the full reachability graph, in the general case it is not clear how to generate the reachability graph for optimal simulation. In some sense this is a negative side-effect of the generality of optimal simulation.

In [6] and [8], we have shown how the reachability graph for optimal simulation can be constructed for Elementary Petri Nets that can be decomposed onto finite state machines, and in [9] for concurrent systems that can be represented by asynchronous automata of [21].

The approach presented in [4–6, 8, 9, 13] was based on the assumption that concurrent behaviours (histories) are modeled by casual partial orders. We represented those partial orders by certain equivalence classes of step sequences, just generalizing the notion of Mazurkiewicz traces [15]. The relational structures and 'not later than' phenomenon, introduced in [7] and first time discussed in detail in [11, 12] are not considered in these papers.

The major methodological difference between our approach based on optimal simulation to minimize reachability graphs and virtually almost all others is that we do not try to minimize (or even deal with) the full reachability graph. All what we were trying to do was to buld a reachability graph which represents optimal simulation relation (a subset of full reachability). We then made a claim, based on the general properties of

optimal simulation, that in majority of cases, such a graph is much smaller than the original full reachability graph.

I believe the concept of optimal simulation (or execution) and its usefulness was and still is a little bit unappreciated. The main reason could be that we never built any software that supported optimal simulation and never applied the ideas of optimal simulation to real concurrent systems as for example security protocols. We were than both relatively young recherches and neither of us had sufficient knowledge, experience and connections for successful huge grant application, that was needed for producing good experimental software.

I also believe that the problem of using optimal simulation/execution is still not closed and still has many potential applications. It may be worth to go back to it in some future time.

References

1. J. A. Begstra et al (eds.), *The Handbook of Process Algebras*, Elsevier Science 2000.
2. Cartier, P., Foata, D.: Problemes combinatoires de communication st rearrangements, *Lecture Notes in Mathematics* 85, Springer 1969.
3. Enjalbert, P., Michel, M.: Many-sorted temporal logic for multiprocesses systems, *Lecture Notes in Computer Science* 176, Springer 1984, 273-281.
4. Janicki, R., Koutny, M.: On Equivalent Execution Semantics of Concurrent Systems, *Lecture Notes in Computer Science* 266, Springer 1987, 89–103.
5. Janicki, R., Koutny, M.: Towards a Theory of Simulation for Verification of Concurrent Systems, Proc. of Parallel Architectures and Languages Europe '89, *Lecture Notes in Computer Science* 366, Springer 1989, 73-88.
6. Janicki, R., Koutny, M.: Net Implementation of Optimal Simulation, Proc. of Application and Theory of Petri Nets '90, Paris, France, June 1990.
7. Janicki, R., Koutny, M.: Invariants and Paradigms of Concurrency Theory. *Lecture Notes in Computer Science* 506, Springer 1991, 59–74.
8. Janicki, R., Koutny, M.: Using Optimal Simulations to Reduce Readability Graphs, *Lecture Notes in Computer Science* 531, Springer 1991, 166-175
9. Janicki, R., Koutny, M.: On Some Implementation of Optimal Simulation, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 3 (1991), 231-250
10. Janicki, R., Lauer, P. E.: *Specification and Analysis of Concurrent Systems*. The COSY Approach, Springer 1992.
11. Janicki, R., Koutny, M.: Structure of Concurrency. *Theoretical Computer Science* 112, 5–52 (1993)
12. Janicki, R., Koutny, M.: Semantics of Inhibitor Nets. *Information and Computation* 123(1), 1–16 (1995)
13. Janicki, R., Lauer, P. E., Koutny, M., and Devillers, R.: Concurrent and Maximally Concurrent Evolution of Non-Sequential Systems, *Theoretical Computer Science*, 43, 213–238 (1986).
14. Lauer, P. E., Campbell, R. H.: Formal Semantics of a Class of High level Primitives for Coordinating Concurrent Processes, *Acta Informatica* 5, 297–332 (1975)
15. Mazurkiewicz, A.: Concurrent Program Schemes and Their Interpretation. TR DAIMI PB-78, Comp. Science Depart., Aarhus University (1977)
16. Nielsen, M., Rozenberg, G., Thiagarajan, P. S.: Behavioural Notions for Elementary Net Systems. Distributed Computing 4, 45–57 (1990)

17. Reisig, W.: *Petri Nets*, Springer 1985.
18. Salwicki, A., Müldner, T., On Algorithmic Properties of Concurrent Programs, *Lecture Notes in Computer Science* 125, Springer 1981, 169–197.
19. Shields, M. W.: Adequate Paths Expressions, *Lecture Notes in Computer Science* 70, Springer 1979, 249–265
20. Wang, J.: *Timed Petri Nets*, Kluwer Academic 1998
21. Zielonka, W.: Notes on Finite Asynchronous Automata, *RAIRO Inform. Théor. Appl.* 21:99–135, 1987.

A Tribute to Maciej Koutny's Exceptional Scientific Leadership

Kurt Jensen

[1] University of Aarhus, Denmark
kjensen@cs.au.dk

Maciej Koutny has been – and still is – a very active and very influential member of the International Petri Net Society. He has participated in most of the nearly forty annual Petri Net Conferences. Nine times, from 1996 to 2010, he was member of the programme committees – one as Chair and two as Workshop & Tutorial Chair. Furthermore, Maciej also organized the conference on three or four occasions.

Maciej served as a member of the International Petri net Steering Committee in nearly twenty years until he became the Chair in 2013 – at which time he also became the Editor-in-Chief of the LNCS Transactions on Petri Nets and Other Models of Concurrency. In 2010 Maciej was also one of the scientific directors of the Advanced Course on Petri Nets and for many years, he has been a regular lecturer at the Petri Advanced Courses and the Petri Net Tutorials.

In addition to this, Maciej has been active in several related scientific communities – in particular the one around the Application of Concurrency to System Design conferences.

Hence, Maciej is without any doubt one of the handful of persons who have been most influential with respect to securing the success of the Petri Net community. We should all be extremely grateful to Maciej for the enormous work he has carried out for the International Petri Net community.

Maciej is known and highly respected as a very knowledgeable and solid scientist. Over a span of thirty years, he has published numerous papers of which nearly one hundred have ten or more citations. This is a quite impressive number for a person working in his research area.

Maciej is always easy to communicate with and he is friendly towards everyone – old friends as well as newcomers in the research society. Moreover, he is one of the most reliable persons I have worked with – always on time and with excellent quality.

I have had the privilege to work with Maciej over a period of more than twenty years – during which we became close friends. This has always been a pleasure, and I sincerely hope that Maciej will continue his successful leadership of the International Petri Net community for many years to come.

Dear Maciej. Thank you very much for the time we have known each other and worked together.

A tale of high-level features in the Petri Box Calculus

Hanna Klaudel[1], Franck Pommereau[1], and Elisabeth Pelz[2]

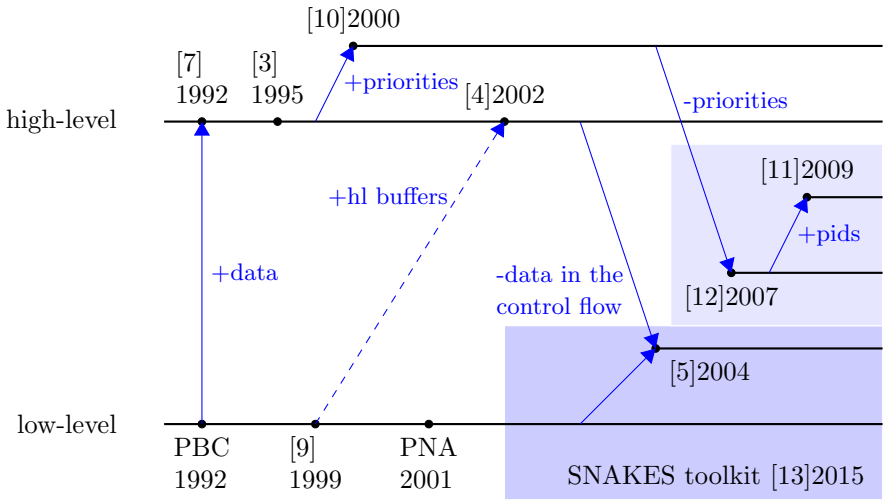
[1] University of Paris-Saclay, University of Évry
[2] University of Paris-East-Créteil

Abstract. Once upon a time the Petri Box Calculus was invented. Let us tell you our tale of the introduction of high-level features in this remarkable formalism that gave us a research topic for our whole research career.

The Petri Box Calculus [1] is a formalism that combines Petri nets and process algebras, which have been an idea that can be traced back to the early 70's. A team started to build on this topic with two EU-funded research projects DEMON and CALIBAN. It resulted in a formal framework (see also [2]) in which Petri nets can be composed like terms in a process algebras. This approach yields standard Petri nets that can be analysed automatically through the numerous methods and tools.

Our tale begins with the publication of the Petri Box Calculus (PBC) in 1992, it tells our story of introducing high-level features in the PBC and the steps it took to progressively improve these features until we were fully pleased with them. This is a personal story, and we apology for the many works we have left apart. Consequently, this tale has no end yet and much remains to be written...

The whole story takes place on this timeline:



The first step was a giant's step, it was made by somebody that's about 5ft tall, and it reached probably too far away as we will see in the sequel. In 1992, closely after the publication of PBC, A-nets [7] were introduced and defined a variant of PBC in which places could contain arbitrary values from arbitrary algebraic data types. This was later ported to the more practical domain of coloured Petri nets by introducing M-nets in 1995 [3]. M-nets were further extended in 2000 by introducing preemption, allowing to suspend/resume/abort arbitrary sub-nets [10]. This was obtained through the introduction of priorities between transitions, which was conceptually nice but led very far away from any hope of a reasonable or efficient implementation of the framework...

In the mean time, asynchronous communication through buffer places were added, first to PBC in 1999 [9], then to M-nets in 2002[4]. While this is not a feature that is directly related to high or low-level aspects, it paved the way to a compromise between the hairy generality of M-nets and the data-bareness of PBC. Indeed, it draws a clear distinction between buffer places that are resources and control-flow places that can remain low-level. This idea led in 2004 [5] to a formalism that can be seen as PBC extended with high-level buffer places, and preserving all the original PBC operations.

At this point, we had a practical formalism that was at the same time expressive enough to model complex systems, and disciplined enough to allow its implementation. This coincides with the starting of the development of the SNAKES toolkit [13]. Still, the preemption features were missing. They could be introduced again, without transitions priorities, in 2007 [12] by forbidding nested parallelism and by considering two-colours control-flow where black tokens • implemented the standard PBC control-flow and white tokens ∘ implemented the preemption control-flow.

The idea of pushing parallelism to the top-level exists in CSP [6] and is similar to the multi-threaded programming paradigm. But, we still lacked the ability to create new processes dynamically, which threads have and CSP does not. This feature was introduced in 2009 [11] by porting into the PBC family the ideas from [8]. Each data or control-flow token is now tagged with a process identifier. This allows the concurrent execution of many instances of a single Petri net with no undesired interference (communication between processes is possible). This solution is remarkably versatile as it proposes unique features: i) process identifiers are rich data structures that allow to trace every process' history; ii) state-space is reduced against process identifiers (more accurate that just a symmetry reduction); and iii) usual notions of threads and processes are unified. For the last feature, that allows the abortion of a working thread, there is no programming language able to implement it at the level of threads.

Research is still active in the domain, for instance, we are presently working on a new PBC variant with operations that are very different from the original control-flow ones, with the aim of developing directly with Petri nets the acting system of autonomous robots.

Looking forward, we feel very lucky to have chosen to embark the PBC ship and get inspiration and research topics for the rest of our life.

References

1. Best, E., Devillers, R., Hall, J.G.: The box calculus: a new causal algebra with multi-label communication. In: Advances in Petri Nets 1992, The DEMON Project. Springer (1992)
2. Best, E., Devillers, R., Koutny, M.: Petri net algebra. Springer (2001)
3. Best, E., Fraczak, W., Hopkins, R.P., Klaudel, H., Pelz, E.: M-nets: An algebra of high-level Petri nets, with an application to the semantics of concurrent programming languages. Acta Informatica **35**(10) (1998)
4. Bui Thanh, C., Klaudel, H., Pommereau, F.: Box calculus with coloured buffers. Tech. Rep. 16, LACL, University Paris East (2002)
5. Bui Thanh, C., Klaudel, H., Pommereau, F.: Box calculus with high-level buffers. In: DADS'04. SCS (2004)
6. Hoare, C.: Communicating Sequential Processes. Commun. ACM **21**(8) (1978)
7. Klaudel, H.: Traitement des données dans l'Algèbre des Petri Boxes. Master thesis, University Paris XI, Orsay, France (1992)
8. Klaudel, H., Koutny, M., Pelz, E., Pommereau, F.: Towards efficient verification of systems with dynamic process creation. In: ICTAC'08. LNCS, vol. 5160. Springer (2008)
9. Klaudel, H., Pommereau, F.: Asynchronous links in the PBC and M-nets. In: ASIAN'99. LNCS, vol. 1742. Springer (1999)
10. Klaudel, H., Pommereau, F.: A concurrent and compositional Petri net semantics of preemption. In: IFM'00. LNCS, vol. 1945. Springer (2000)
11. Pommerau, F.: Algebras of coloured Petri nets. Habilitation thesis, UPEC (2009)
12. Pommereau, F.: Versatile boxes: a multi-purpose algebra of high-level Petri nets. In: DADS'07. SCS/ACM (2004)
13. Pommereau, F.: SNAKES: a flexible high-level Petri nets library. In: Proc. of PETRI NETS'15. LNCS, vol. 9115. Springer (2015)

Partially Ordering Koutny's Traces

Jetty Kleijn and Paul Smit

Leiden, The Netherlands

Abstract. Motivated by the remarkable event of the 60th birthday of Maciej Koutny, this illustrated paper considers various traces of synchronising events involving Maciej. It does not claim to be complete, but provides a partially ordered survey of important features of a longtime collaboration between Newcastle and Leiden.
Keywords Research, Collaboration, Friendship, Travel, Synthesis

1 Introduction

In this survey paper, we describe synchronisations between specific processes executed by local agents in Newcastle and Leiden. The underlying mechanism is based on voluntary and eager cooperation. In almost three decades, this has led to an intricate combination of theory and applications based on the sharing of events and interests of all sorts.

Now is a good time to trace and order what has been achieved and what can still be done. In the past 30 years, founding father Maciej Koutny managed to double his age. Indeed, as Victor Khomenko put it in his invitation mail, a remarkable event. Apparently also a festive event in which many agents take part. Here we will identify various types of events in which Maciej (often with his spouse Marta) interacted with the authors (also a couple). Depending on the type and the relations between these types, some would be ordered, while others took place in parallel.

The paper is organised as follows. After explaining how it all began, we first focus in Section 3 on main events involving both Newcastle and Leiden. We identify different types of events, including 'work', 'friends', and 'travel'. In Section 4, focus is on their enabling conditions. In the concluding section, we summarise and briefly consider how to proceed from here.

It must be noted here that we do not give explicit references, but rather provide photos to support the claims made. More information can be obtained by approaching Maciej directly and verifying with him what is written below.

2 Basics or How It All Began

As already mentioned in the Introduction, the cooperation between Newcastle and Leiden has existed for almost 30 years. It was initiated in 1989 at the first meeting of DEMON (DEsign Methods based On Nets) an ESPRIT Basic Research Action, a European project coordinated by Eike Best. This meeting

took place in Bonn as a satellite event of the 10th Petri Net Conference. It was an excellent setting to get acquainted with and connect to many people of different places with various interests in a common field of research: concurrent systems and in particular those based on the principles of Petri Net theory.

Hildesheim 1993

There were several people from Leiden, among whom Grzegorz Rozenberg and Jetty; and also a team from Newcastle with Maciej. DEMON had different working groups and Newcastle and Leiden collaborated among others in one concerned with Abstract Models. One year later, in 1990, the Petri Net Conference was held in Paris, again with a meeting of DEMON. Here Maciej and Paul met for the first time. Still, the oldest picture of our collaboration we could find, is only from 1993. It was taken at CONCUR in Hildesheim.

By 1997, Maciej, together with Eike and Raymond Devillers, was about to finish their book Petri Net Algebra, one of the outcomes of DEMON and its successor CALIBAN, CAusal calcuLI BAsed On Nets. At the meeting of the Program Committee of that year's Petri Net Conference in Toulouse, we compared our notes on local traces and comtraces and got intrigued by the question how to relate the features of multiple tokens and inhibitor arcs in a causality semantics. (A feature from the past: real-life PC meetings with real-time discussions, and opportunities for interaction.) This led to the submission of a proposal later that year for a project on Local Comtraces. The application was successful, we got the grant and the first synchronisation across the Channel was a fact.

Maciej could not attend the Petri Net Conference in Toulouse, because his daughter had to study for her GCSE exams while Marta had a paper to present at the conference. Thus Jetty and Marta met for the first time in Toulouse.

Early 1998, Jetty went to England as the start of our first Newcastle-Leiden project. She lived close to the university, but there was ample opportunity to visit the Koutnys at their home just out of Newcastle and to go together on weekend excursions in the countryside. This stay of several weeks laid the basis for numerous research papers, quite a few also with Marta, and many excursions and travels with the four of us.

3 Main Events

We are now ready to identify the main events underlying the cooperation between Newcastle and Leiden. In the context of this section, an event is an occurrence of an action. Each action may be executed a number of times, but its occurrences are always linearly ordered. Two events representing different actions (e.g., when

they are labeled as 'work' and 'travel', respectively) are not necessarily ordered and may even take place simultaneously.

Work

The events here categorised as 'work' are occurrences of the action 'research' (an abbreviation of 'doing research together'). Obviously, we could refine this action and consider processes consisting of 'thinking', 'meeting', 'discussing', 'writing' etc, but that is not necessary for the purpose of this paper. Note that instead of 'work', also the word 'fun' would have been appropriate to capture doing research together. As will become clear further on, this would however have led to an overloading of 'fun'.

Over the years Maciej and Jetty have co-authored approximately 50 papers and the count is still on. The first journal paper took a lot of effort. Before we could start on the local comtraces, we first had to investigate and systematise what it actually was that we were after and what needed to be defined and proven. Using the general framework then established, it was possible to investigate many different classes of nets and semantical models. After some time, our research area widened and we found additional co-authors, like Marta, Grzegorz, Łukasz Mikulski, and Ryszard Janicki. So, it happened that Jetty was recognised as an honorary Pole; and it took not long before also Paul was affected, even to the extent that he has started to appreciate beetroot.

