Causality, Behavioural Equivalences, and the Security of Cyberphysical Systems

Sibylle Fröschle(✉)

OFFIS and University of Oldenburg, 26121 Oldenburg, Germany
froeschle@informatik.uni-oldenburg.de

Abstract. The large cyberphysical systems that are currently being developed such as Car2X come with sophisticated security architectures that involve a complex interplay of security protocols and security APIs. Although formal methods for security protocols have achieved a mature stage there are still many challenges left. One is to improve the verification of equivalence-based security properties. A second challenge is the compositionality problem: how can the security of a composition of security protocols and APIs be derived from the security of its components. It seems intuitively clear that foundational results on causal equivalences and process calculi could help in this situation. In this talk we first identify four ways to exploit causality in security verification. In particular, this will lead us to review results on causal equivalences. Finally, we discuss how such results could help us to tackle the two challenges.

1 Motivation

Cyberphysical systems such as Car2X are potentially vulnerable against attacks that could have a drastic impact on the safety as well as the privacy of their users. Therefore such systems must be protected by a sophisticated security architecture. Take Car2X as an example. Based on a threat and risk analysis, the ETSI[1] standards advocate a security architecture that includes authenticated Car2X communication by digital signatures, cryptographic keys and credentials management, privacy enabling technologies by pseudonyms, and in-car software and hardware security based on hardware security modules (HSMs) [11,12].

Two mechanisms are central within such a security architecture: *security protocols* and *security APIs*. A security protocol specifies an exchange of cryptographic messages between two or several principals, intended to achieve security objectives such as authentication, key establishment, or confidentiality of data. A security API (Application Programming Interface) is the software interface to a security services layer. At the lowest level this will typically be the API to an HSM that stores and uses sensitive cryptographic keys.

This work is partially supported by the *Niedersächsisches Vorab* of the Volkswagen Foundation and the Ministry of Science and Culture of Lower Saxony as part of the *Interdisciplinary Research Center on Critical Systems Engineering for Socio-Technical Systems.*

[1] European Telecommunications Standards Institute: http://www.etsi.org.

R. Meyer et al. (Eds.): Olderog-Festschrift, LNCS 9360, pp. 83–98, 2015.
DOI: 10.1007/978-3-319-23506-6_8

The security properties that are central for the verification of security protocols and APIs fall into two categories: one is that of *reachability-based* (or *trace-based*) properties, the second is the class of *equivalence-based* properties. Traditional properties such as authentication and syntactic secrecy are trace-based properties. They express properties of protocol runs: if A and B exchange a secret s and during no run of the protocol the attacker can obtain the value s then the protocol satisfies syntactic secrecy of s. In contrast, privacy-type properties such as untraceability, vote secrecy, or anonymity have to be expressed in terms of indistinguishability: if an attacker has no way to distinguish the process in which A votes 'yes' from the process in which she votes 'no' then vote secrecy is satisfied. Formally, indistinguishability is expressed in terms of a notion of behavioural equivalence.

The verification of security protocols has reached a mature state with many tools available that can automatically check whether a security property is satisfied (up to certain assumptions or abstractions) (e.g. [46]). However, there are still many challenges left. Decidability and complexity results as well as automatic tools mainly target reachability-based properties so far. Hence, one challenge is to improve the foundations and verification techniques for equivalence-based properties. Only few results on the applied equivalences are known (c.f. [6]). The tool ProVerif does support the verification of privacy-type properties but it does so by an ad hoc encoding of the situation when processes differ only in their choice of some terms [4].

A second challenge concerns compositionality. Most of the formal methods and automatic tools are only capable of checking one protocol at a time. It is folklore that as long as two protocols are disjoint in that they do not share any data their composition is secure iff each protocol is secure in isolation. However, as exemplified by real security architectures such as that of Car2X this situation is far from reality. A stack of different protocols and APIs is necessary exactly because there are different interconnected phases such as key management on a server, key establishment between a server and a principal, and exchange of confidential information between them. Another problem is that an attacker might deliberately induce that protocols share a key they are not supposed to share, or this might be induced by users who use the same passwords in different situations. For a summary of works that already address the compositionality problem see [2,7].

In this talk we explore how causality as a general theme, and causal equivalences from 'pure' concurrency theory can help us in the verification of complex security architectures. We proceed as follows. In Section 2 we review four ways to exploit causality in security verification, and note that causality has mainly been applied to reachability-based verification problems. We identify that the fourth way to exploit causality could be very relevant for equivalence-based properties. The idea is to apply and lift a positive trend for causal equivalences from concurrency theory to equivalence-based verification. Thereby motivated, in Section 3 we take a closer look at these causal equivalences. We give an overview of known decidability and complexity results, and identify some open problems relevant

for their application. Finally, in Section 4 we discuss what it takes to lift this trend into an applied setting, and close with some general remarks.

2 Four Ways to Exploit Causality

2.1 Modelling

1. Modelling: "A causal model says more than a million transitions."

Transition systems are the natural model when it comes to automatic state space explorations. But when it comes to modelling or analysing a system by hand then models that faithfully represent the causal structure of the system are usually the model of choice. The reason for this success of causal models is twofold: they avoid the state explosion problem by modelling concurrency explicitly; and, they typically come with an intuitive graphical notation. So just as "a picture says more than a thousand words" "a causal model says more than a million transitions".

In security verification this is exemplified by the *strand space model*, which is *the* causal model for security protocols. The strand space model was introduced by Thayer, Herzog, and Guttman in their paper "Strand Spaces: Why is a Security Protocol Correct?" [13] as a special-purpose model that allows one to develop correctness proofs by hand. To use the author's own words, it is "distinguished from other work on protocol verification by the simplicity of the model and the ease of producing intelligible and reliable proofs of protocol correctness even without automated support".

2.2 Verification

2. Verification: "Refute that an attack exists by tracing all possible causal constellations to a contradiction."

Everybody has a notion that some event A is a cause of another event B. And that if event A hadn't happened then event B wouldn't have happened either. This translates into a natural proof principle of backwards analysis. Say we wish to show that a bad event B cannot happen. To the contrary assume that B has occurred and analyse what must have happened beforehand. If we can lead all possible causal constellations to a contradiction then we can conclude that our assumption was wrong and B can indeed *not* happen.

This is the proof method that is originally associated with the strand space model [13]. The original manual proof method also sparked off several generations of semi-automated tools which work by backwards search. Among the earliest is the semi-automated tool Athena by Song [47], which translates the backwards reasoning style into a backwards search algorithm. The Athena tool has in turn influenced many of the more recent ones: Cremer's tool Scyther [8], which has recently been used to analyse the large Internet protocol IPSec by using supercomputer power [9], and the successor tool Tamarin [46].

The proof principle has also been employed in the verification of security APIs. The strand space method has inspired our backwards reasoning approach to security API verification [18,23], including case studies of the standard PKCS#11. Moreover, an approach for automated backward analysis for a class of PKCS# 11 configurations as been presented in [35]: by translation into the Tamarin prover. Finally, the strand space method has also been used to tackle the compositionality problem: Among the recent proof methods by hand there is Lowe et al.'s approach to verify layered security architectures including a case study of TLS [33].

2.3 Decidability and Complexity I

3. Decidability and Complexity I: "Exploit causal structure to reduce a search space of attacks to a decidable search space of 'well-structured' attacks."

If we work with a model that faithfully represents the causal structure of a system, we may be able to exploit this extra structure to obtain results we may not be able to formulate and prove otherwise. An explicit representation of causal dependencies gives us concepts at hand such as the causal shape or the causal depth of attacks. Moreover, if we can show that there is an attack iff there is one with a particularly 'good and regular' causal structure then we may be able to reduce our search for attacks to a search for more manageable 'well-structured attacks', and thereby obtain new decidability and complexity results.

We have used this principle to prove that REACHABILITY is decidable in NEXPTIME for protocols with disequality constraints and bounded message size [17]; and more recently, that LEAKINESS is decidable for well-founded protocols [19]. (Leakiness is a type of secrecy that does not admit temporary secrets.)