In addition to 'research', there is also a distinguished action that belongs to 'work' and is referred to as 'The Book'. This is a collaborative effort between Newcastle (Maciej), Leiden (Jetty), Toruń (Łukasz), and Hamilton (Ryszard). It grew out of our semantics/trace research and it is still growing ... So, obviously, a lot of research went into the book which also led to new papers, but an important difference between 'research' and 'The Book' is that the latter is a one-time-only event.

Signing of the contract for The Book

Moving

A perhaps surprising action that took place on both sides of the Channel was 'moving'. There was no perfect synchronisation here, but almost. In 2003 Maciej and Marta moved to a house in Newcastle proper, perhaps inspired by our settling in the city centre of Leiden the year before. Moreover, both houses were old and dilapidated (making 'moving' quite different from 'fun') and for something like a year we mostly discussed house renovation. In 2003, when we took our children for an English holiday we also visited Maciej and Marta in the remains of their house. It then seemed that 'moving' was an action that should not be

repeated, but times change and it may become enabled again for both couples when the preconditions are favourable.

Friends

'Friends' stands for actions that amount to sharing the important resources of family, friends, and colleagues (note that, also here, 'fun' would have been applicable). Sharing of such resources does not lead to conflicts, but rather to a perfect tuning of a common context.

Maciej at 50

As may be concluded already from the above, over time we have been introduced to many members of each other's families and friends (some of whom were also colleagues, like Alex Yakovlev and his wife Masha). Family (children, parents, siblings) and friends from all backgrounds could be encountered on different occasions, at their homes, for dinners, during holidays, and festivities (like for our Silver Wedding). A very special occasion was the celebration of Maciej's 50th birthday in Poland[1] for which he invited old and new friends. We were delighted to be able to attend and meet so many people important to Maciej. Also present were Hanna and Witek Klaudel whom we did not really know personally at the time.

Travel

Working intensively together from different countries implies that one has to travel every now and then. And indeed there have been many working visits from Maciej and Jetty to Leiden and Newcastle. Moreover, Jetty is a fellow in Newcastle and Maciej has been a Pascal professor in Leiden. In addition there are the frequent conference visits (and trips to Bilthoven).

If that were all, we could have included this kind of events under 'work'. What makes 'travel' so special however (and even more 'fun'), are the additional journeys of an exploratory character, often to far away places. We have been traveling like this with Maciej and Marta almost every year since 2008. In that year the Petri Net Conference was in Xi'an. We had never been to China, but Maciej and Marta had. They were so enthusiastic that we decided to stay a bit and try to get an impression. We formed a

Maciej in Leiden with Grzegorz

[1] How come he now suddenly turns 60? It seems like yesterday ..

group with them and Elisabeth Pelz, who also had some China experience. We decided on an itinerary and means of transportation and had a wonderful time altogether. The next year, there was an opportunity to visit Iran and with the four of us, we travelled from place to place and saw amazing things. In 2011, we had a round trip through the West of Turkey, where we also visited the well-known mathematical village of Sirince.

Near Sirince, the mathematical village

In Khiva with Al-Khwarizmi

Traveling to exotic places became a tradition. In 2010 Hanna and Witek joined the club and we travelled from Shanghai to explore yet another part of China. Going home became a bit problematic due to the eruption of the Icelandic vulcano, but in the end we had quite a jolly flight to Europe. With this group we have been again to China, to Japan and to Uzbekistan where we visited Khiva, the birthplace of Al-Khwarizmi. In addition we made smaller trips in the neighbourhood (France, UK, Italy, the Netherlands). So, 'travel' has become a crucial ingredient of the ongoing interactions between Newcastle and Leiden. It operates like a clockwork, we each have our role with Maciej being the moneyman aka the tax collector.

4 Enabling Conditions

As the reader is probably aware, actions — in order to occur — have to be triggered or at least become enabled (meaning that their preconditions have to be fulfilled). This principle is the basis for every causality semantics and we will apply it also here. In this section we describe some necessary preconditions for the actions categorised above. This should contribute to our understanding of the causalities in the complicated interactions between the different agents.

Triggering events
For the occurrence of the actions studied here, it is of paramount importance that someone takes the initiative. Often it is Maciej who initiates shared actions like 'research' and 'The Book'. He is generous with his ideas and works hard, often with many different people at the same time. He seems to have an infinite amount of energy and to never sleep. His mails can come in any day of the week, at any time, and from any place.

Mailing from Yazd

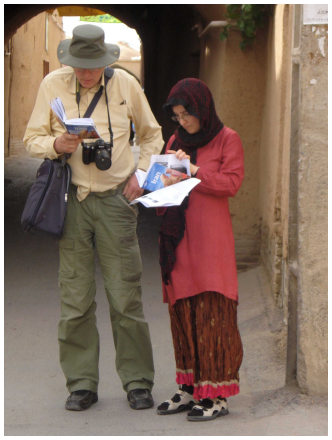
Chair in Hamburg

Maciej gets things done, fast and in an efficient way. He is the chairman of the steering committee for the Petri Net Conferences and the director of research of the School of Computing at Newcastle University. Nevertheless, he always tries to avoid bureaucracy as much as possible and to focus on what is really important. Anybody who has ever seen his office, remembers jealously the emptiness of his desk.

To further perfect his style, Maciej tries to learn the trade from great leaders. He studies their histories looking for crumbles of wisdom. Maciej is however NOT the initiator of the 'travel' events that really matter. He is the moneyman, capable of keeping track of even the most complicated transactions (between euros, pounds, and yuans, yens, rials, or sums), but it is Marta who is the driving force together with Paul. They are the true leaders (with some good, relaxing influence exercised by Hanna).

Mao

Tammerlane



Paul and Marta

Food

Another important aspect of events are the resources they need to consume to be able to occur. This literally holds for all actions in which Maciej is involved. A stomach filled with good food is an absolute prerequisite for him to get anything done. Maciej has a love-hate relationship with food. As Marta pointed out: Maciej smiles at his food. Every now and then however, there has been a bit too much of the good life and Maciej being Maciej has a drastic solution: the cabbage diet. For about a week he lives on cabbage, smiles less, but paves the way for more pleasant encounters.

Maciej and Friend

A dining Philosopher

Maciej is a good cook, a wonderful host and knows how to get splendid food in the most remote places. All this greatly contributes to the synchronisation required for 'work', 'travel', and 'friends'. In general, Maciej loves the good things of life and likes to share them sometimes while singing great Polish songs.

5 Conclusion

In this paper we have followed some of the traces of our longtime professional and personal friendship with Maciej Koutny. It is impossible to be complete and to capture every aspect and describe every relevant event in this relatively short survey. Therefore we could present no more than a partial order of important features to fulfil our aim of giving the reader at least an impression of this striking synthesis of work, friendship and travel. Hopefully, it has become clear how Maciej with Marta on his side, has made great contributions to the quality of our lives, that we cherish the memories and that we hope to create many more in the years to come.

Gratulacje, Drogi Przyjacielu, i Sto Lat!

An important question now is 'what can be done still once the milestone of 60 years has been passed?' From our own experience, we can tell that being < 60 or ≥ 60 really does not make all that much difference. We strongly advise to go on with all fun-raising activities like research, travel, friends, and family, with having grandchildren as a great new highlight.

Acknowledgment

We are grateful to Victor Khomenko and colleagues, students and friends of Maciej, for the invitation to contribute to this Festschrift. Victor was quite flexible about the form of the contributions and so after ample consideration we eventually decided that we would like to write something together on the occasion of the diamond birthday of our colleague, fellow traveler and friend of many years. Moreover, unlike Maciej and Marta, we never coauthored a paper yet.

References

Pictures from the collection of Paul Smit, Leiden, 2018.

A Brief Story of The Partnership

Łukasz Mikulski

Faculty of Mathematics and Computer Science,
Nicolaus Copernicus University
Toruń, Chopina 12/18, Poland
lukasz.mikulski@mat.umk.pl

"Remember, remember the fifth of November". It was November 5th, when I have exchanged with Maciej last mails before my first research visit to Newcastle University (several months earlier I was attending Petri Nets conference in Newcastle). Two days later I was already in Northumbria and met Maciej in Claremont Tower. After only a few meetings we establish the common language and get to know each other's interests. The outcome was almost immediate and during three weeks of my first visit in Newcastle we obtained advanced draft of our first common paper – "Hasse diagrams of combined traces" – together with several ideas and plans for future. The paper was ready in January, submitted and accepted for ACSD'12.

The submission process overlapped with our second research meeting, when Maciej came to Toruń as Visiting Professor. We had a lot of inspiring conversations, very intensive weeks of seminars when Maciej and whole FoLCo group (including Kamila Barylska, Marcin Piątkowski, Edward Ochmański† and me) presented their recent results. In the meantime, I was working on my second paper related to combined traces. Thanks to priceless comments done by Maciej, the paper "Algebraic Structure of Combined Traces" was good enough to be accepted for CONCUR'12. For me this was something incredible – new research topic, new friend, and two papers accepted for quite recognizable conferences in less than four months. But the magic only begins...

After the conference period (namely summer) I got back to Newcastle. This time for much longer, as I obtained a half-year postdoc scholarship and I became officially a visiting researcher in Newcastle University. The winter was fantastic. I started with the review of very early draft of the book (also known as THE BOOK) that Maciej has been writing with Jetty Kleijn and Ryszard Janicki. It turned out that my doubts related to the generalisation of combined traces were reasonable. The redefinition of this model became one of the main challenges for the postdoc. And we succeeded! The definition of generalized traces was simplified by introducing "sequentialization" relation and founding on their base traces equivalence. The equations present in the definition became simple and elegant. Everything indicated that we got it!

Moreover, in order to obtain a generalization, the set of axioms sufficient for structures underlying combined traces was firstly extended by "indivisible steps synchronization". That directly solved the counterexamples related to my earlier doubts. It remained to check whether the obtained model is indeed sound.

The attempt to prove the suitable version of Szpilrajn theorem (which narrowly bonds sets of equivalent observations and the structure of their invariants) revealed one more missing axiom – due to cross structure of the property and our roots related to Toruń called by us "teutonic axiom". And we hit the spot once more! When everything is defined properly, all the elegant properties and their proofs are only the matter of time. As a result, we solved the problem that was remained open in an old paper written by Maciej and Ryszard – "Structure of concurrency". Moreover, I have joined the editorial board of THE BOOK and wrote with Maciej, Jetty i Ryszard several papers discussing various issues related to the newly defined objects. We have related inhibitor bounded Petri nets with mutexes to step traces and their invariant structures, provided many useful analysis techniques and tools or classify step alphabets with some applications for their synthesis, to give examples not exhausting the topic.

The long and fruitful period of my collaboration with Maciej during this time was not limited to the trace theory. It turned out that together with his wife Marta Pietkiewicz-Koutny, he was involved in a project that aim to propose a useful mathematical framework for globally asynchronous, locally synchronous systems. Taking the opportunity, I jumped into this project on the stage when the first algorithm was already written. Together with Maciej and Marta, we refined the mathematical part of the algorithm and extend underlying concurrency taxonomy. This line of research systematized local view to persistence and formally introduced its second facet – called nonviolence. After some incremental corrections and extensions made during our meetings in Newcastle and Toruń, the final taxonomy was presented in 2017 in the form of paper "An extension of the taxonomy of persistent and nonviolent steps" published in Information Sciences with "Gold Open Access".

In 2016, together with New Year's greetings, Maciej surprised me asking, whether I am interested in reversible computations. The reason was his opportunity to visit Toruń and discuss with me their paradigms in the view of Petri nets. He correctly predicted that there is no need to repeat such propositions twice to me. And in February we met for a two-week Short-Term Scientific Mission in Toruń. Like usually in the collaboration with Maciej, the results came out of the blue. We proposed the framework and together with the rest of FoLCo team prepared a paper for RC'16. This way we also have some material to present in the meeting of COST Action that funded Maciej's scientific mission already in March 2016. We managed to prove that even the question whether adding a strict reverse to a single transition in Petri net changes its behaviour (enlarges the set of reachable markings) is undecidable. After some time (and supported later by Evgeny Erofeev) we conducted some tests getting opposite result in the case of bounded nets. The decidability of considered question was not surprising, but it appears that after delicate relaxing of the aim (namely by allowing to split reverses) we are always able to revert whole system.

The topic of reversible computations in view of Petri nets became our main common research direction for the following two years. Together with David de Frutos Escrig we focused on possibility of reversing bounded systems without

splitting (with the paper "An efficient characterization of Petri net solvable binary words" presented in ATPN'18) as well as reversing steps and multisteps, and we had three common Short Term Scientific Missions (two in Newcastle and one in Madrid), the fourth STSM is planned in Toruń. This research direction proposed by Maciej opened up a possibility to met and cooperate with Ivan Lanese (Bolonia) and Anna Philippou (Nicosia).

Finally, we reached another topic of our common interests – Reaction Systems. During the STSM in Madrid in 2017 we had an opportunity to utilize shared evenings and conduct discussions in casual atmosphere. We made some informal proofs related to previous common research (like minimality of our axiom system for invariant order structures) and started the discussion on Reaction Systems. It materialized as a common paper (written together with Grzegorz Rozenberg and Jetty Kleijn) – "Reaction Systems, Transition Systems, and Equivalences". Maciej presented its insights during the First International Workshop on Reaction Systems and we plan to lift our collaboration in this field to more formalized level.

The invitation to write such informal paper to celebrate Maciej's 60th birthday is for me the priceless opportunity to summarise and appreciate all the income that I got from our collaboration. How his patience and incredible ability to match right people with right challenges allowed me to develop and became an independent researcher in the field of Concurrency Theory. I am extremely thankful for all the time that he devoted to our conversations. I am finding our meetings similar to fairy tales – there is a story, some adversities and difficulties (which only underlines that the obtained results are far from trivial), but finally, sometimes with a little help from newly met friends, there is always happy ending! During the past seven years of collaboration we have written a lot of common papers (seventeen of them are indexed by DBLP). I am really glad that the November 2011 took place and I am looking for more such "Novembers" in the future!

How to Mix Concurrency and Choice and Not Explode*

Andrey Mokhov

Newcastle University, United Kingdom
andrey.mokhov@ncl.ac.uk

Abstract. This paper makes two observations. The first one may be surprising but is probably useless. The second one may someday help solve a real problem but is somewhat tedious. The reader is encouraged to take note of the former and make use of the latter.
The first observation is that *concurrency* is wrongly blamed for the dreadful state explosion problem. In fact, *choice* is a more explosive substance that should be handled with great care. Avoid mixing concurrency and choice; but if you must, read on.
The second observation is that the old Divide and Conquer strategy can be used to effectively deal with the mix of concurrency and choice. To conquer concurrency, slash it into fine diamonds. To conquer choice, wield the laws of thought. As a concrete demonstration, this paper presents a compact encoding of the Concurrent Kleene Algebra using Conditional Partial Order Graphs, where the explosion is avoided by dividing concurrency from choice and using partial orders and Boolean algebra, respectively, for their compact representation.

1 Introduction: Concurrency and Choice

Concurrent systems, which comprise multiple components that can operate and interact simultaneously, are notoriously difficult to design, control and reason about. The complexity of concurrent systems grows exponentially with the number of constituent components, and one way of coping with the complexity is to employ formal models for describing the behaviour of concurrent systems.

State graphs have been used for formal description of behaviour for many decades. However, state graphs of concurrent systems are not compact enough due to the *state space explosion problem* [1], which is illustrated in Fig. 1: as the number of concurrent events in a system increases by one, the state graph describing possible orders of their occurrence doubles in size: the state graph of a system with n concurrent events will therefore have 2^n states.

More surprisingly, state space explosion may also occur in fully sequential systems if they contain a lot of choices. In fact, as illustrated by Fig. 2, the explosion due to choice is not bounded by $O(2^n)$, but can reach $\Theta(n2^n)$ in some cases. The state graphs in Fig. 2 model systems that, given an input choice of one of 2^{n-1} possible behaviours, generate the corresponding *birmutations* on a set of n events $S = \{e_1, e_2, \cdots, e_n\}$, as defined below.

* This paper is dedicated to Prof Maciej Koutny on the occasion of his 60th birthday.

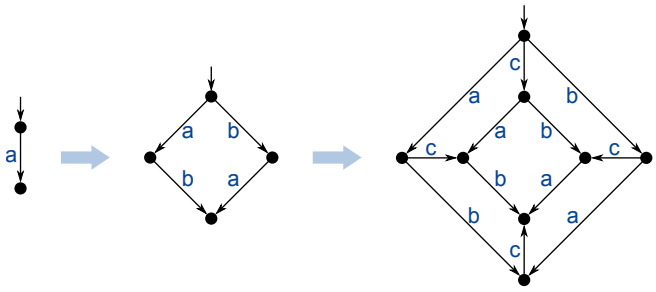


Fig. 1. State space explosion due to concurrency: 2^n states for n events.

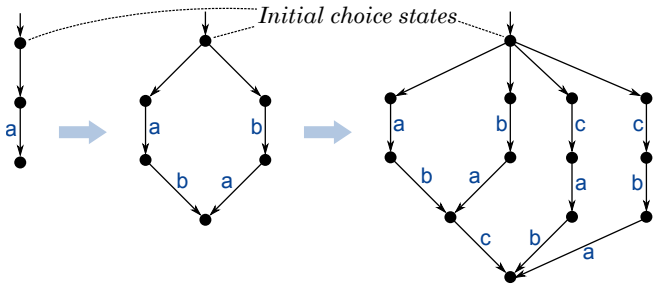


Fig. 2. State space explosion due to choice: $(n+3)2^{n-2}+1$ states for n events.

Definition 1 (Birmutations). *The set of* birmutations B_n *comprises* 2^{n-1} *event sequences that are chosen out of* $n!$ *permutations on* n *events. The chosen subset is recursively defined as follows:*

- *There is one birmutation on the singleton set* $S = \{e_1\}$*: the sequence* e_1*. That is,* $B_1 = \{e_1\}$*.*
- *When* $n > 1$*,* B_n *is obtained by inserting* e_n *either at the beginning or at the end of all birmutations* B_{n-1}*. That is,* $B_n = (e_n \circ B_{n-1}) \cup (B_{n-1} \circ e_n)$*, where* $\circ$ *denotes concatenation of an event to all sequences in a given set.*

Birmutations $B_{1\ldots4}$ are given below in the lexicographic order. We use $\{a, b, c, d\}$ to denote events instead of e_i for clarity.