Well-founded protocols strictly contain a group of protocols that impose conditions that make encrypted messages context-explicit [5,37,43–45]. The idea is that such protocols merely satisfy the prudent engineering practice recommended by Abadi and Needham [1]. For example, a 'light' way to achieve context-explicitness is to tag protocols by introducing a constant into each encryption, and thereby to uniquely identify encrypted subterms occurring in the protocol specification. The decidability of well-founded protocols confirms that even under this static notion of context-explicitness security protocols lose their ability to encode Turing complete models, even without bounding message size or the number of nonces. The key to the result was to introduce a notion of *honest causality*, which captures that honest information is propagated from one event to another: there is a causal chain which contains a backbone of messages and control flow transitions that could not have been manipulated by the intruder. One can then show that the depth of honest causality is bounded for well-founded protocols.

Context-explicit protocols also enjoy good properties wrt the compositionality problem. In particular, in [7] Ciobâcă and Cortier obtain: any attack trace on the composition of two differently tagged protocols can be transformed into an attack against one of the protocols. Hence, the security of a composed protocol can be derived from the security of each component protocol. The result

is obtained for a variant of the applied pi-calculus that covers both parallel and sequential composition. It therefore includes the case where one protocol uses a sub-protocol for an initial key exchange phase. This result is extended in [2], and also investigated for equivalence-based properties. A composition result is obtained for the case of key-exchange protocols wrt *diff-equivalence*.

2.4 Decidability and Complexity II

4. Decidability and Complexity II: "Use causal counterparts of equivalences and logics rather than interleaving ones to obtain good decidability and complexity."

Typical for the previous subsection is that the verification problem is still in the realm of interleaving semantics, i.e. can be formulated without relating to causality, but we use causal concepts to obtain the proof. The fourth slogan suggests to exploit causality more directly by taking the verification problem itself into the causal setting. The hope is that then good structural properties of the systems directly translate into good decidability and complexity properties, e.g. by admitting a divide and conquer approach.

While reachability-based security properties naturally fall into the previous category equivalence-based properties seem ideally suited to this approach. Indeed, there are only few decidability results for equivalence checking in the context of security protocols, and most of them have concentrated on how to handle messages rather than the computational power induced by the composition operators. In contrast, in "pure" concurrency theory there is a body of work, which has investigated the decidability and complexity of equivalence-checking for both classical and causal equivalences. Thereby motivated let us next take a closer look at the standard causal equivalences and their computational power compared to their interleaving counterparts.

3 Equivalences

3.1 Three Causal Equivalences

Equivalences for concurrency have mainly been studied in the classical setting where the behaviour of a concurrent system is captured in terms of transitions labelled by atomic actions rather than sending and receiving of terms. The various behavioural equivalences can be classified according to two main distinctions: one is *linear-time* versus *branching-time*; the second is *interleaving* versus *causality*.

In the linear-time view the behaviour of a system is understood in terms of its set of possible runs. If concurrency is abstracted away by nondeterministic interleaving a system run will simply be modelled as a totally ordered sequence of labelled transitions. Thus, the coarsest behavioural equivalence is *trace equivalence*: two systems are *trace equivalent* iff their sets of runs are equivalent up to isomorphism (i.e. as sequences of actions). In the causal approach a system run

is more faithfully modelled by a partially ordered set of labelled events. Since events with the same label may occur concurrently, technically we are dealing with *partially ordered multisets* of actions, or *pomsets* as coined by Pratt. The causal counterpart of trace equivalence is then represented by *pomset trace equivalence*: two systems are *pomset trace equivalent* iff their sets of labelled partial order runs are equivalent up to isomorphism (i.e. as pomsets of actions).

In branching-time notions of conflict, choice, or branching that naturally arise during a computation are modelled faithfully. The behaviour of a system is understood in terms of an 'unfolding' that reflects such information, and thereby shows how the system can unfold into many different possible futures. In the interleaving view a system naturally unfolds into a tree, or to be precise into a *synchronization tree* [39]. Attempts to capture what it means to distinguish branching in an *observational* way have culminated in the notion of *bisimulation equivalence* (short: *bisimilarity*) [39,42]. It is best explained in terms of a game between two players, Spoiler and Duplicator, on the two systems to be compared: Spoiler chooses a transition of one of the systems, and in response, Duplicator must choose a transition of the other system such that the labels are matching. The game then continues at the resulting pair of processes. The game continues like this forever, in which case Duplicator wins, or until either Spoiler or Duplicator is unable to move, in which case the other participant wins. Two systems are bisimilar iff Duplicator has a winning strategy in this game.

What are the unfolding structures of the causal approach? It turns out there are two different ways of capturing causality. One way is to stay within a tree-shaped view of the world but keep pointers that indicate when one transition is causally dependent on a previous transition of the same branch. The unfolding structure is then a *causal tree* [10]. The corresponding behavioural equivalence is *history preserving bisimilarity* (short: *hp-b*) [50]. It refines the bisimulation game as follows. Game positions now keep track of the history of the game. Technically, the histories are pairs (r_1, r_2, f) where r_1 is a partial order run of the first system, r_2 is a partial order run of the second system, and f is a pomset isomorphism between them. In her move Duplicator must now respond such that this pair of runs grows pomset isomorphic, i.e. $f \cup \{t_1, t_2\}$ must remain a pomset isomorphism.

The second way of capturing causality while keeping branching information departs from a tree-shaped structure but unfolds a system into an *event structure* [40]. In its most basic form an event structure is a set of events with a partial order that models causal dependence, and a symmetric and irreflexive relation added on that captures when two events are in conflict. Several axioms must hold to implement natural intuitions of this interpretation. They satisfy a basic principle of concurrency: whenever two independent events can occur consecutively they can also occur in the opposite order. The corresponding behavioural equivalence is *hereditary history preserving bisimilarity* (short: *hhp-b*) [3,31]. It further refines the bisimulation game by giving Spoiler the option of a backtrack move: Spoiler may choose a transition in one of the runs that is maximal in the partial order, and backtrack it. Duplicator must respond by backtracking the

transition in the other run that is related to Spoiler's transition in f. The game continues at the resulting histories. Note how the backtrack move reflects that history can be traced back in different ways just as independent transitions can be shuffled in their order.

3.2 Finite-State Results

For finite-state systems the decidability and complexity of the discussed equivalences are well-understood. Fig. 1 gives an overview for finite 1-safe Petri nets. Checking trace equivalence on finite-state transition systems is similar to checking language equivalence on finite automata, and turns out to be PSPACE-complete [34]. Bisimilarity on finite-state transition systems is PTIME-complete [34]; it can be solved in polynomial-time by partition-refinement algorithms [34,41]. Based on these classical results Jategaonkar and Meyer have obtained the following results for finite 1-safe Petri nets: trace equivalence is EXPSPACE-complete, and bisimilarity is DEXPTIME-complete respectively [30]. The blow-up in complexity reflects that the transition system induced by a finite 1-safe Petri net is in general exponentially larger than the size of the net.

	finite 1-safe Petri nets	
trace equivalence	EXPSPACE-complete	[30] using [34]
pomset trace equivalence	EXPSPACE-complete	[30]
bisimilarity	DEXPTIME-complete	[30] using [34]
hp-b	DEXPTIME-complete	[30]
hhp-b	undecidable	[32]

Fig. 1. Finite-state results

Hp-b and pomset trace equivalence behave similarly to their interleaving counterparts. Checking hp-b is DEXPTIME-complete [30]. It can be decided analogously to bisimilarity using the following insight: it is not necessary to keep the entire history to capture hp-b, but to see whether pomsets grow isomorphic it is sufficient to record only those events that can act as maximal causes. Moreover, this essential fragment of history can be captured in a finite way: by the *ordered markings* of [51], or the *growth-sites* of [30] respectively. The same insight leads to EXPSPACE-decidability of pomset trace equivalence [30].

In contrast, hhp-b turns out undecidable for finite 1-safe Petri nets [32]. The root cause of the higher power lies in the different way of capturing causality: by allowing Spoiler to backtrack the game is taken to the event-based unfolding level, where the relationship of transitions concerning concurrency and conflict is globally captured. A key insight is to use the following gadget that is inspired by a similar tool in [38]: A tiling system T to be played on the $\omega \times \omega$ grid can be universally encoded by a finite 1-safe Petri net $N(T)$ such that the building of a domino snake can be faithfully mimicked by a special pattern of forwards

and backtrack moves in the unfolding structure of $N(T)$. Hence, on their event-based unfolding level, systems such as finite 1-safe Petri nets are strong enough to encode tiling problems, and hence the computations of Turing machines, in a relatively straightforward sense.