- $B_1 = \{a\}$.
- $B_2 = \{ab, ba\}$.
- $B_3 = \{abc, bac, cab, cba\}$.
- $B_4 = \{abcd, bacd, cabd, cbad, dabc, dbac, dcab, dcba\}$.

A simple state graph representation of a system generating birmutations B_n would explicitly enumerate all 2^{n-1} sequences, resulting in a state graph of size $1 + (n+1)2^{n-1}$, where the leading 1 corresponds to the initial choice state, and $n+1$ states are used to model the generation of a sequence of n events.

This representation is clearly not the most compact one, because many of the 2^{n-1} sequences have common suffices which can be merged, thereby reducing the size of the state graph. For example, sequences *abcd* and *bacd* have the same suffix *cd* and can therefore be merged after the prefixes *ab* and *ba*. However, it turns out that by merging common suffices one cannot achieve any asymptotic improvement: the resulting state graph will still contain $\Theta(n2^n)$ states.

More precisely, let T_n denote the smallest size of the state graph describing birmutations B_n. Then T_n satisfies the following recurrence:

$$T_n = 2T_{n-1} + 2^{n-2} - 1.$$

Note that sequences from the set $L = B_{n-1} \circ e_n$ have no common suffices with those from the set $R = e_n \circ B_{n-1}$, and their state graphs are therefore disjoint, apart from the initial state – hence the term $2T_{n-1} - 1$. All sequences from the set L have the common suffix e_n, which is merged; all sequences from the set R have common prefix e_n, which cannot be merged, resulting in the term 2^{n-2}. The base case $T_1 = 3$, as well as the cases $T_2 = 6$ and $T_3 = 13$ are illustrated in Fig. 2. By solving the recurrence we obtain $T_n = (n+3)2^{n-2} + 1 = \Theta(n2^n)$.

As state graphs grow, humans (and at some point machines too) lose the ability to comprehend them, which motivates computer scientists to search for more compact models, such as *partial orders* [2], *event structures* [3], *Petri nets* [4], *structured occurrence nets* [5], numerous *process algebras* [6], *conditional partial order graphs* [7][8] and many others.

To deal with concurrency, a common approach is to dissect state graphs into *diamonds*, i.e. sets of independent events, and represent these diamonds compactly using partial orders. To deal with choices, one can employ *Boolean algebra* [9] to compactly describe the conditions for generating particular events. In this paper we demonstrate how these two approaches can be combined to compactly describe systems with a mix of concurrency and choice.

2 Concurrent Kleene Algebra

Concurrent Kleene Algebra (CKA) was introduced by Hoare *et al.* [10] as a unifying theory axiomatising the fundamental concepts of choice, sequential and concurrent composition, and iteration. Many models of CKA have been studied to date, e.g., see [11]. In this paper we show that the *Conditional Partial Order Graph* (CPOG) formalism [7][8] is a compact model of CKA.

In this section we briefly recap basic CKA definitions [10]. We start by introducing three common constants:

- *Bottom* $\bot$ is a contradictory specification, which permits no behaviour and can be equated to the predicate *false*.
- *Skip* 1 describes doing nothing. Note, that this is different from $\bot$; indeed, specification 1 is *implementable* (by doing nothing), while $\bot$ is impossible to implement (by its definition).

– *Top* $\top$ is the opposite of bottom in the sense that it permits any behaviour; it can be equated to the predicate *true*. Any behaviour, including doing nothing, is an admittable implementation of $\top$.

The algebra provides a rich collection of composition operators, the most important of which are listed below.

– *Sequential composition* $(p; q)$ describes the execution of both p and q, where p finishes before q starts. Sequential composition is associative, has 1 as unit and $\bot$ as zero.
– *Concurrent composition* $(p|q)$ describes the execution of both p and q, where p and q can start and finish together (but not required to do so). During the execution p and q are allowed to communicate with each other and with the environment. This operator is both associative and commutative, has 1 as unit and $\bot$ as zero.
– *Choice* $(p \cup q)$ describes the execution of either p or q. Choice is associative, commutative and idempotent, has $\bot$ as unit and $\top$ as zero. The operator is also distributive by both sequential and concurrent composition:

$$p; (q \cup r) = (p; q) \cup (p; r) \quad p|(q \cup r) = (p|q) \cup (p|r).$$

The notion of *refinement*, which is central for formal design and verification methods, is defined as a reflexive and transitive relation $p \Rightarrow q$, which holds iff p's behaviour is included in that of q. This can be equivalently expressed using the choice operator:

$$p \Rightarrow q \quad \text{iff} \quad p \cup q = q.$$

The refinement ordering has $\bot$ as the minimum (empty) behaviour and $\top$ is the maximum (any) behaviour and $\bot \Rightarrow \top$ (compare this to logical implication, where $\mathit{false} \Rightarrow \mathit{true}$).

An important law combining the above concepts is the *exchange law*, which is the most general form of so-called *concurrency reduction* that is often used when implementing a concurrent system using interleaving:

$$(p|q); (p'|q') \;\Rightarrow\; (p; p')|(q; q').$$

An interesting consequence of the exchange law is

$$p; q \;\cup\; q; p \;\Rightarrow\; p|q,$$

that is, one possible way to implement a concurrent composition is by combining choice with sequential composition.

In this work we focus on the set of operations and laws defined above. We refer the reader to [10][11], where a much broader exposition can be found.

3 Sets of Maximal Partial Orders

In this section we demonstrate that a set of maximal partial orders is a model of CKA, which is our first step towards compact models introduced in §4.

3.1 CKA without choice

Consider a partial order $P = (E, \prec)$, where $E \subseteq \mathcal{E}$, $\prec \subseteq E \times E$, and $\mathcal{E}$ is a universe of *events* that can occur in all possible behaviours. We can define some elements of CKA as follows:

- Skip 1 is the empty partial order:

$$1 \stackrel{df}{=} (\emptyset, \emptyset).$$

- Sequential composition of $P_1 = (E_1, \prec_1)$ and $P_2 = (E_2, \prec_2)$ is

$$P_1; P_2 \stackrel{df}{=} (E_1 \cup E_2, \prec_1 \cup \prec_2 \cup E_1 \times E_2).$$

  In words, assuming $E_1 \cap E_2 = \emptyset$ we schedule all events of P_1 to occur before all events of P_2.
- Concurrent composition of $P_1 = (E_1, \prec_1)$ and $P_2 = (E_2, \prec_2)$ is

$$P_1 | P_2 \stackrel{df}{=} (E_1 \cup E_2, \prec_1 \cup \prec_2).$$

  In words, assuming $E_1 \cap E_2 = \emptyset$ events of P_1 occur concurrently to events of P_2. (It is sometimes useful to consider the case when $E_1 \cap E_2 \neq \emptyset$, where the above formulation synchronises common events, thus making communication between P_1 and P_2 possible.)
- Refinement $P_1 \Rightarrow P_2$ of $P_1 = (E_1, \prec_1)$ and $P_2 = (E_2, \prec_2)$ is

$$P_1 \Rightarrow P_2 \stackrel{df}{=} E_1 = E_2 \wedge \prec_2 \subseteq \prec_1,$$

  which means that P_1 permits less concurrency than P_2.

Theorem 1. *Both* ; *and* | *as defined above are associative and have* 1 *as unit; furthermore,* | *is commutative, and the exchange law holds.*

Proof. Follows from Theorem 3.5 of [11]: (p,s,c) is replaced by $(E,\prec,(E\times E)\setminus \prec)$.

Similarly to the *trace model* [11], the *partial order model* of CKA is not powerful enough for handling the choice operator. The next subsection shows that *sets of partial orders* are sufficiently powerful.

3.2 Modelling choice

Consider a set of partial orders $S = \{P_1, P_2, ...\}$, where all constituent partial orders $P_k = (E_k, \prec_k)$ are defined on the universe of events $\mathcal{E}$, that is $E_k \subseteq \mathcal{E}$, and the set S is *downward closed*, i.e., if $P \in S$ and $P' \Rightarrow P$ then $P' \in S$. This construction is inspired by [11]. The downward closure operation will henceforth be denoted as *dc*.

We can now lift previously defined operations on partial orders to sets of partial orders, as well as introduce the missing elements of CKA, namely $\bot$, $\top$ and the choice operator:

- Bottom $\bot$ is the empty set of partial orders:

$$\bot \stackrel{df}{=} \emptyset.$$

- Skip 1 is the singleton set containing the empty partial order:

$$1 \stackrel{df}{=} \{(\emptyset, \emptyset)\}.$$

- Top $\top$ is the set containing all possible partial orders that can be defined in universe $\mathcal{E}$.
- Sequential composition of S_1 and S_2 is

$$S_1; S_2 \stackrel{df}{=} dc\left(\{\ P_1; P_2 \mid P_1 \in S_1,\ P_2 \in S_2\}\right),$$

  where $P_1; P_2$ is sequential composition of partial orders defined previously.
- Concurrent composition of S_1 and S_2 is

$$S_1 | S_2 \stackrel{df}{=} dc\left(\{\ P_1 | P_2 \mid P_1 \in S_1,\ P_2 \in S_2\}\right),$$

  where $P_1 | P_2$ is concurrent composition of partial orders defined previously.
- Choice between S_1 and S_2 is simply $S_1 \cup S_2$, where $\cup$ is the usual set union.
- Refinement can now be defined via choice as in Section 2:

$$S_1 \Rightarrow S_2 \stackrel{df}{=} S_1 \cup S_2 = S_2,$$

  which can be further simplified to the set inclusion relation:

$$S_1 \Rightarrow S_2 \stackrel{df}{=} S_1 \subseteq S_2.$$

Theorem 2. *Sets of downward closed partial orders is a model for CKA.*

Proof. (Sketch.) Here we only prove the distributivity properties of $\cup$ and the exchange law.

(1) We prove that choice $\cup$ is distributive by ; by using the definition of ; splitting set $q \cup r$ into sets q and r, and using the properties of downward closure, as shown in Fig. 3. The proof of choice distributivity by | is analogous.

(2) The exchange law is proved thanks to its *linearity*, which allows lifting the law from partial orders to downward closed sets of partial orders as explained in §3.2 and Appendix B of [11]. The rest of the laws can be verified analogously.

3.3 Reduction to maximal partial orders

As explained in the previous subsection, sets of downward closed partial orders is a model of CKA. An explicit representation of such sets, however, is very inefficient and cannot be directly used for verification or synthesis; indeed, 6.6 trillion different partial orders can be defined on just 10 events!

$$
\begin{array}{rcl}
p;(q \cup r) & = & \text{(by definition of ;)} \\
dc(\{\ a;b \mid a \in p,\ b \in q \cup r\ \}) & = & \text{(splitting set } q \cup r \text{ into } q \text{ and } r) \\
dc(\{\ a;b \mid a \in p,\ b \in q\ \} \ \cup\ \{\ a;b \mid a \in p,\ b \in r\ \}) & = & \text{(downward closure)} \\
dc(\{\ a;b \mid a \in p,\ b \in q\ \}) \ \cup\ dc(\{\ a;b \mid a \in p,\ b \in r\ \}) & & \text{(by definition of ;)} \\
(p;q) \cup (p;r). & &
\end{array}
$$

Fig. 3. Proof sketch for Theorem 2.

The first step towards compact CKA models is to notice that it is sufficient to keep only *maximal partial orders* and drop the requirement of downward closure. A partial order P is *maximal* with respect to set S, if there is no partial order $P' \in S$ such that $P \neq P'$ and $P \Rightarrow P'$. Indeed, keeping only maximal partial orders does not lead to any loss of information, because all omitted partial orders can be restored by downward closure of the remaining maximal ones.

The above is a significant improvement; however, the number of maximal partial orders is still exponential. The next section discusses two approaches that can be used for compact representation of sets of partial orders.

4 Conditional Partial Order Graphs

A CPOG is a directed graph (V, E), whose *vertices* V and *arcs* $E \subseteq V \times V$ are labelled with Boolean functions, or *conditions*, $\phi : V \cup E \to (\{0,1\}^X \to \{0,1\})$, where $\{0,1\}^X \to \{0,1\}$ is a Boolean function on a set of Boolean *variables* X.

Fig. 4 (the top left box) shows an example of a CPOG H containing 4 vertices $V = \{a, b, c, d\}$, 6 arcs and 2 variables $X = \{x, y\}$. Vertex d is labelled with condition $x + y$ (i.e. 'x OR y'), arcs (b, c) and (c, b) are labelled with conditions x and y, respectively. All other vertices and arcs have trivial conditions 1 (trivial conditions are not shown for clarity); we call such vertices and arcs *unconditional*.

There are $2^{|X|}$ possible assignments of variables X, called *codes*. Each code induces a subgraph of the CPOG, whereby all the vertices and arcs, whose conditions evaluate to 0 are removed. For example, by assigning $x = y = 0$ one obtains graph H_{00} shown in the bottom right box in Fig. 4; vertex d and arcs (b, c) and (c, b) have been removed from the graph, because their conditions are equal to 0 when $x = y = 0$. Different codes can produce different graphs, therefore a CPOG with $|X|$ variables can potentially specify a *family* of $2^{|X|}$ graphs. Fig. 4 shows two other members of the family specified by CPOG H: H_{01} and H_{10}, corresponding to codes 01 and 10, respectively, which differ only in the direction of the arc between vertices b and c. Codes will be denoted in a bold font, e.g. $\mathbf{x} = 01$, to distinguish them from vertices and variables.

It is often useful to focus only on a subset $C \subseteq \{0,1\}^X$ of codes, which are meaningful in some sense. For example, code 11 applied to CPOG H in Fig. 4 produces a graph with a loop between vertices b and c, which is undesirable if arcs are interpreted as causality. A Boolean *restriction function* $\rho : \{0,1\}^X \to \{0,1\}$ can be used to compactly specify the set $C = \{\mathbf{x} \mid \rho(\mathbf{x}) = 1\}$ and its complement

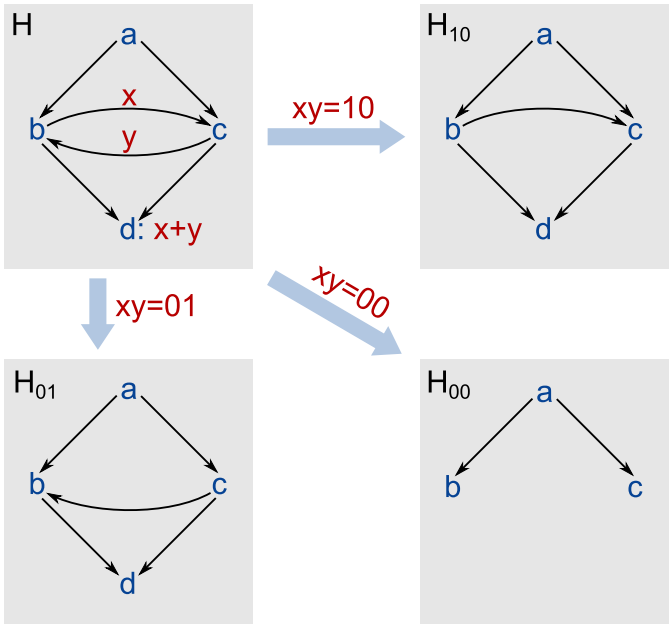


Fig. 4. A CPOG and the associated family of graphs

$DC = \{\mathbf{x} \mid \rho(\mathbf{x}) = 0\}$, which are often referred to as the *care* and *don't care* sets [12]. By setting $\rho = \overline{xy}$ one can disallow the code $\mathbf{x} = 11$ as $\rho(11) = 0$, thereby restricting the family of graphs specified by CPOG H to three members only, which are all shown in Fig. 4.

The *size* $|H|$ of a CPOG $H = (V, E, X, \phi, \rho)$ is defined as:

$$|H| = |V| + |E| + |X| + \left| \bigcup_{z \in V \cup E} \phi(z) \cup \rho \right|,$$

where $|\{f_1, f_2, \ldots, f_n\}|$ stands for the size of the smallest circuit [13] that computes all Boolean functions in the set $\{f_1, f_2, \ldots, f_n\}$.

A CPOG $H = (V, E, X, \phi, \rho)$ is *well-formed* if every allowed code $\mathbf{x}$ produces an acyclic graph $H_{\mathbf{x}}$. By computing the transitive closure $H^*_{\mathbf{x}}$ one can obtain a *strict partial order*, an irreflexive and transitive relation on the set of *events* corresponding to vertices of $H_{\mathbf{x}}$.

We can therefore interpret a well-formed CPOG as a specification of a family of partial orders. We use the term *family* instead of the more general term *set* to emphasise the fact that partial orders are *encoded*, that is each partial order $H^*_{\mathbf{x}}$ is paired with the corresponding code $\mathbf{x}$. For example, the CPOG shown in Fig. 4 specifies the family comprising the partial order H^*_{00}, where event a precedes concurrent events b and c, and two total orders H^*_{01} and H^*_{10} corresponding to sequences *acbd* and *abcd*, respectively.

It has been demonstrated in [14][15] that CPOGs are a compact model for representing families of partial orders. In particular, they can be exponentially more compact than *labelled event structures* [3] and *Petri net unfoldings* [16]. Furthermore, for some applications CPOGs provide more comprehensible models than other widely used formalisms, such as *finite state machines* and *Petri nets*, as has been shown in [7] and [17].

4.1 CPOGs as a compact model for CKA

CPOGs are capable of representing arbitrary sets of partial orders compactly and can therefore be considered a compact CKA model. Below we establish a correspondence between CPOG and CKA constructs. Bear in mind that a CPOG is a quintuple $H = (V, E, X, \rho, \phi)$.