All we have established so far is this: *in the finite-state world the causal equivalences are at least as hard as their interleaving counterparts.* So is our suggestion to use causal equivalences for their better composition properties a futile endeavour? Indeed, we have only reviewed here results on the full class of finite 1-safe Petri nets. There is a trend that suggests that as soon as we look at system classes that have good composition properties, and hence, a 'tame' interplay between causality, concurrency and conflict then hhp-b and to a degree also hp-b are better behaved than classical bisimilarity. A survey of results on subclasses of finite 1-safe Petri nets and open problems can be found in [16]. In the following, we will investigate this trend for infinite-state classes generated by process calculi.

3.3 A Hierarchy of Causal Processes

In the interleaving setting equivalence checking has been investigated along a hierarchy of process behaviours that can be captured in terms of rewrite rules. This *Process Rewrite Systems (PRS)* hierarchy is inspired by the Chomsky hierarchy of formal languages but the PRS grammars are interpreted as generators of infinite-state transition systems rather than languages. For borderline investigations of *causal* equivalences we consider the process algebras of the PRS-hierarchy as generators of infinite-state 1-safe Petri nets (or other causal models such as asynchronous transition systems).

Fig. 2 gives an overview of the PRS classes, expanded by the classes *Simple BPP* and *Simple PA* to be explained below. The root of the hierarchy comprises all *finite-state transition systems (FS)*. At the next level there are two extensions that can be seen as two interpretations of context-free grammars: *Basic Process Algebra (BPA)* extends FS by a sequential composition operator while *Basic Parallel Processes (BPP)* integrate a parallel composition operator. The class *Process Algebra (PA)* generalizes BPA and BPP by admitting both parallel and sequential composition. *PDA* is the class of *pushdown processes*, the processes described by pushdown automata, while on the right side we have *Petri nets (PN)*. The process classes on the left are not interesting here: since they do not integrate any parallel operator, the causal equivalences will coincide with their interleaving counterparts. Note that while in the classical interpretation the infinite-state classes contain all finite-state transition systems, under causal semantics up to PN they are incomparable with finite-state 1-safe Petri nets. This is so because BPP and PA restrict the interplay between concurrency and conflict due to the discipline of the grammars.

Given a set *Act* of atomic *actions*, usually denoted by $a, b, \ldots$, and a set *Var* of process *variables*, ranged over by $X, Y, \ldots$, the grammars for *FS, BPP, or PA process expressions* over *Act* and *Var* are defined as follows:

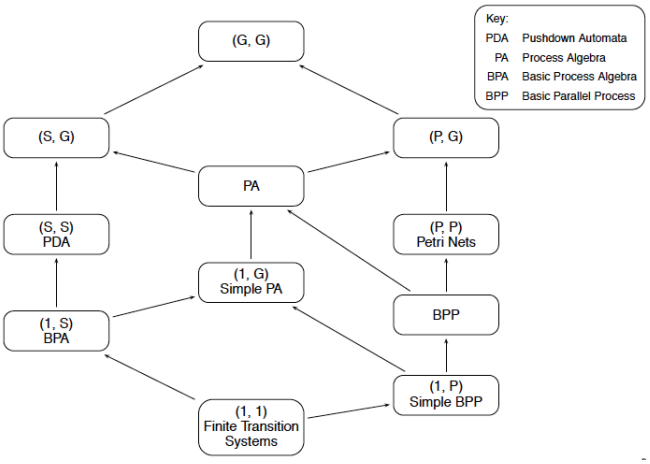

Fig. 2. The causal PRS-hierarchy

$$\begin{array}{lll} FS: & F ::= \mathbf{0} \mid X \mid a.F \mid F + F & \\ BPP: & E ::= \mathbf{0} \mid X \mid a.E \mid E + E \mid E \parallel E & \\ PA: & P ::= \mathbf{0} \mid X \mid a.P \mid P + P \mid P \parallel P \mid P \cdot P & \end{array}$$

where $\mathbf{0}$ denotes the empty process, X stands for a process variable, and $a._-$, $\_ + \_$, $\_ \parallel \_$, $\_ \cdot \_$ denote the operations of *action prefix* (for each $a \in Act$), *nondeterministic choice*, *parallel composition*, and *sequential composition* respectively. BPP processes are defined as pairs (E, Δ) where Δ is a finite family of (possibly recursive) defining equations $X_i \stackrel{\text{def}}{=} E_i$. As usual we require that each occurrence of a variable in E_i is guarded, i.e. within the scope of an action prefix. This analogous for FS and PA.