- Bottom $\bot$ is the empty set of partial orders, which can be obtained by setting the restriction function ρ to *false*:

$$\bot \stackrel{df}{=} (\emptyset, \emptyset, \emptyset, 0, \emptyset).$$

- Skip 1 is the singleton set containing the empty partial order, which should be permitted by setting $\rho = true$:

$$1 \stackrel{df}{=} (\emptyset, \emptyset, \emptyset, 1, \emptyset).$$

- Top $\top$ is the set containing all partial orders that can be defined in universe $\mathcal{E}$:

$$\top \stackrel{df}{=} (\mathcal{E}, \mathcal{E} \times \mathcal{E}, X, 1, \phi),$$

  where X contains a variable v_z for each vertex and arc z in the graph, and $\phi(z) = v_z$. In words, top $\top$ is described by a most general CPOG that has different single-literal conditions on every graph element, which allows to obtain any possible subgraph from it by setting variables v_z accordingly. This definition demonstrate compactness of CPOGs: the size of the definition is quadratic in $\mathcal{E}$, yet it describes an exponential number of different partial orders that can be defined on the events in $\mathcal{E}$.
- Sequential composition of CPOGs $H_k = (V_k, E_k, X_k, \rho_k, \phi_k)$ for $k \in \{1, 2\}$ is

$$H_1; H_2 \stackrel{df}{=} (V_1 \cup V_2, E_1 \cup E_2 \cup E_1 \times E_2, X_1 \cup X_2, \rho_1 \wedge \rho_2, \phi^{seq}),$$

  where ϕ^{seq} can be defined as follows:

$$\phi^{seq}(v) = \begin{cases} \phi_1(v) & \text{if } v \in V_1, \\ \phi_2(v) & \text{if } v \in V_2. \end{cases}$$

$$\phi^{seq}(u \to v) = \begin{cases} \phi_1(u \to v) & \text{if } u \to v \in E_1, \\ \phi_2(u \to v) & \text{if } u \to v \in E_2, \\ 1 & \text{if } u \in V_1, v \in V_2, \\ 0 & \text{otherwise.} \end{cases}$$

  We assume that $V_1 \cap V_2 = \emptyset$ as before, and $X_1 \cap X_2 = \emptyset$ (the latter may be relaxed in case we want to allow H_1 and H_2 to synchronise on their *choices*).

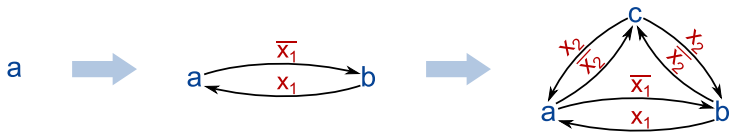


Fig. 5. CPOG models for birmutations B_1, B_2 and B_3.

– Concurrent composition of CPOGs is

$$H_1|H_2 \stackrel{df}{=} (V_1 \cup V_2, E_1 \cup E_2, X_1 \cup X_2, \rho_1 \wedge \rho_2, \phi^{par}),$$

where ϕ^{par} can be defined as follows:

$$\phi^{par}(v) = \begin{cases} \phi_1(v) & \text{if } v \in V_1, \\ \phi_2(v) & \text{if } v \in V_2. \end{cases}$$

$$\phi^{par}(u \to v) = \begin{cases} \phi_1(u \to v) & \text{if } u \to v \in E_1, \\ \phi_2(u \to v) & \text{if } u \to v \in E_2, \\ 0 & \text{otherwise.} \end{cases}$$

We assume that $V_1 \cap V_2 = \emptyset$ as before (unless we want to allow H_1 and H_2 to synchronise on their *events*), and $X_1 \cap X_2 = \emptyset$ (unless we want to allow H_1 and H_2 to synchronise on their *choices*).

– Choice between $H_1 = (V_1, E_1, X_1, \rho_1, \phi_1)$ and $H_2 = (V_2, E_2, X_2, \rho_2, \phi_2)$ is

$$H_1 \cup H_2 \stackrel{df}{=} (V_1 \cup V_2, E_1 \cup E_2, X_1 \cup X_2, \rho_1 \oplus \rho_2, \phi^{par}),$$

where $X_1 \cap X_2 = \emptyset$ (this requirement can be relaxed if $\rho_1 \wedge \rho_2 = 0$, that is H_1 and H_2 do not share any codes).

– Refinement $H_1 \Rightarrow H_2$ for $H_1 = (V_1, E_1, X_1, \rho_1, \phi_1)$ and $H_2 = (V_2, E_2, X_2, \rho_2, \phi_2)$ can be defined as

$$H_1 \Rightarrow H_2 \stackrel{df}{=} \begin{array}{c} \bigwedge_{v \in V_1} \rho_1\phi_1(v) \Rightarrow \rho_2\phi_2(v) \\ \wedge \\ \bigwedge_{u \to v \in E_2} \rho_2\phi_2(u \to v) \Rightarrow \rho_1\phi_1(u \to v). \end{array}$$

In words, if an event v appears in H_1 it must also appear in H_2, and if there is a dependency constraint $u \to v$ in H_2 then it must also exist in H_1.

The above completes the correspondence between CPOGs and CKA. Importantly, the defined constructs are very compact: they are either linear (parallel composition, choice, refinement) or quadratic (sequential composition, due to quadratic explosion of newly added arcs $E_1 \times E_2$; see [17], where this issue is resolved by using *dummy* vertices), despite potentially operating on exponentially large families of partial orders.

Fig. 5 shows compact CPOG models for birmutations: B_n is described by a CPOG with n vertices, $n^2 - n$ arcs, and $n - 1$ variables $X = \{x_1, x_2, \cdots, x_{n-1}\}$, such that each birmutation corresponds to one possible code. For example, the code of the birmutation $cab \in B_3$ is $(x_1, x_2) = 01$. The construction is a direct translation of the recursive case $B_n = (e_n \circ B_{n-1}) \cup (B_{n-1} \circ e_n)$ from Definition 1.

5 Final Remarks

The paper has demonstrated that choice can lead to larger state space explosion compared to concurrency, and showed how to avoid the state explosion by combining partial orders with Boolean algebra. As a concrete example, the paper described how CPOGs can be used as compact CKA models with potential applications in verification and synthesis of CKA specifications. Partial automation has already been implemented and reported in [14][17]. The future work includes validation of the approach on real-life case studies.

Algebra of Parameterised Graphs [17] provided an algebraic characterisation for CPOGs; the algebra satisfies the CKA laws defined in Section 2, thereby providing an alternative demonstration that CPOGs are a model for CKA.

CPOGs have been used in a number of applications, in particular for compact representation of processor instruction sets [7], synthesis of on-chip communication controllers [18], and in process mining [19], where they have shown superior performance compared to conventional approaches due to their compactness. We conjecture that CPOGs may be a generally useful CKA model that can lead to efficient verification and synthesis algorithms.

Acknowledgement

This work was conducted during a 6-month research visit to Microsoft Research Cambridge that was funded by Newcastle University, EPSRC (grant reference EP/K503885/1), and Microsoft Research. The author would like to thank Tony Hoare for his comments on an earlier draft of this paper.

References

1. A. Valmari. The state explosion problem. In *Lectures on Petri nets I: Basic models*, pages 429–528. Springer, 1998.
2. A. Mazurkiewicz. Trace theory. In *Petri nets: applications and relationships to other models of concurrency*, pages 278–324. Springer, 1987.
3. M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures and domains, part I. *Theoretical Computer Science*, 13:85–108, 1981.
4. J. Esparza. Decidability and complexity of petri net problem – an introduction. In *Lectures on Petri Nets I: Basic Models*, pages 374–428. Springer, 1998.
5. Maciej Koutny and Brian Randell. Structured occurrence nets: A formalism for aiding system failure prevention and analysis techniques. *Fundamenta Informaticae*, 97(1-2):41–91, 2009.
6. R. Milner. *A calculus of communicating systems*, volume 92. Springer Verlag Berlin, 1980.
7. A. Mokhov. *Conditional Partial Order Graphs.* PhD thesis, Newcastle University, 2009.
8. A. Mokhov and A. Yakovlev. Conditional partial order graphs: Model, synthesis, and application. *IEEE Trans. Computers*, 59(11):1480–1493, 2010.
9. George Boole. *An investigation of the laws of thought: on which are founded the mathematical theories of logic and probabilities.* Dover Publications, 1854.

10. T. Hoare, B. Möller, G. Struth, and I. Wehrman. Concurrent Kleene Algebra and its Foundations. *The Journal of Logic and Algebraic Programming*, 80(6):266–296, 2011.
11. T. Hoare, S. van Staden, B. Möller, G. Struth, J. Villard, H. Zhu, and P. O'Hearn. Developments in concurrent kleene algebra. pages 1–18, 2014.
12. Giovanni de Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Higher Education, 1994.
13. Ingo Wegener. *The Complexity of Boolean Functions*. Johann Wolfgang Goethe-Universitat, 1987.
14. H. Ponce de León and A. Mokhov. Building bridges between sets of partial orders. In *International Conference on Language and Automata Theory and Applications (LATA)*, 2015.
15. Hernán Ponce de León and Andrey Mokhov. Compact and efficiently verifiable models for concurrent systems. *Formal Methods in System Design*, pages 1–25.
16. K. McMillan. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *Computer Aided Verification*, pages 164–177. Springer, 1993.
17. A. Mokhov and V. Khomenko. Algebra of Parameterised Graphs. *ACM Transactions on Embedded Computing*, 13(4s), 2014.
18. Crescenzo D'Alessandro, Andrey Mokhov, Alex Bystrov, and Alex Yakovlev. Delay/phase regeneration circuits. In *13th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2007.*, pages 105–116. IEEE, 2007.
19. Andrey Mokhov, Josep Carmona, and Jonathan Beaumont. Mining conditional partial order graphs from event logs. In *Transactions on Petri Nets and Other Models of Concurrency XI*, pages 114–136. Springer, 2016.

Memorable Occurrences

Brian Randell

School of Computing, Newcastle University, Newcastle upon Tyne, UK
Brian.Randell@ncl.ac.uk

My first interaction with Maciej Koutny was rather one-sided, and indeed unknown to me at the time! This occurrence arose from a project that I initiated at Newcastle University in 1977, on the construction and exploitation of an experimental highly parallel real time control system. The project involved a recently-purchased large collection of hardware that incorporated a half-a-dozen mobile actuators which were to be controlled by a minicomputer and several microcomputers. The movements of the actuators were strictly constrained, and the whereabouts of the actuators relative to their constraints could be monitored with the aid of fifty or so fixed sensors. I should mention that uninformed spectators tended to view, and refer to, this experimental highly parallel real time digital control system as a model railway or train set, in which the movement constraints (which they termed "track sections", which were thirty-two in number) were also the means of delivering power to, and hence causing movement of, selected actuators (or "trains"), between sensors ("stations"). It was as far as I know the first such computer-controlled model railway, at least in any computer science department, anywhere. (The idea for it came from a computer-controlled model railway I'd seen at Toronto University while on sabbatical there – but this was a much simpler system, that merely demonstrated the speed control of just a single engine.)

This was long before the availability of model railways embodying a microprocessor in each engine that could be controlled by signals sent along the rails. But we realised that just a single computer selectively controlling the power supply to each of the set of electrically-isolated track sections, so that a number of "dumb" trains (we acquired six) could be directed to travel independently, would provide a challenging concurrent real time programming environment. In this environment each train (actually just an engine – we didn't bother with passenger carriages or goods wagons) was in effect an independent process, and one of the most obvious interesting problems was how to control these "processes" so as to prevent them from crashing into each other! In fact, potential collisions between the trains were fairly easily prevented, at the cost of occasional partial or complete system deadlocks.

The problem of how to avoid such deadlocks, while facilitating multiple concurrent train movements along pre-defined journeys, was one that Phil Merlin, a visiting colleague, and I investigated and solved, fully and fairly quickly (albeit somewhat informally), using the concepts of occurrence nets, the acyclic directed graph formalism, allied to Petri Nets, with which he was already familiar [HOLT1968]. (Phil was a brilliant young researcher from IBM Research and the Technion, Haifa, who was to die tragically young, just two years later. His memory is honoured to this day at the Technion by the Annual Dr. Philip M. Merlin Memorial Lecture and Prize Award.) We documented our solution in a brief technical note, and left it at that.

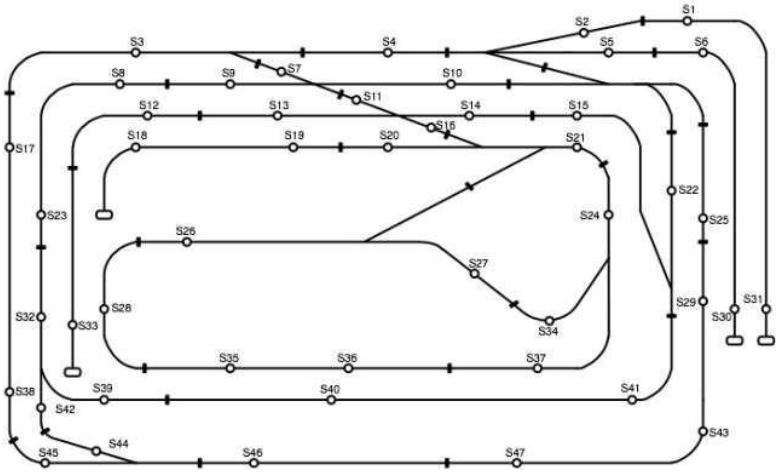
A number of other people elsewhere took up the study of what came to be called "The Merlin-Randell train control problem". It was only years later that I learnt that Maciej Koutny had taken this problem as the subject of his PhD research, and that I saw – I'm afraid I cannot say I read – his impressively lengthy thesis *O Problemie Pociagów Merlina-Randella* [KOUT1984A].

A year later, Maciej became a Research Associate at Newcastle, where his research career – subsequently first as a Lecturer, then Reader, and since 2000 as a Professor of Computing Science and head of Newcastle's theoretical computing group – has flourished ever since.

For a while he continued his study of the Merlin-Randell problem — in [KOUT1985] he provided perhaps the best informal statement of the problem in the following terms:

"*There is a finite set of trains and a layout. The layout is represented by an undirected graph, the nodes of which represent places where trains can reside (stations), the arcs of which represent possible moves. Each station can hold only one train. Each train has a program to follow consisting of a directed path through the graph. The train can leave a station when the station it is immediately to travel to is empty. The problem is to find a synchronisation among train movements which allows parallel movements where possible and enables each journey to be completed*."

This problem of finding such sets of synchronised train movements, which was in essence that of calculating a suitable occurrence net, remained in vogue for quite a while, much like the "Dining Philosophers Problem" before it; I took no part in this research, but I did spend quite a bit of time on various issues to do with controlling the train set, though my main concern remained that of system dependability.



Newcastle's First Train Set

Newcastle's research on dependability started in 1970, and initially concerned the problem of tolerating residual accidental design faults in simple sequential programs. (I had formed the then unfashionable view that current work on proving programs correct would not suffice for large complex programs, and could perhaps be usefully complemented by work on software fault tolerance.) From this, we had soon moved on to considering the problems of hardware and malicious faults, not just software faults, in concurrent programs, and then in distributed computing systems.

Phil Merlin enthusiastically joined in on this research while he was with us, and worked with me on a particular (backward) error recovery problem, i.e. the task of restoring a distributed system to a previous state which it is believed or hoped preceded the occurrence of any existing errors. Our work was based on the use of what we called "Occurrence Graphs",

which we described as similar to occurrence nets, differing mainly in that we viewed an occurrence graph as a dynamic structure that was "generated" as the system that it was modelling was executed, and which contained certain additional information indicating which prior states have been archived and so were restorable.

We tackled the problems arising from concurrency in their full generality, so as to deal with the possibility of there being multiple concurrent faults, some even occurring during error recovery. The solution we produced was a decentralized recovery mechanism that we entitled the "chase protocol". We assumed that each node of a distributed system would hold a record of the part of the occurrence graph that related to its contribution to the overall system behaviour. Then each node would execute a protocol that had the effect of causing error recovery commands to "chase" through the dynamically-growing occurrence graph so as to overtake ongoing error propagations, and the nodes to co-operate in identifying a consistent set of restorable states [MERL1978].

My third involvement with occurrence nets was in collaboration with Eike Best, whilst he was one of Peter Lauer's PhD students, in Maciej's research group at Newcastle. This resulted in the development of a formal model of atomicity [BEST1981]. One of our starting points was David Lomet's work on "Atomic Actions" [LOME1977]. This work was undertaken during David's sabbatical at Newcastle from the IBM Research Center at Yorktown Heights, and involved extending the recovery block concept [HORN1974]. (This involved the use of a "recovery cache" to provide programs with a means of undoing all of the effects of a recovery block after an error had been detected, in order that an alternative program strategy could be attempted.)

"Atomic actions are similar to procedures . . . And in an isolated setting, they behave just like procedures. When concurrent activity is present, the body of an atomic action continues to behave as it did in isolation, while an ordinary procedure, if it accesses shared state, may behave very differently" [LOME1977].

Thus atomic actions allowed the recovery block concept to work in the presence of concurrently executing activities. (They can be viewed as the programming language equivalent to the transaction concept, an idea that the database community was also developing at this time [LOME2011].)

Eike Best and I pursued this idea of an atomic action as a programming concept, rather than a "hardware feature" or a "synchronisation method", and inspired by [MERL1978] were led to propose a formal model for atomic actions based on what we termed structured occurrence graphs. Using an occurrence net to represent a program's activities, the dynamic structure arising from any single programmer-defined atomic action can be collapsed to a single event. Multiple co-existing such collapsings can be performed, as long as the graph remains cycle-free. The concept of structured occurrence graphs involves the imposition of a nested structure of possible collapsings, i.e. of atomic actions, on a conventional occurrence graph. Being nested, one can be sure that multiple co-existing collapsings of any the so-identified sub-graphs would not introduce any cycles into the graph.

My most recent interactions with Maciej again have centred on occurrence nets. A very fruitful, indeed memorable, collaboration ensued when Maciej saw how I was using little occurrence net diagrams to explore whether and how such nets could be used to model the activity of an evolving system. I was actually reconsidering, yet again, the fundamental dependability concepts and definitions that I and colleagues have developed over many years, and whose most complete description is in [AVIZ2004].

I had belatedly realised that our notions about 'fault/error/failure chains' did not cope well with situations in which the set of components (i.e. sub-systems) in a system of systems was

for whatever reason changing. Examples include a large hardware system which suffers component break-downs, reconfigurations and replacements, a large distributed system whose software is continually updated (or patched), a multi-organisational computer system whose human operators undergo regular re-training, or a typical large bureaucracy. However, the little example I often used in my initial thought experiments was that of the confusion that could be caused to someone struggling to create a WORD document when unaware of the fact that the WORD system being used was itself suffering a succession of faults and updates!)

I was trying to use occurrence nets to represent both the activities of such a system, and of whatever was causing it to evolve. I was sketching diagrams that involved multiple interrelated occurrence nets, the relationship I initially tried to formulate being that of "is a behaviour of". The idea was that one occurrence net would portray the activity of the system that was in fact evolving, and another net would portray the activity of the entity (system) that was controlling this evolution. States portrayed in this second (in fact more abstract) net corresponded to the different versions of the evolving system, and were related to the relevant sections of the evolving system's occurrence net. Maciej saw my diagrams, and said "That looks interesting – can I join in?" Needless to say, I warmly welcomed this suggestion!

This led to a much-needed clarification, and a belated formalization, of the concept of behaviour relations, and to our defining a number of other relations between occurrence nets in order to produce what we called "Structured Occurrence Nets" (SONs), as a means of describing the activities of an evolving system and aiding the analysis of its failures. The other types of relations that we defined enabled the activities of separate sub-systems to be distinguished from each other, and provided various means of abstraction, both temporal and spatial, that aided the representation and analysis of complex systems. In total what we had developed was a very general formal notation for representing and investigating complex causalities, a notation that significantly extended the expressive power and practicality of occurrence nets.

These ideas have been documented in a number of papers, e.g. [RAND2007], [KOUT2009], [RAND2011], and Maciej has led an EPSRC-sponsored research project with the rather contrived title and acronym "UNderstanding Complex eVolution through structured behaviours (UNCOVER)". This project extended the initial definitions of SONs with new features related to alternate behaviours and timing information [BHAT2016], and implemented all the basic functionalities in a protoype tool called SONCraft [LIRA2018]. This tool, which was designed and implemented by Bowen Li, provides a user-friendly graphical interface which facilitates model entry, supports interactive visual simulation, and enables the use of a set of analytical tools, e.g. for reachability analysis. (SONCraft is based on Workcraft, a framework for interpreted graph models, a product of Newcastle's joint EEE/CS Asynchronous Systems Laboratory – see http://workcraft.org/, which provides download links for SONCraft.)

The original motivation for SONs and for building SONCraft was the problem of analyzing the causes of failures in complex *computer* systems. But in fact the SON ideas, and the SONCraft tool are as indicated above very general in nature, and essentially just about the representation of causality. Indeed one of the application areas we found ourselves concentrating on was in fact criminal investigation. This involved our viewing criminals or criminal gangs as 'systems', and their crimes as 'failures'! Our view is that portrayals of the criminal activities that result in crimes, i.e. of the observed or inferred causality links between criminals' various actions, are potentially useful both for identifying the criminals, and for portraying and explaining the evidence-based reasoning that will later be needed to convict them.

The UNCOVER project has thus had numerous very useful interactions with a number of police and other investigative agencies, and tool suppliers to these agencies. As a result, rather pleasingly, support has recently been obtained from Innovate UK for a two-year "Knowledge Transfer Project" that will enable the collaborative incorporation by the Bristol-based company Clue Computing Co. of SONcraft-like facilities into their CLUE investigation support system.

In fact our work together on SONs has been the most memorable of the various occurrences I have described here of my very fruitful interactions with Maciej over the years. What is particularly pleasing is that it is still continuing, as we explore the possible ramifications of what we have come to call "structured causality".

References

[AVIZ2004] A. Avizienis A, J.-C. Laprie, B. Randell, C. Landwehr: *Basic concepts and taxonomy of dependable and secure computing*. IEEE Transactions on Dependable and Secure Computing 2004, 1(1), pp.11-33.

[BHAT2016] A. Bhattacharyya, B. Li, B. Randell: *Time in Structured Occurrence Nets*. In: International Workshop on Petri Nets and Software Engineering (PNSE'16), pp.35-55.

[BEST1981] E. Best, B. Randell: *A Formal Model of Atomicity in Asynchronous Systems*. Acta Informatica 16 (1981) pp. 93-124.

[HOLT1968] A.W.Holt, R.M.Shapiro, H.Saint, S.Marshall. *Information System Theory Project*. Report RADC-TR-68-305, US Air Force, Rome Air Development Center (1968).

[HORN1974] J.J. Horning, H.C. Lauer, P.M. Melliar-Smith, B. Randell: A Program Structure for Error Detection and Recovery, in *International Symposium on Operating Systems: Theoretical and Practical Aspects.* (Springer Verlag, 1974), pp. 171-187.

[KOUT1984A] M. Koutny: *O Problemie Pociagoẃ Merlina-Randella (in Polish).* PhD Thesis, Department of Mathematics, Warsaw University of Technology, Warsaw, Poland (1984)

[KOUT1984] M. Koutny: *On the Merlin-Randell Problem of Train Journeys*. In: Proc. of 6th International Symposium on Programming, Springer Verlag, Lecture Notes in Computer Science 167 (1984) 179–190.

[KOUT2009] M. Koutny, B. Randell: *Structured Occurrence Nets: A formalism for aiding system failure prevention and analysis techniques*. Fundamenta Informaticae 2009, 97(1-2), 41-91.

[LIRA2018] B. Li, B. Randell, A. Bhattacharyya, T. Alharbi, M. Koutny: *SONCraft – A Tool for Construction, Simulation and Analysis of Structured Occurrence Nets*. In: Proc. ACSD (2018).

[LOME1977] D.B. Lomet: *Process Structuring, Synchronization and Recovery Using Atomic Actions*. In: Language Design for Reliable Software, pp. 128-137.

[LOME2011] D.B. Lomet: Transactions: From Local Atomicity to Atomicity in the Cloud. In Dependable and Historic Computing. Springer (2011) pp.38-51.

[MERL1978] P. Merlin, B. Randell: *State Restoration in Distributed Systems*. In: FTCS-8, IEEE Toulouse (1978) pp. 129-137.

[RAND2007] B. Randell, M. Koutny: *Failures – Their Definition, Modelling and Analysis*. In: Theoretical Aspects of Computing: 4th International Colloquium (ICTAC). 2007, Macao, China: Springer-Verlag.

[RAND2011] B. Randell: *Occurrence Nets Then and Now – The Path to Structured Occurrence Nets*. In: Application and Theory of Petri Nets. 2011, Springer-Verlag.

Reflections on Our Collaboration and Friendship

Grzegorz Rozenberg

University of Leiden (The Netherlands)
University of Colorado at Boulder (USA)
g.rozenberg@liacs.leidenuniv.nl

It is difficult for me to get used to the fact that Maciej turns 60 years old in September. He was always a young kid for me and then, all of a sudden, one of my favorite kids becomes 60 years old!

I do not remember exactly when I met Maciej for the first time, but this must have happened during one of the Petri Nets conferences in the 1980's.

I like to think that I had some influence on his scientific development and academic career. His early research on Petri nets was perhaps influenced by my work on step semantics, elementary net systems, and theory of traces. Then, he and his wife Marta got very involved in research on the synthesis of concurrent systems based on the theory of regions developed by Andrzej Ehrenfeucht and myself.

Gradually a cooperation developed between Maciej (and Marta) and my group in Leiden. In particular, Jetty Kleijn, my special daughter, developed a close scientific cooperation with Maciej and Marta, and in fact (to my delight) they became very close family friends.

I have also succeeded in infecting Maciej with an enthusiasm for problems related to information processing in biology and consequently we (including Jetty) have collaborated quite intensely on reaction systems, a novel model of interactive computation inspired by the functioning of the living cell, developed by Andrzej Ehrenfeucht and myself about 10 years ago. As a result of many years of collaboration, we have developed a close relationship on both professional and personal levels.

It is a pleasure to collaborate scientifically with Maciej. He is a very gifted researcher, really engaged in formulating and solving interesting problems. He is also very efficient and very reliable - these features are also very important in organizational tasks. It is no surprise that he is such an excellent Chair of the Steering Committee for the well established International Conference on Applications and Theory of Petri Nets and Concurrency. Having invested so much energy and enthusiasm in building up this community, it is very nice and reassuring for me to know it is in very good hands.

On a personal level, Maciej is a very nice friend to have. We have visited each other many times. He is always a very good host and I feel very comfortable (also from the health point of view) and very welcome whenever I visit him in Newcastle. He is a real food connoisseur and an excellent cook. I always enjoy the dinners he prepares as well as his guided culinary "escapades" in Newcastle to sample excellent foods of various origin (Chinese, Iranian, British, …), including the famous Newcastle fish and chips. My stays in Newcastle are always very pleasant and invigorating with the special benefit of being able to enjoy the company of Marta - Maciej is really lucky to get such a joyful and interesting life partner.

The question arises whether meeting Maciej, and subsequently developing a special professional and personal relationship, was accidental or was it predestined? This question is clearly important for the magician in me, but it seems to be difficult to answer. Here are some relevant facts.

- We both come from Poland.

- Even though I am from Warsaw, while Maciej spent his childhood in Torun, we share an Alma Matter, viz., Warsaw University of Technology. Maciej started his studies there long after I left Poland.
- When Maciej moved from Torun to Warsaw, he got a room in a student house, which was located just about 200 meters from the house where I was living in Warsaw.

As my famous son says "coincidences do not exist", and so I tend to believe that meeting Maciej was predestined, a very nice predestination indeed.

Dear Maciej:

Congratulations on your 60th birthday and best wishes for good health, productivity, satisfaction, and joy (your grandchildren will take care of this) for many years to come. Thank you for your friendship.

Big Birthday Hugs, also from Maja,
Grzegorz Rozenberg
Bilthoven, August 2018

Acknowledgement: I am grateful to Paul Smit for giving me a permission to use the wonderful picture above, which was taken by him.

Reversibility, Event Structures and Petri Nets

Irek Ulidowski

Department of Informatics, University of Leicester, England

I came to know about Maciej in the nineties as a brilliant researcher from the University of Newcastle upon Tyne, a co-author with Ryszard Janicki of several seminal papers on non-interleaving models of concurrency [6, 5]. We met I think in State College, Pennsylvania at the 11th International Conference on Concurrency Theory CONCUR 2000. Maciej had a paper with Victor Khomenko on LP deadlock checking using partial order dependencies [7], and I presented a joint work with Shoji Yuen on process calculi for eager bisimulation in [15]. I thought Maciej was a very friendly and quite knowledgeable in all aspects of computer science and life in general. What I also remember well is that we enjoyed a local speciality of alligator meat at one of the evening meals.

Next time we attended CONCUR together was in 2011 in Aachen, Germany, where our proposal to host the 23rd edition of CONCUR in Newcastle upon Tyne in 2012 was approved. Naturally, we were delighted with the decision, and it was clear to me that Maciej's standing in the concurrency community and his skills as well as the allure of Newcastle and Durham were the deciding factors that persuaded the Steering Committee of CONCUR to bring the conference to England (for only the second time ever). Apart from celebrating our good fortune, we discussed among others my interest in the modelling of reversible computation in process calculi, the topic that I started to work on in 2004. Maciej told me about the rôle of reversibility in traditional Petri nets research (during a coach trip to Maastricht for a social event dinner). It was a peripheral topic in Petri nets research, and we thought that the same held about other areas of research, like program inversion, debugging, circuit design, semantics of concurrency. Wouldn't it be interesting we thought to involve all those scientists from different fields of computer science who are interested in reversibility into an international network to discuss, research and apply reversibility?

A year later at CONCUR in Newcastle, I was working on extending event structures with reversibility to model both the causal-consistent reversibility [4, 10, 8] and the so-called out-of-causal order reversibility [13], which is present in many bio-chemical reactions. There was not much opportunity for research in Newcastle because Maciej and I were busy with running the conference. However taxi rides from Spital Tongues to Newcastle Hilton, the venue of CONCUR 2012, allowed ample time to think. What was started in Newcastle in 2012 resulted in several papers. Iain Phillips and I showed how to reverse prime and asymmetric event structures so that different forms of reversibility can be modelled [11, 12]. Jointly with Shoji, Iain and I proposed the enabling with prevention relation that extends general event structures with reversibility [14].

I finally enticed Maciej to the topic of reversibility a few years after CONCUR 2012. I applied for a COST Action grant titled "Reversible Computation:

extending the horizons of computing" in 2014. COST Action grants aim to support networking, training, meetings and one to one research visits, and our idea was to use COST Action funds to develop, promote and popularise reversible computation research in Europe. When the grant was awarded and the COST Action IC1405 was established in the spring of 2015, Maciej became one of the members of the Management Committee of the Action. In collaboration with Łukasz Mikulski from Nicolas Copernicus University in Toruń (Poland) Maciej started working on reversibility in Petri nets. Soon afterwords, Kamila Barylska and Marcin Piątkowski from Toruń joined the research and they published a number of papers including [2, 3], where they consider a number of ways that the reverse versions of transitions can be added to Petri nets, and how adding such reverse transitions impacts on their behaviour. This work within the framework of the Action IC1405 led also to cooperation with Anna Philippou and Kyriaki Psara from the University of Cyprus who also became interested in reversibility in Petri nets [9]. They have recently carried out joint work with Anna Gogolińska on the modelling of reversing computation in coloured Petri nets [1]. Thanks to Maciej's involvement and encouragement, and the support of IC1405, there is now a group of young researchers who work on reversibility in Petri nets. As for me, I am still aiming to join this research effort and transfer some of the reversible event structures techniques to Petri nets.

I appreciate very much Maciej's friendship, advice and generosity, and hope that we continue in this way for some time to come.

References

[1] K. Barylska, A. Gogolinska, L. Mikulski, A. Philippou, M. Piatkowski, and K. Psara. Reversing computations modelled by coloured Petri nets. In *Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data 2018*, volume 2115 of *CEUR Workshop Proceedings*, pages 91–111, 2018.

[2] K. Barylska, M. Koutny, L. Mikulski, and M. Piatkowski. Reversible computation vs. reversibility in Petri nets. *Science of Computer Programming*, 151:48–60, 2018.

[3] K. Barylska, E. Erofeev, M. Koutny, L. Mikulski, and M. Piatkowski. Reversing transitions in bounded Petri nets. *Fundamenta Informaticae*, 157(4): 341–357, 2018.

[4] V. Danos and J. Krivine. Reversible communicating systems. In *Proceedings of CONCUR 2004*, volume 3170 of *LNCS*, pages 292–307. Springer, 2004.

[5] R. Janicki, and M. Koutny. Semantics of inhibitor nets. *Information and Computation*, 123(1):1–16, 1995.

[6] R. Janicki, and M. Koutny. Structure of concurrency. *Theoretical Computer Science* 112(1): 5–52, 1993.

[7] V. Khomenko, and M. Koutny. LP deadlock checking using partial order dependencies. In *Proceedings of CONCUR 2000*, volume 1877 of *LNCS*, pages 410–425. Springer 2000.

[8] I. Lanese, C.A. Mezzina, and J-B. Stefani. Reversing higher-order pi. In *Proceedings of CONCUR 2010*, volume 6269 of *LNCS*, pages 478–493. Springer, 2010.

[9] A. Philippou and K. Psara. Reversible computation in Petri nets. To appear in *Proceedings of RC 2018*. 2018.

[10] I.C.C. Phillips and I. Ulidowski. Reversing algebraic process calculi. *Journal of Logic and Algebraic Programming*, 73:70–96, 2007.

[11] I.C.C. Phillips and I. Ulidowski. Reversibility and asymmetric conflict in event structures. In *Proceedings of CONCUR 2013*, volume 8052 of *LNCS*, pages 303–318. Springer, 2013.

[12] I.C.C. Phillips and I. Ulidowski. Reversibility and asymmetric conflict in event structures. *Journal of Logic and Algebraic Methods in Programming*, volume 84(6):781–805, 2015.

[13] I.C.C. Phillips, I. Ulidowski, and S. Yuen. A reversible process calculus and the modelling of the ERK signalling pathway. In *Proceedings of Reversible Computation 2012*, volume 7581 of *LNCS*, pages 218–232. Springer, 2013.

[14] I.C.C. Phillips, I. Ulidowski, and S. Yuen. Modelling of bonding with processes and events. In *Proceedings of Reversible Computation 2013*, volume 7948 of *LNCS*, pages 141–154. Springer, 2013.

[15] I. Ulidowski, and S. Yuen. Process languages for rooted eager bisimulation. In *Proceedings of CONCUR 2000*, volume 1877 of *LNCS*, pages 275–289. Springer 2000.

Maciej Koutny 60: Congratulations!

Wil M.P. van der Aalst

Process and Data Science (PADS), RWTH Aachen University, Germany.
wvdaalst@pads.rwth-aachen.de

Abstract. This Festschrift celebrates Maciej Koutny's 60th birthday in September 2018. I'm very happy to contribute to this wonderful event and I would like to congratulate Maciej and thank him for his scientific achievements, service to the community, and friendship. In this brief contribution, I reflect on his career from two angles: (1) research in Newcastle and (2) service to the Petri net community.

1 Petri Net Research in Newcastle

Thirty-three years ago (i.e., in 1985) Maciej Koutny joined the Computing Laboratory of the University of Newcastle upon Tyne to work as a Research Associate. He was the key person to turn Newcastle into stronghold for Petri net research. In 1986, Maciej became a lecturer and later got a readership position. Since 2000, Maciej Koutny is a Professor of Computing Science in the School of Computing at Newcastle University. Maciej belongs to a generation of smart scientists that moved from eastern Europe (Poland, Russia, Roemania, etc.) to western Europe.

Maciej worked on concurrency and formal methods, often using Petri nets as a representation. He was one of the key persons in the development of Petri net algebra, opacity of transition systems, priorities in nets, net unfoldings, box algebra, and framed temporal logic programming. He greatly contributed to a better understanding of the relationship between temporal logics, Petri nets, and process algebras and studied Petri net based behavioural models of membrane systems.

DBLP shows about 100 conference and workshop papers and around 80 journal papers covering the areas mentioned above. Maciej collaborated with many scientists all over the globe, these include Jetty Kleijn, Ryszard Janicki, Marta Pietkiewicz-Koutny, Lukasz Mikulski, Raymond Devillers, Victor Khomenko, Hanna Klaudel, Alex Yakovlev, Eike Best, Grzegorz Rozenberg, and many others.

2 Supporting and Leading the Petri Net Community

Next to his research efforts, Maciej played and still plays an important role in the Petri net community. He is the chair of the Steering Committee of the International Conferences on Application and Theory of Petri Nets and Concurrency. He followed in the footsteps of Kurt Jensen and Grzegorz Rozenberg. He is also the editor-in-chief of the LNCS Transactions on Petri Nets and Other Models of Concurrency (ToPNoC). He chaired ICATPN'01, ACSD'08, CHINA'08, MeCBIC'10 and CONCUR'12 and was

co-director of the 5th Advanced Course on Petri Nets held in 2010. In 2001 the Petri net conference took place in Newcastle.

In 2011, the Petri net conference was planned to take place in Kanazawa, Japan. However, due to Tohoku earthquake and tsunami, and the tragic events that followed, the conference had to be relocated. Maciej decided to bring the conference to Newcastle upon Tyne for a second time and organized a wonderful event despite the short time to prepare. This shows great leadership and commitment to the community.

I would like to thank Maciej for his great work in supporting and leading our wonderful Petri net community and I'm looking forward to host the next Petri net conference in Aachen in June 2019.

Living in terms of Causes and Effects

Alex Yakovlev

Newcastle University, Newcastle upon Tyne, NE1 7RU, UK

alex.yakovlev@ncl.ac.uk

Abstract. The essay reflects on my most memorable experiences in studying causality and concurrency in the realm of electronic hardware. It covers thirty years of my friendship and collaboration with Maciej Koutny. This bond helped us build an important synergy at the cognitive and methodological level and create a unique academic environment enabling young researchers to tackle theoretical and practical problems in one of the most intriguing fields of computer science and engineering, the area of concurrency theory and asynchronous systems design. For me personally, Maciej has made strong impact on my understanding of semantics of concurrency and discovering the intricacies of the relationship between different models of concurrency that laid foundation to design methods and tools for asynchronous digital hardware. The paper concludes with the discussion of what actually Causality is, is it about representation (purely cognitive notion), or has it a physical meaning? My view it has place in both!

Keywords: Asynchronous Systems, Causality, Concurrency, Energy current, Petri nets, Signal Transition Graphs, Theory of Regions

1 Instead of introduction

When I joined the group of Professor Victor Varshavsky in 1980 as a PhD student ('aspirant' in Russian), I had very little idea about concurrency. However, the group used term "parallel asynchronous processes", actually meaning concurrent processes by them. In order to have a good feel what such processes are actually in real systems, I had a two prong attack at the literature. One side was understanding concurrency by reading about operating systems and various mechanisms used in them, such as semaphores and monitors. The other was modelling and designing the so-called Muller circuits, i.e. circuits whose behaviour didn't depend on delays. My first mentors here were Dr Leonid Rosenblum and Dr Vyacheslav Marakhovsky, as well as two other Varshavsky's aspirants Alex Taubin and Mike Kishinevsky, who had joined the group before me.

Muller's theory of asynchronous or better say speed-independent circuits was a great vehicle in learning the fine grain concurrency effects, such as arbitration, non-persistency and synchronisation. Understanding the dynamic behaviour of those circuits was like going through the baptism of concurrency.

Another good training vehicle was modelling communication protocols, and trying to reduce the model of a protocol between two objects to absolute minimum, keeping only the essential aspects of the inter-object interaction while hiding all the internal actions of the objects.

So, these three ingredients, parallel programs (in operating systems), protocols and asynchronous logic circuits formed me as a researcher in concurrent systems, and that brought me to my PhD level at the end of 1982. Of course, the beloved Petri nets were the key modelling formalism and they equipped me with the necessary understanding of the

relationship between structure and behaviour, static and dynamics, of concurrency models, let alone the wonderful axes of correspondence between Petri nets and asynchronous circuits.

Fig. 1. We weren't even 30 then! (left to right: Maciej Koutny, Luigi Mancini, Wouter Batelaan, Alex Yakovlev, Thomas Dongman Lee and Paul Ezhilchelvan, Newcastle Summer 1985)

Two years later I managed to land in Heathrow as a postdoc, funded by the Higher Education Ministry of the USSR and The British Council, and on the 17th October 1984 I first time arrived in Newcastle to what was then the Computing Laboratory (led by Prof Harry Whitfield). That was the place where I had met Prof Brian Randell, and heard about atomic actions in designing distributed systems, about various types of concurrent systems and languages to program them. I had also met with Prof David Kinniment, whose name was known to me as a pioneer in metastability, synchronisers and arbiters. At the end of 1984, a young research associate arrived from Warsaw to work with Brian on the formalisation of the problem defined as the Merlin-Randell protocol. After a couple of silent passings by each other, we started to talk. And that's how I first met with Maciej. I liked him from the start. We were close in age, in height, and Maciej had a very 'scientific beard', which to me was a good sign. I used to have had beard myself before but the Soviet Education Ministry instructors told me that I should look tidier when I go to Britain and so I had shaved it before that.

That was the start of our great friendship with Maciej. We used to spend hours walking on the coast in Tynemouth or in Newcastle's parks, talking about everything in life. Poland was the first country I had visited abroad as a student (1976), and so I had lots of warm memories of that time. Maciej, on the other hand, had already visited USSR before, too. We could talk about our impressions of each other's country. Also, as it happened both our families we far away from us – and here again was a similarity, let alone that Maciej's daugher Ola and my son Greg were borne in the same year of 1981. In August 1985 I had to go back to Leningrad while Maciej was of course staying in Newcas-

tle and later that year he was joined by Marta and Ola, but that was after I had already gone back to the USSR. We exchanged Christmas cards and yearly updates after that, little did we know then that our roads will meet again soon!

2 Asynchronous circuit theory

Working in Varshavsky's group was both a challenge and great fun. One could design circuits for some real applications such as, for example, a token-ring communication channel for an on-board multi-computer system with fault-tolerance levels achieved due to the effect of self-checking properties of asynchronous logic. At the same time, there was an encouragement to study theory of asynchronous circuits to fully understand and capture, formally, such notions as hazards and deadlocks in these circuits. Varshavsky associated asynchronous circuits with parallel programs and concurrent processes – this was a very useful analogy. There was always a team of people with whom to discuss concurrency. Petri nets were often in the centre of these debates.

One requirement, however, that Varshavsky imposed on us was that the formalisms had to be used in close relation with two main problems in asynchronous circuit design process, one of analysis and the other of synthesis. Analysis concerned taking an asynchronous circuit description and verifying whether its behaviour contained potential hazards, deadlocks and arbitration conditions. Synthesis was concerned with the way how one could derive the logic circuit netlist (i.e. a system of Boolean equations of the circuit gates) from an initial behavioural specification of the circuit.

Analysis in those days was approached as the process of generating the set of reachable states in the so-called Muller model of the circuit. Informally, the Muller model considered a circuit to be a network of logic elements. Each element was a combination of a functional block (described by a Boolean function) with zero delay and a delay element, whose delay was finite but unbounded. Due to this description of an element, it was possible to separate the notion of excitation of the output signal of the element (when the value of its Boolean function was different from the value of the output of its delay element) and the notion of firing the signal, i.e. changing its value to the opposite one, say from 0 to 1, or from 1 to 0. An interesting property of Muller circuits was that of semi-modularity. The circuit was semi-modular if it would never reach a state in which an element would be excited say in its value 0 (we wrote it with an asterisk 0*) and not change to 1. Thus, the circuit with a transition from state where some signal was labelled 0* to the state with a stable 0, would be called non-semi-modular. This behaviour raised an immediate analogy with a notion of non-persistence in Petri nets. To detect non-semi-modularity in a circuit we had to go through all reachable states.

The reachable state space, for the whole circuit, could of course grow fast due to the amount of concurrency in the circuit. So, the team was concerned with finding efficient ways to tackle the state space explosion problem. Alexander Taubin's PhD was largely focused on analysis. Together with Mike Kishinevsky, they developed a method of analysis based on Boolean characteristic functions of the reachable state space as well as characteristic functions of some properties such as semi-modularity of the circuit. At some point, Yury Mamrukov joined the group with a bunch of fresh ideas on how to implement these methods by his highly efficient ways of representing characteristic functions in bracketed form, which was well before the world learned about BDDs! And moreover, he could pack large state vectors in a small number of memory words, so that his software, written in PL (if my memory doesn't let me down!), could easily cope with the state spaces of the scale

of 2^50 or more. We rarely could produce circuits (manually) bigger than 50 signals. So, overall, Varshavsky and Marakhovsky, who were the main circuit generators, were quite happy with analysis … but for some time.

In terms of automating synthesis, things were lagging behind, although the methods and theory was gradually developed. Kishinevsky had developed in his PhD thesis some new methods for synthesis of distributive circuits using NAND only gates, and semi-modular circuits using NAND and NOR gates, with limited fan-in and fan-out. Those were theoretically important results, even though they didn't lead to particularly efficient practical designs. The main ways of synthesis were the following: (i) manual gate-level design with subsequent analysis of semi-modularity, (ii) direct translation of Petri nets to circuits using so called David cells, (iii) formally justified (yet still unautomated procedures) of synthesis from state graphs (aka Muller diagrams) and, (iv) some initial (again, unautomated yet) procedures of synthesis from signal graphs (later called STGs by T.-A. Chu). The latter was the area where I contributed with my PhD thesis with application to the design of controllers for protocols and interfaces, such as UNIBUS, VME Bus etc. Those were relatively small control circuits. Larger controllers, such as those for the above-mentioned token-ring communication channel for our industrial partners developing on-board computers, were designed using direct translation of Petri nets into David cell circuits.

As things progressed into VLSI in the latter half of the 1980s, more computational power emerged. There was no longer a need for Mamrukov's renting night hours at the Computer Centre of Gostinny Dvor (the biggest department store in Leningrad). People could use IBM PCs!

3 Models with true concurrency

We needed models and tools to handle synthesis and analysis more efficiently and, ideally, interactively. The road to efficiency was seen in avoiding an explicit exploration of states for concurrent events produced by interleaving of these events. Namely, if two events A and B were concurrently enabled in the circuit, the reachability analysis explored both traces A;B and B;A, thereby producing four states forming a so-called diamond. As the number of concurrent events (and hence signals) grew as n, the state space exploded as O(2^n). Something different had to be used for analysis and synthesis of highly concurrent models – we all sought to liberate ourselves from the tyranny of interleaving! Leonid Rosenblum and I had already pioneered Signal Graphs in our paper at Turin's Workshop on Timed Petri nets, July 1985. Interestingly, Tam-Anh Chu from MIT had his first paper on STGs published a bit later, in November 1985. We had no idea about each other's work until a few years after that.

That was the time when the model of Change Diagrams emerged. It didn't have a Petri net underneath, though it retained the important aspect of modelling true concurrency in the form of partial orders. Change diagrams captured both forms of causal dependency or precedence, namely strong (AND) causality and weak (OR) causality. The fact that OR causality could be captured in Change Diagrams in an explicit form was very useful because logic circuits, build of gates with NAND, NOR, AND-OR-INVERT functions, ultimately could exhibit both of those forms of precedence. However, one somewhat esoteric feature of Change Diagrams was a bit unwieldy and non-intuitive. Firstly, they required use of negative marking, namely tokens were borrowed from the input arcs which were not active for the OR-causal vertices. Secondly, because Change Diagrams weren't meant to model processes with choice, they had so-called disengageable arcs, in order to capture the

initialisation of a choice-free process. I wasn't happy about those features which I had thought would be difficult to promote to practitioners in the future. Nevertheless, Alex Taubin, Mike Kishinevsky and Alex Kondratyev managed to bring Change Diagrams to the level of excellent software tools TRANAL and TRASYN, that were developed within the suite of tools under Varshavsky's co-operative TRASSA – these were the first self-timed CAD tools in the world that could be run on a PC. A nice book was later published by John Wiley & Sons, which I had a pleasure to translate into English (when I was already in Newcastle), and which described all the theory and algorithms behind analysis and synthesis methods for Change Diagrams. The book was equipped with a CD containing these tools. Some people from the Asynchronous design community around the world used these tools, but they didn't get far, possibly partly because of those unintuitive aspects of Change Diagrams.

Fig. 2. No place for interleaving!? (left to right: Alex Yakovlev, Leonid Rosenblum, Marta Koutny, Maria Yakovlev, Maciej Koutny; place: Pizzeria Francesca, Newcastle 2000)

For me, it was fairly clear that use of Petri nets was essential, both for the reasons of their wider acceptance by research community and also because I was really curious about certain relationships between Petri nets and state-based models. From the late 1980s I teased myself with the following problems: (a) what classes of Petri nets corresponded to the classes of distributive, semi-modular and speed-independent circuits (strictly speaking, lattices on cumulative states in terms of Muller theory), and in particular what were the conditions of Petri net transition labelling; and (b) what type of concurrency relations between events would lead to state-models with distributivity and semi-modularity. I managed to solve those problems around 1990 as I was leaving Russia and spent a year in Wales, after which I came back to Newcastle as a lecturer.

The solution to the first problem was that the class of safe and persistent nets with an injective labelling corresponded to distributive lattices, while k-bounded and persistent nets to semi-modular lattices. As for the second problem, the correspondence was reached

at the level of binary concurrency relations which could always be generalised to n-ary relations for distributive lattices. For example, if we had three actions, A, B and C, which were mutually pairwise concurrent, i.e. CONC={(A,B),(B,C),(A,C)}, then for a distributive process we could also generalise CONC relation to a 3-way concurrency tuple (A,B,C), whereas for a semi-modular yet non-distributive process the 3-way relation didn't hold. This was an interesting result which somehow led to the notion of resource-constrained behaviours. For example, in a non-distributive case, although actions could have been executed in parallel without any data dependency, if we had only two processors, we could not execute all three of them in parallel.

These results were later published in my papers of the early 1990s when I was in Newcastle. I also managed then to clarify and remove some modelling restrictions in Tam-Ahn Chu's STGs, and present a general unified STG model with the appropriate characterisations and relationships with state-based models and trace-theoretic models of delay-insensitive circuits (of the Dutch school at TU Eindhoven). In this work a great help came from my friend Luciano Lavagno, who was then finalising his PhD thesis at UC Berkeley under the supervision of Prof Alberto Sangiovanni-Vincentelli.

Later, together with Kishinevsky, Kondratyev and Lavagno we solved another interesting problem, namely that of how to unify Change Diagrams and Petri nets. We managed to identify the classes of Petri nets exactly matching Change Diagrams at the level of bijective labelling. We also showed that it was possible to unify and generalise these models by Causal Logic Nets, which finally raised the modelling power to that of Turing Machine and allowed the modelling of non-commutative state transition behaviour in a purely causal form. Some crucial proofs were done by Marta Koutny, who was then my PhD student! Marta co-authored our key paper on modelling OR-causality in Formal Methods in System Design. In all those developments I was also very happy to discuss ideas with Maciej, who gave his word of wisdom what concerned the semantics of concurrency and notions of steps in all our inter-modelling characterisations.

In those 1990s, in search of escape from semantical interleaving, a big push was also made on Petri net unfoldings. Firstly, it was through our work with my first PhD student at Newcastle Alex Semenov, with whom we developed many algorithms for the unfoldings of Petri nets, STGs, timed Petri nets, and Petri nets with read arcs. Semenov developed first unfolding-based software tools for verification and synthesis of asynchronous circuits – called PUNT. All this work was nicely reported in his thesis, which was successfully passed in front of Maciej, who acted as internal examiner, and Prof Steve Furber, who was external examiner. Unfortunately, this unfolding research was interrupted in 1997 when Alex Semenov decided to leave academia for work in the City of London. But, to our luck, this interruption wasn't for that long, as in 1999 Maciej accepted Victor Khomenko as his PhD student to work on unfoldings, and I gladly joined their team. Victor brought unfolding methods and tools based on them to another level of sophistication and computational power (in particular, an interesting result was obtained for merged processes which effectively compressed the branching process to a compact structurally cyclic, but semantically acyclic graph), and after a few years, new methods and tools for verification and synthesis of asynchronous logic appeared, which used unfoldings and SAT solvers. They were PUNF (a parallel unfolder) and MPSAT (unfolding-based verification and synthesis engine). Up till now they are being advanced and used within our Workcraft tool suite. Thus, true concurrency was elevated from its pure theoretical attraction to the level of a working instrument!

4 Synthesis of Petri nets and Petrify

Let's now get back to the year of 1994. That year was very important for the whole community of people surrounding me in promoting Petri nets as a key modelling language underlying asynchronous control logic design. Why? In that year, after the previous two years of establishing close collaboration with Kishinevsky, Kondratyev and Lavagno on the investigation of STGs as a theoretical and practical underpinning for asynchronous circuit design, all of us started to closely interact with Jordi Cortadella, a young professor from UPC in Barcelona. June 1994 was the time when Luciano and I visited Jordi before going to the Petri nets annual conference held in Zaragoza, where I presented our work on OR-causality models. In the same year, Mike Kishinevsky was a visiting EPSRC fellow in my group in Newcastle and in August 1994 we went to Windermere (Lake District) to participate in the first Amulet workshop organised by Prof Steve Furber. Prior to that trip we had an interesting chat with Maciej, who had for the first time mentioned to us the fact of an interesting theoretical characterisation of behavioural models of concurrent systems – Theory of Regions. He pointed us to the work of Nielsen, Rozenberg and Thiagarajian of 1992 (NPT92) on Elementary Transition Systems. 'Infected' with this fascinating read, we went to Windermere. While we interacted at length with the whole of Furber's group, 'infecting' them with the use of STGs in designing controllers for pipelines in the first Amulet processor (the PhD dissertation of Nigel Paver was very useful then!), our minds were captivated by this idea of Regions. It was so elegant and beautiful – Mike and I spent hours walking around our hotel at night discussing what would be its use in our more practical setting of asynchronous circuit design automation. Our first idea was that, with Regions we could take the model of a circuit in states, obtained for example, through the reachability analysis, and convert it into the model with true concurrency – say for the purposes of visualisation. A circuit with a state graph with thousand vertices could be shown nicely in a Petri net graph with some fifty vertices! Shortly after we got back from Windermere, we tried to formalise the synthesis of Petri nets and STGs from the state graphs through the notion of regions, by re-defining the key state and event separation axioms of the region theory in terms of excitation closure. That first draft was immediately sent to Jordi Cortadella, Alex Kondratyev and Luciano Lavagno. A very hot discussion of what can be done with all this new theory followed, and all of us had agreed that the visualisation was a good anecdote to promote it! Jordi, with his remarkable talent of making things happen in real and very fast, had managed, within a couple of weeks (!), to develop practical algorithms of constructing regions in the BDD framework. And that was virtually the birth of a new tool called Petrify. It is hard for me to remember who exactly came up with this funny name but we all loved it – as we were all petrified by its elegance!

So, the truly historic tool Petrify was born somewhere around September 1994. Maciej's suggestion to look at Regions and the NPT92 paper was monumental!

Shortly after, I managed to have solved an interesting challenge posed by Charlie Molnar to the asynchronous community, which was to design control logic for a counterflow pipeline processor (CFPP). Molnar's description of the control consisted of a 5-state diagram which intertwined concurrency and arbitrating choice in a very smart and compact interleaving. The puzzle was as to how to unravel that interleaving? Some solutions appeared in the community, but all of them departed from the original state graph into other interleaving-based models such as CSP and process algebra. I managed to find a way of performing an equivalent (very close to isomorphism) transformation on this state graph – by splitting a state and inserting a dummy event. And bingo, that gave me a way to cover

the 6-state graph with regions, and hence synthesize an equivalent Petri net, involving produce and consume arcs, as well as read-arcs. The rest was a piece of cake with my techniques for refining and converting the Petri net to an STG and then synthesise it to a control circuit. This CFPP example of using Regions and Petrify in synergy, not only for visualisation but circuit synthesis, had convinced me that we were on the right track (a paper was published in Formal Methods in System Design).

Then followed a sequence of developments in Petrify of various features, more related to asynchronous control logic synthesis, such as solving state coding problem, logic decomposition for hazard-free technology mapping, introduction of relative timing for timing optimisation, and numerous model conversions between state and event based representations. The paper of 1997 (Japan's IEICE Transactions on Information and Systems) about Petrify is one of the top-cited papers for all of us. Around 2000, Petrify and STGs, underpinning it, became a de facto standard technology in asynchronous circuit synthesis. People in academia and industry had used it with enthusiasm. In 2002 we published a book on STG based synthesis in Springer. In the same year we were nominated as finalists of the European Descartes Prize!

Meanwhile, the years 1999-2000, were significant for our Newcastle team, and largely thanks to our work on Unfoldings and Regions. In 2000, Marta defended her PhD thesis (with Dr Philippe Darondeau as external examiner) – which was largely about Regions. The thesis title was "Relating formal models of concurrency for the modelling of asynchronous digital hardware". Its particular, emphasis was on the class of semi-elementary transition systems and elementary net systems with inhibitor arcs (ENI-systems). The use of steps, a-priori and a-posteriori semantics in Marta's research helped me to understand better some intricacies of the modelling of concurrency in systems with inhibitors. The other important fact about that period was that in those days Maciej and I founded and actively promoted Asynchronous Systems Lab (ASL) and our ASL seminar, which had later gone through two decades of its successful run. ASL seminar series helped many of our RAs and PhDs to familiarise themselves with the fine grains of concurrency theory and its use in better understanding the behaviour of asynchronous circuits and systems. Causality and its various forms were essential in this process. These foundations helped building a new generation of theories of Petri net synthesis and tools for modelling electronic systems. Examples of such a wonderful synergy of theory and applications were methods and tools for visualisation of asynchronous circuits models, particularly in the PhD work of Victor Khomenko and Agnes Madalinski, synthesis of nets with policies in our work with Philippe Darondeau, modelling of GALS systems in the PhD work of Sohini Dasgupta and Johnson Fernandes.

Our constant interactions in the ASL community had led to other impressive results. They included methods and tools for direct mapping of Petri nets and STGs (recall David cells!), in the PhD work of Danil Sokolov, with active involvement of Dr Alex Bystrov who worked as an RA on a series of crucial EPSRC grants. Two EPSRC-sponsored and highly productive visits of Dr Nikolay Starodoubtsev to Newcastle had led to methods of synthesis of asynchronous control logic in negative gates and wire delay-insensitivity, all through finding elaborate ways of refining causality in STGs. This later helped our PhD student Yu Li to develop his method of refining STGs to lift the isochronic fork timing assumptions to much weaker timing assumptions. Work on Asynchronous Communication Mechanisms (aka ACMs), with active involvement of Dr Fei Xia and Dr Ian Clark, and our collaboration with Prof Tony Davies (from Kings College London) and Dr Hugo Simpson and Eric Campbell (from MBDA), helped Delong Shang, in his PhD work, to turn design methods for ACMs to real silicon chips. For example, our HADIC chip, fabricated in

0.6micron CMOS process in 2000, was the first harbinger in the series of chip tape-outs in the group, which were all strongly coupled to our research on causality and concurrency.

5 Tools, tools, tools: Interpreted Graph Models and Workcraft

In the late 1990s, I started to actively interact with Dr Oleg Mayevsky, my friend and former colleague from Varshavsky group. Oleg was an associate professor at the Kyrgyz-Slavic Russian University (KRSU), located in Bishkek, the capital of Kyrgyzstan, a relatively small country in Central Asia (a former Soviet republic). Thanks to the energy of Oleg's head of department Prof Gennady Desyatkov and Dr Chris Phillips, who acted as coordinator of EU-funded TEMPUS project between Newcastle and KRSU, aimed at developing a Master curriculum in Computer Science in KRSU, we had an excellent revival of the research collaboration with Oleg. The first result of this collaboration was the EPSRC-funded PhD work of Danil Sokolov between 2001 and 2004, which was linked to the EPSRC BESST project on asynchronous control logic synthesis from Petri nets.

Fig. 3. ASL is conquering the Great Wall with Concurrency and Causality (Maciej, Marta, Ola, Maria and Alex; trip to China, 2006)

Around 2002, thanks to the ASL funding, we had Oleg Mayevsky at Newcastle as a senior RA for six months, and productive work with David Kinniment, Gordon Russell, Alex Bystrov and myself on time measurement and time-to-digital converters (a famous Time Amplifier was invented then!). Subsequently, in 2005, when I had another vacant PhD studentship from EPSRC, linked to a project on networks on chip, I was fortunate to be introduced by Mayevsky and Desyatkov to Andrey Mokhov, who was finishing his MSc degree at KRSU and had just been amongst the finalists of the World Programming Contest. The arrival of Andrey in Newcastle in September 2005 had given a massive boost in our ASL team as he was able to inject his great potential and enthusiasm into the group. In his research he took the model of partial orders and added there conditionals to capture

choice, produce an entirely new model Conditional Partial Order Graphs (CPOGs), which enabled a whole range of new developments, from the models of communication channels based on phase encoding (PhD of Enzo d'Alessandro), to control of microarchitecture in CPUs (PhD of Max Rykunov), and to more general graph algebraic approaches (promoted by Andrey himself) in wide variety of applications, requiring compact modelling of multitudes of scenarios.

With Andrey's PhD on CPOGs completed in 2009, and PhD work of another KRSU alumni, Ivan Poliakov, who started work on unifying our graph based models, such as Petri nets, Circuit Petri nets, STGs, CPOGs and data-flow structure under the new tool set called Workcraft. Ivan's work was linked to the EPSRC-funded project SEDATE, which launched our collaboration with the Manchester CAD group led by Dr Doug Edwards. The aim of the project was to develop methods and tools for synthesis of asynchronous data-paths. In this work we actively promoted our approach to modelling strong (AND) and weak (OR) causality, previously used for control models such as STGs to static data-flow structures (SDFS). Here, it was possible to relate these causality paradigms with the data-path notions of full acknowledgement and early propagation, respectively. Notions of forward data tokens and anti-tokens, expressed in the work of others, were unified by our Petri net based interpretations. Danil and Andrey were closely working with Ivan, as well as we had fruitful interactions with our PhD student Zhou Yu and Will Toms (from Manchester). Accompanied by the PhD work of Julian Murphy and Ashur Rafiev (also from KRSU), on exploiting causality and concurrency in circuits for cryptographic applications, this gave the group a great pathway to impact – in which we worked with Atmel Smart card ICs, where our tools (largely thanks to Danil's efforts) were used in mid 2000s.

The significance of the transition to the toolset of Workcraft, around 2007, is hard to overestimate. Our previous tools were all command-driven and based on textual representation of state and event based models, where the key format was that of a "dot-g" (we often pronounced it 'dodgy'!) file. Workcraft stipulated that the tools had to be linked to an interactive GUI-based framework, to which other problem-oriented tools, such as Petrify and MPSAT, could be connected as backends or plugins. That view, promoted in Ivan's thesis, and later driven by the developments of new tools by PhD students Arseniy Alekseyev (again, from KRSU!), and Stas Golubcovs, coordinated by Danil Sokolov, has resulted in a powerful CAD environment, which has by now, the year 2018, given rise to the new evolution step thanks to the efforts of Danil, Victor and Andrey.

All this work is now at an entirely new level of maturity and no wonder it has been noticed again (after our prior experience with Atmel) by industry. In 2014, we launched a fruitful collaboration with Dialog Semiconductor, a company specialising in developing IP solutions for electronic power management and communications. This is the area of analogue-mixed-signal (AMS) electronics, in which the significant role is played by islands of embedded digital logic. We gave this logic the name of 'little digital' on the analogy of the digital logic that was of modest complexity in the era before VLSI design. Then in the 50s and 60s, that digital design was largely clock-free, i.e. asynchronous. Normally AMS designers would design such logic by hand and then include its components into the overall process of painstaking simulations – which would last for days if not weeks. With a bit of destiny turn, in 2013, I was approached by Dr David Lloyd, from Dialog, who used to be a member of the Amulet group in Manchester and was interested in asynchronous design. David wanted to try our Petrify tool in designing control logic for power regulators. We started active collaboration which was partly initially sponsored by EPSRC in the A4A (Async for Analog) grant. The prospect of automating the design of little digital was very attractive.

At the moment this industrial take-up is strong and what's important, today we are teaching practical engineers to think in terms of causality and concurrency, and they can immediately see these concepts in action – all through the Workcraft tools – they can run extensive simulations, they can verify circuits and their specifications, they can synthesise and decompose their circuits, and they can visualise the behaviour the models using such representations as waveforms. For example, Jonny Beaumont's recent PhD work was about Concepts – these are elementary building blocks of the designer's knowledge about the asynchronous circuit behaviour. They are captured and can be converted to synthesisable specifications.

I have not mentioned many other developments within Workcraft in which Petri nets play the behaviour-interpreting role to capture the semantics of the specific instrumental models. These include the models of structured occurrence nets (SONs), which have been actively investigated by Maciej, Brian Randell and their ex-PhD student Bowen Li and applied in the application domain of crime investigations, models of the communication fabrics of xMAS for networks on chip and GALS systems, developed by Dr Frank Burns, models of instruction set architectures, developed in the PhD theses Alessandro De Gennaro and Georgy Lukianov, and other models. We are also advancing STGs into new applications for AMS through PhD work of Vladimir Dubikhin, who combines our Workcraft tool with the analogue verification methods based on Labelled Petri nets and tool LEMA of Prof Chris Myers (University of Utah) to produce a new design flow for AMS with asynchronous control (behaviour mining and model generation are the core of his method). Further to that, work of Sergey Mileiko is aimed on applying STG based methods to design controllers for Switched Capacitor Converters, in collaboration with Alex Kushnerov (graduate of Ben-Gurion University and now working at Intel, Israel).

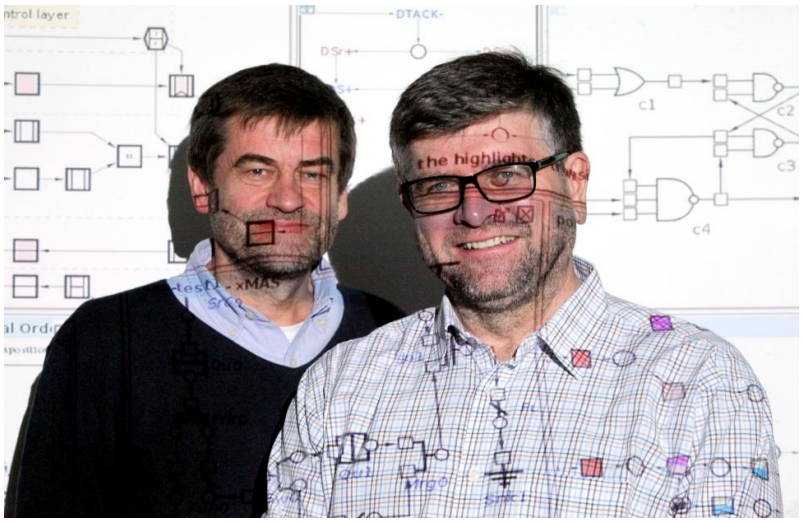


Fig. 4. Tools, only tools, nothing but tools (our REF-2014 Impact showcase; Newcastle 2015)

All these models have distinct elements of Maciej's fundamental notions of the semantics of concurrency and causal representations based on graphs, showing causality through the relations of precedence, control and data dependency, timing relations.

6 What is actually Causality?

So far, I have been addressing the notion of causality in the framework of event-based models, namely Petri nets. The relationship between events (transitions or their occurrences) in Petri nets is based on temporal aspect, i.e. event A precedes event B. This precedence relation is captured by an arc in a graph, and it is sometimes important to show the dynamics of this relation using the notion of an explicit condition (place) which can have a token or not have it. The idea of a token is important to be able to interpret the behaviour in the form of states of the system (token marking). If a condition between two events A and B is marked with a token the causal dependence is active, meaning that the event A has occurred and may lead to the occurrence of B at any moment, possibly conditioned by other causes of B incident on transition B.

The notion of causality based on precedence is probably best captured by models such as occurrence nets or event structures, which are acyclic and show the causal relationships purely in the form of directed arcs.

People studying causality in different forms and application domains may give it other interpretations and emphasis, which may not be directly linked to the temporal aspect. However, very often the temporal aspect, or the notion of precedence, is implied there. For example, the work of Professor Judea Pearl on causality modelling is more of the probabilistic nature, where causation between facts can be implied by some forms of high likelihood or probability of something to be true based on the fact of something else being true. This form of causation associated with knowledge or facts thereof can often be cast on the timeline. Therefore, Pearl's models can be translated to say SONs or event structures. In similar vein, of more cognitive nature, one can see the works of other researchers of causality, such for example as Steven Sloman (see next section).

My interest in causality is nowadays extending from cognitive to physical nature. I have recently become interested in electromagnetism and its relation to causality. Here one important element is the idea of energy flow, i.e. the transfer of energy in space and in time and its effects on the matter. An example of a physical causal system could be a system consisting of energy storage, such as a capacitor, and a load, a switching circuit whose power signal is connected to the capacitor. Here clearly we cannot obtain any switching behaviour if the voltage level on the capacitor is not sufficient to overcome the threshold voltage of the switching device in the load.

Further thoughts in physical direction bring me to the connections between types of logical (AND, OR) causality at the event level and notions of integration, differentiation and proportionality, typically used in automatic control. Indeed, think about PID control and relations between cause(s) and effect(s). Fig. 5 illustrates this connection. AND causality is linked to the integration paradigm, which is a positive hysteresis – we accumulate enough of the increasing causes before we see the effect in the rising edge of the signal. Similarly, we need to accumulate the decreasing causes before the falling edge of the effect. OR causality is linked to the differentiation paradigm, which is a negative hysteresis – we react to the effects early, for both the rising edge and the falling edge. The third case, of combinational dependence, has no hysteresis at all – it corresponds to the proportional paradigm.

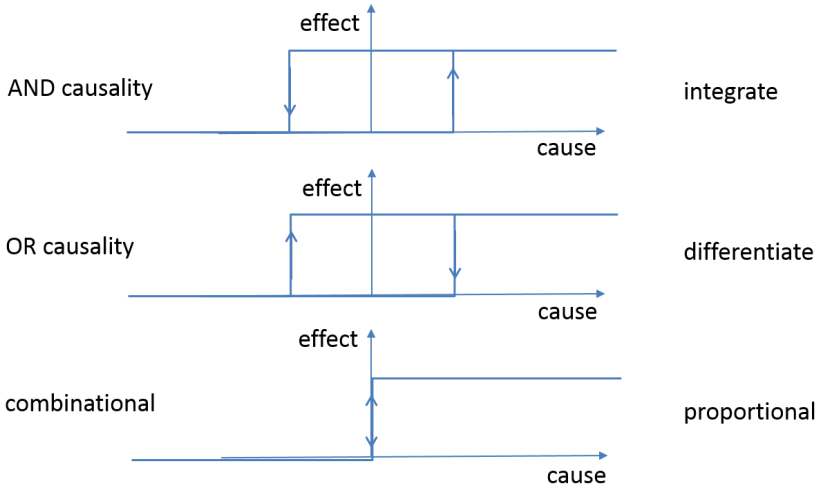


Fig. 5. Relationship between types of causality and PID control

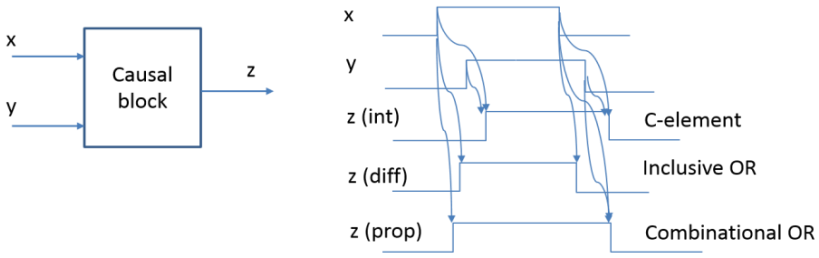


Fig. 6. Example of causality and PID forms for the case of two causes x and y and effect z.

All these paradigms can be implemented using logic circuits such as C-element (Integrate), inclusive OR (differentiate) and combinational OR (proportional) as illustrated in Fig. 6.

So, what is Causality after all (or better say before all!)? This question is probably equivalent to what is Time?

Steven Sloman in his book on Causal Models (published by Oxford University Press, 2005) refers to Bertrand Russell, who argued in his famous article "On the notion of cause" (1913) that "invariant laws aren't causal at all but rather mathematical". Sloman then writes: "Perhaps the world is nothing but a flow of energy … Perhaps, everything we misconstrue as cause and effect is just energy flow directed by mathematical relations that have determined the course of history and will determine our destiny in a long chain of events linked by the structure of energy in time and space." He then makes a sort of disclaimer that his book "isn't about metaphysics. It's about representation".

That's exactly the Gordian knot of the question of what Causality is. I agree with Sloman in that Causality has place in representation and hence plays a key instrumental role in cognitive science – that's what we have been exploring with Maciej in our research in concurrency. However, I tend to disagree that Causality has no physical meaning. Even if the world is modulated by the flow of energy, as Oliver Heaviside called "energy current"

in his Electrical Papers (Vol.2, pp.91-95), this is physics and it has causal meaning. Not all "invariant laws" aren't causal! If we go back to the work of Galileo and Newton, we can find important geometric proportionality between energy and momentum (associated with mass) via a spatiotemporal coefficient that is velocity. It is important that this proportionality is geometric – this makes relation between energy as cause and momentum as effect physical! As formulated by Heaviside for electromagnetism, we have this proportionality between energy current (aka Poynting vector) and electromagnetic fields mediated by the velocity of light, and this is also causal. There are of course some relationships in physics which are, quite mistakenly, interpreted as causal, which are in fact purely algebraic – such for example as the relationship between electric and magnetic field. In other words, when one looks at equations in physics one has to distinguish geometric (physical) proportionality from algebraic (mathematical). Some philosophy researchers, e.g. Ed Dellian, even put the names of the great scientists at this boundary – Newton at the former and Leibniz at the latter.

In my opinion, the physical world in which everything is governed by the energy current does not stop being Causal. Causality is not only representational – it is fundamental! On this highly optimistic point I should perhaps stop.

7 Instead of conclusion

The last word here will however have to be with classic philosophers …

PROPOSITION 28 in Part I of Spinoza's Ethics says: "Every individual, or each thing which is finite and has a determined existence, is not able to exist or to be determined to act unless it is determined to exist and act by another cause which is also finite and which has a determined existence: and again, this further cause is not able to exist or to be determined to act unless it is determined to exist and act by another which is again finite and has a determined existence, and so on to infinity."'

Happy Birthday, Dear Maciej!!!

Benefit and Cost of Cloud Computing Security

Wen Zeng and Vasileios Germanos

School of Computer Science and Informatics
De Montfort University, LE1 9BH, U.K.
wen.zeng.wz@gmail.com, vasileios.germanos@gmail.com

Abstract. Running information security technologies and policies in cloud computing systems will create negative impact to business organizations. Chief information security officers, as potential users, need a solution to analyze the cost and benefit of implementing information security technologies and policies in cloud computing systems. Petri nets can be used to analyze the workflow security in cloud computing systems, diagnose possible malicious behavior, and analyze invisible executions and failures in such systems.

Keywords: Petri nets · security · cloud computing · opacity.

1 Introduction

The importance of cloud computing is increasing due to the ever-increasing demand for internet services and communications. However, the large number of services and data stored in the clouds creates security risks due to the dynamic movement of data, connected devices, and users among various cloud environments. Information security technologies and policies are developed to keep the data and information secure in the systems. However, the implementation of these technologies and policies in cloud computing systems will create negative impact to the business organizations. For example, increasing the complexity of services in the system will increase the rate of failure. Therefore, it is important to evaluate the cost and benefit of implementing information security technologies and policies in cloud computing systems.

2 Cloud Computing Security

In 1995, Bill Gates wrote a memo entitled "The internet tidal wave" which described how the internet was going to forever change the landscape of computing [1]. He indicated that the rich foundation of the internet will unleash a services wave of applications available instantly over the internet to millions of users. Services designed to scale to hundreds of millions will change the nature and cost of software solutions. Now all his predictions became reality.

Cloud computing is a type of internet-based computing which provides shared computing resources and data to computers and other devices on demand.

Instead of building individual information technology infrastructures to host databases or software, a third party can host them in its large server clouds.

Cloud computing platforms are designed to automatically replicate data across a worldwide infrastructure. Therefore, the cloud providers can dynamically engage more resources, such as, servers and storage, as your site needs them. However, security is the most significant barrier to widespread adoption of cloud computing.

Firstly, although cloud computing services are relatively new, data breaches in all forms have existed for years [2]. Over 50 percent of the IT and security professionals believed their organization's security measures to protect data on cloud services are low [2]. Data breaching was three times more likely to occur for businesses that utilize the cloud than those that do not [2].

Secondly, the cloud's unprecedented storage capacity has allowed both hackers and authorized users to easily host and spread malware, illegal software, and other digital properties [2]. Consequently, this practice affects both the cloud service providers and their clients [2].

Thirdly, application programming interfaces give companies the ability to customize features of their cloud services to fit business needs; however, this practice can potentially leave exploitable security risks [2]. The vulnerability of application programming interfaces lies in the communication that takes place among applications [2]. A simple example is YouTube, where developers have the ability to integrate YouTube videos into their sites or applications [2].

3 Information Security Technologies and Policies

Information security technologies have been investigated and developed to keep the data and information secure in cloud computing systems.

Large IT companies, like Microsoft, IBM and Google, provide authentication technologies that only authorized users can access data resources and applications in the clouds. For example: Microsoft Azure develops Azure active directory to ensure that only authorized users can access clients' environments, data and applications [3]. IBM provides identity and access management capabilities designed to strengthen compliance management and reduce risk in today's cloud environments [4]. IBM Cloud lets users build authentication and authorization into users' cloud-native apps, and manage access to cloud resources [4]. Google Cloud Identity offers the identity services and endpoint administration that are available in G Suite as a stand-alone product [5].

Protocols are used to keep the data secure in the clouds. For example: Microsoft Azure uses industry standard protocols to encrypt data in transit. These secure users' data as their travel between devices and Microsoft datacenters [3]. IBM Cloud also provides encryption of data at rest and while data are moving across storage and data services, along with a key management service [4]. Google Cloud Platform, by default, encrypts customer data stored at rest, without requiring any additional action from its users [5].

Security analytics technologies are developed to monitor servers, networks and applications in order to detect and respond to threats in the clouds. For example: Microsoft Azure multipronged threat-management approach includes technologies and processes to constantly strengthen Azure's defenses and reduce risks [3]. IBM Cloud provides a main set of network segmentation and network security services to secure workloads from network threats [4]. IBM Cloud has built in features that allow users proactively monitor and gain security intelligence across users' hybrid cloud deployments [4]. Users, based on security analytics, can detect and respond to threats quicker, dramatically accelerate investigation times and proactively manage compliance [4]. Google Cloud Security Scanner is a web security scanner for common vulnerabilities in Google App Engine application, which can automatically scan and detect common vulnerabilities [5].

One would notice that cloud leading companies are focusing on security technologies, data and application security in cloud computing systems. The cost and benefit of executing these technologies and policies on cloud computing have not been analyzed. Digital Institute Newcastle proposed [6] cost models for multi-clouds systems, which considered the access control policies on the multi-clouds. However, trade-off between running access control policies and potential cost in the multi-clouds has not been considered. There are some cloud comparison website companies (e.g., CloudOrado.com) that provide the information about basic security policies/services/certifications of each cloud provider (e.g., encrypted storage, ISO/IEC 27001, and Firewall). However, the security and cost of workflow deployment in these multi-clouds, and security metrics of the workflow have not been considered.

In addition, there has not been an established systematic approach to quantitatively evaluate the project economics of information security technologies in cloud computing systems, including their effectiveness, benefits and costs to organizations, although information security researchers proposed various methods for addressing security investment problems. As a top security expert, Ross Anderson at Cambridge University pointed out security researchers should consider the costs of subsequent maintenance of the software. For example: security protocols failures may be explained by economics. Especially, the failures of large systems would cost industry billions.

4 Benefit and Cost Analysis

There are several different types of clouds which the clients can operate with depending on their business model. Public clouds are the most commonly used; their resources are made available to the general public by a particular provider, such as Microsoft, IBM or Google. On the other hand, private clouds are operated for a single organization, which can be managed either internally or by a third party, and can be hosted either internally or externally. Private clouds require a significant amount of engagement and re-evaluation of business strategy to be used effectively. However, large organizations may wish to keep sensitive

information on their more restricted servers, rather than in the public cloud. This has led to the introduction of federated cloud computing, in which both public and private cloud computing resources are used [6].

Information flow refers to paths followed by data from their original positions to the end users in computational processes [7]. Workflows are used to specify the implementation of such processes [7]. Information flow security in software engineering becomes an active topic, due to access control cannot control internal information propagation once accessed. Information flow security is to ensure that the information propagates throughout the execution environment will not leak sensitive information to public. The large number of services and data on a cloud system creates security risks, due to the dynamic movement of the services and data resources on the cloud systems [7].

In our study, we introduce security lattices for the components of a federated cloud system. Moreover, we assign security levels to individual services, data resources and clouds, and sign clearance level to individual services. We also assign clouds to individual services and data resources.

Then, we adopt the Bell-LaPadula multi-level control model into cloud computing systems [6]. A service can only operate at a security level that is less than or equal to its clearance [6, 7]. A service cannot read data that are at a higher security level than its own clearance [6, 7]. A service cannot write data residing at a lower security level [6, 7]. If an entity is located in a cloud, the security level of the cloud must be higher or equal to the security level of the services and data resources [6, 7].

4.1 Cloud Computing Trade-offs

After we adopt Bell-Lapadula multi-level control model into the systems, we will have different valid mapping options of services and data resources to clouds [6]. In some cases, the performance and cost of private and public clouds are different. The data size, the length of time the data are stored in the clouds also affect the cost of the different options. CPU consumed in the execution of the services in different clouds affects the cost of the systems. Furthermore, the complexity of services in the system will affect system failure rate.

There are two parts we need to consider for analyzing the benefit and cost of implementing information security technologies and policies in cloud computing systems: 1) The value or benefit that information security technologies or policies can bring to the organizations; 2) the cost and impact that information security technologies and policies have on organizations.

Benefit Information is treated as a business asset with varying levels of commercial values as introduced by [8]. Since it is usually difficult to measure the benefits of information security technologies directly, costs of information disclosure or modification are measured instead to quantify the benefits. Therefore, the organization's digital information assets should be evaluated and classified. Information should be classified into different categories by their level of confidentiality according to company policies. Then, they are assigned estimated

values to digital information of each category. This information valuation process should be based on the business needs of the organization [8].

For a cloud computing system, observing patterns of users' behaviour can lead to leakages of secure information. Information sharing means that the behaviour of one cloud user may appear visible to other cloud users or adversaries, and observations of such behaviour can potentially help them to build covert channels [9, 10]. We consider using opacity as a promising technique for analysing information flow security. Opacity has been proposed as a uniform approach for describing security properties of computing systems expressed as predicates [9, 10]. A predicate is opaque if an observer of the system is unable to determine the truth of the predicate in a given system run [9, 10].

In a cloud computing system, the security policy is the architecture with public/private clouds, yielding an observation function, and a predicate that needs to remain opaque [10]. The cost of information leakage is based on the assumption that there is a security policy and a competing cost of opacity. The cost of the security policy is the financial cost of setting up and using a particular architecture. The cost of opacity is the financial cost of leaking confidential information that should remain hidden. The security policies play a key role in determining the best design of the system. In addition, the security policies also influence the opacity of the systems.

Due to the complexity of cloud computing systems and data distribution on them, the risk of information loss is high. Cloud providers should be able to provide techniques for detecting potential invisible malicious events in the systems. These techniques can increase the dependability of their services, and consequently organizations will feel more confident to use them. Thus, invisible malicious events, which violate the proposed security policies in cloud computing systems, must be detected. Diagnosis is the procedure of detecting abnormal behaviours of a system. A notion of diagnosability [17, 18] - an associated property of diagnosis - can been used to detect such events in clouds.

Cost Cloud costs are measured using the metrics by which cloud providers allocate charges and impact of the allocated strategies [6], which are affected by information security technologies and policies. Some costs are tangible, for example, the capital expenditures on information security technologies and associated hardware, software and daily operational expenditures associated with maintenance. These tangible costs are readily to be accounted for in monetary terms. There are also some intangible costs. For example, it has been digital documented that implementing a strict security mechanism will reduce the efficiency of the organization [11]. One of the impacts of information security technologies on staff productivity is quantified in terms of non-productive time [11].

In our study, the tangible costs of a cloud computing system include: purchase of security technologies/policies and any associated costs to upgrade hardware and software, the cost of the data stored in the clouds, the cost of CUP consumed in the execution of services, and data transfer costs [6].

The intangible costs of a cloud computing system include: employee non-productive loss, administrator costs, and training employees.

Non-productive time [11] incurred as the result of the reduction in organization efficiency after the implementation of cloud computing security technologies and policies. The more services and security technologies are deployed on a system, the more complex the system will be. In general, increasing the complexity of services in the system will increase the rate of failure. System failure will affect the productivity of the organization. For many projects, this is a conservative estimate, because of high daily operational rates. For example, the daily rates of drilling rigs in oil and gas industry.

Administrators have to be hired to handle the cloud computing systems and the requests in the organization.

Cost of training and education of employees. It has been reported that most security incidents are caused by human errors instead of technology failures, although security incidents can result from nature disasters, technical issues and human acts. Therefore, the organization should provide information security training to employees.

5 Quantitative Evaluation Methodologies

We consider that Petri nets and the associated verification techniques can be used to analyze the cloud computing security.

Petri nets are a graphical modelling tool for a formal description of systems whose dynamics are characterized by concurrency, synchronization, mutual exclusion and conflict [12]. In particular, they have been widely used for structural modelling of workflows and have been applied in a wide range of qualitative and quantitative analysis [12, 13].

A basic Petri net consists of places, transitions, a set of arcs and the initial marking. Places represent possible states of the system, transitions are events or actions which cause the change of state, and every arc simply connects a place with a transition or a transition with a place. A change of state is denoted by consuming/producing tokens (black dots) from places to places, and is caused by the firing of a transition. The firing represents an occurrence of the event or an action taken. The firing is subject to the input conditions, denoted by token availability. A transition is firable or enabled when there are sufficient tokens in its input places. After firing, tokens will be consumed from the input places, and be produced to the output places. For example: in [7], Petri nets are introduced to diagnose the possible malicious behaviour in the cloud computing systems. In [9, 10], Petri nets are used to capture the invisible workflow in cloud computing systems.

Coloured Petri nets allow the modeler to use a number of different colour sets, making it possible to represent data values in a more intuitive way instead of having to encode all data into a single shared set [14]. Coloured Petri nets introduce a set of coloured tokens that can be distinguished from one another, unlike the indistinguishable black tokens in the basic Petri nets, and use arc

expressions to define how transitions can occur in different ways depending on the colours of input and output tokens. Coloured Petri nets can be used to capture the security policies in cloud computing systems. In [7], coloured Petri nets are used to capture the Bell-LaPadula rules and cloud security rules. The security policies for services migration and data migration can be captured by the functions associated with transitions.

Stochastic Petri nets are an extension of classic Petri nets [15], and Stochastic Activity Networks are a class of stochastic Petri nets [16]. One can associate a firing delay with each transition of a Petri net; such a delay specifies the time that the transition has to be enabled before it can actually fire. If the delays are given by a random distribution function, we obtain a stochastic Petri net. A stochastic Petri net model includes a set of ordered activities to be undertaken by humans or other resources of the organization or a system. A stochastic Petri net model is a structure for actions and implies on how work is done within an organization or a system. These actions are work activities across time and space, with a beginning and an ending. In [11], non-productive time associated with information security technologies are captured by the firing delay of the transitions, and the system failure rate can be captured by the probability associated with the transitions.

References

1. B., Gates.: The Internet Tidal Wave, http://www.lettersofnote.com/2011/07/internet-tidal-wave.html. Last accessed 1 Jun 2018
2. J., Ma.: Top 10 security concerns for cloud-based services, https://www.incapsula.com/blog/top-10-cloud-security-concerns.html. Last accessed 1 Jun 2018
3. Microsoft.: Microsoft Azure, https://www.microsoft.com/en-us/trustcenter/security/azure-security. Last accessed 1 Jun 2018
4. IBM.: Security in the IBM Cloud, https://www.ibm.com/cloud/security. Last accessed 1 Jun 2018
5. Google.: Trust & Security, https://cloud.google.com/security/. Last accessed 1 Jun 2018
6. Watson, P.: A Multi-Level Security Model for Partitioning Workflows Over Federated Clouds. Journal of Cloud Computing **1**(1), 15 (2012)
7. Zeng, W., Koutny, M., Watson, P., Germanos, V.: Formal Verification of Secure Information Flow in Cloud Computing. Journal of Information Security and Applications **27-28**, (2016)
8. Humphreys, E.: Information Security Risk Management. British Standards Institution, (2010)
9. Bryans, J., Koutny, M., Mazare, L., Ryan, P.: Opacity Generalised to Transition Systems. International Journal of Information Security **7**(6), 421–435, (2008)
10. Zeng, W., Koutny, M., Watson, P.: Opacity in Internet of Things with Cloud Computing. In: 8th IEEE International Conference on Service-Oriented Computing and Application, pp. 201–2017, Rome, Italy, (2015)
11. Zeng, W., van Moorsel, A.: Quantitative Evaluation of Enterprise DRM Technology. In: Electronic Notes in Theoretical Computer Science, (2011)
12. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE **77**(4), 541–580 (1989)

13. v. d. Aalst, W.: The application of Petri Nets to Workflow Management. Journal of Circuits, Systems and Computers **8**(1) 21–66 (1998)
14. Jensen, K.: Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. 2 edn. Springer-Verlag Berlin Heidelberg (1996)
15. Marsan, M. A., Balbo, G., Conte G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets. 1st edn. John Wiley & Sons, Inc., New York (1994)
16. Sander, W.: Construction and Solution of Performability Models Based On Stochastic Activity Networks. Ph.D. Thesis, The University of Michigan (1998)
17. Germanos, V., Haar, S., Khomenko, V., Schwoon, S.: Diagnosability under Weak Fairness. ACM Trans. Embed. Comput. Syst. **14**(4) 1–19 (2015)
18. Bérard, B., Haar, S., Schmitz, S., Schwoon, S.: The Complexity of Diagnosability and Opacity Verification for Petri Nets. In: van der Aalst, W., Best, E. Application and Theory of Petri Nets and Concurrency - 38th International Conference, PETRI NETS, vol. 10258, pp. 200-220. Springer (2017)