The PRS grammars give rise to BPP, and respectively PA, in normal form. While in the interleaving world they represent the entire process classes, under causal semantics they only describe the subclasses *Simple BPP*, and *Simple PA* respectively. They are defined by the following grammars:

$$\begin{array}{lll} SBPP: & E ::= X \mid S_E \mid E \parallel E & \\ SPA: & P ::= X \mid S_P \mid P \parallel P \mid P \cdot P & \end{array}$$

where S_E stands for an *initially sequential SBPP expression* given by the following grammar:

$$S_E ::= \mathbf{0} \mid a.E \mid S_E + S_E$$

and analogously for S_P. Thus, SBPP restrict the mixture of choice and parallel composition: general summation is replaced by *guarded summation*. In particular, this excludes processes such as $(P_1 \parallel P_2) + P_3$. This is similar for SPA.

Causal semantics for BPP processes have been provided in terms of net unfoldings, e.g. in [14], and equivalently in terms of event structures via their

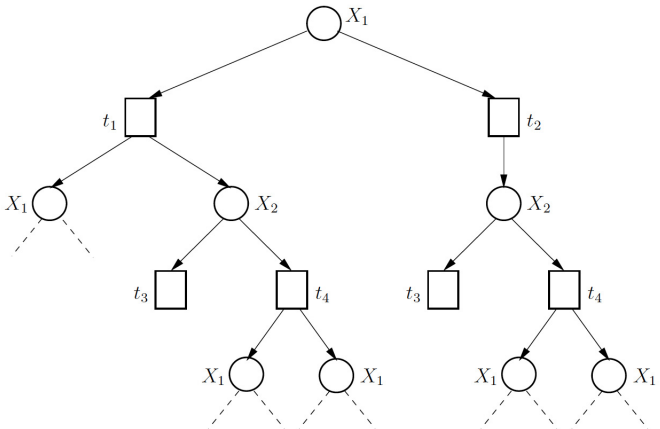

Fig. 3. The unfolding of SBPP $\mathcal{E}$.

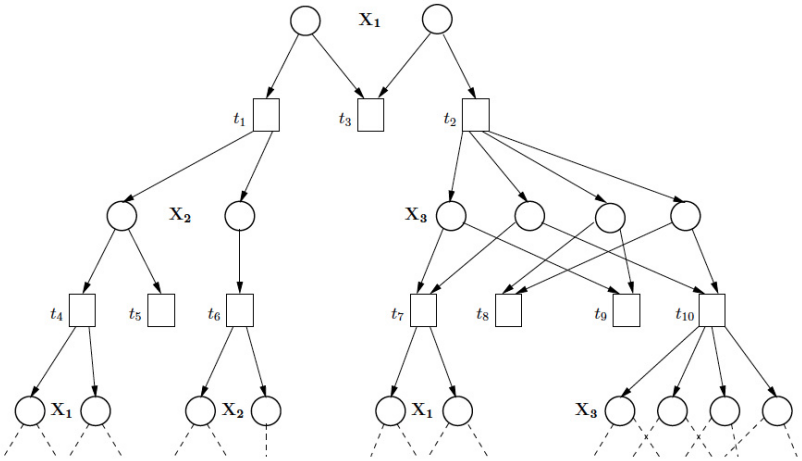

Fig. 4. The unfolding of BPP $\mathcal{E}$.

syntax-tree unfoldings [20]. Under such semantics BPP and SBPP have a tree-like structure. We provide two examples as an illustration.

Example 1. Fig. 3 gives the net unfolding of the SBPP $\mathcal{E} = (\Delta, X_1)$, where $\Delta = \{X_1 \stackrel{\text{def}}{=} t_1.(X_1 \parallel X_2) + t_2.X_2;\ X_2 \stackrel{\text{def}}{=} t_3.\mathbf{0} + t_4.(X_1 \parallel X_1)\}$.

	SBPP = BPP?	BPP	PA	PN
$=_{tr}$	yes	undecidable [25]	undecidable	undecidable [24,28]
$=_{pom}$	yes [49]	decidable [49]	?	undecidable
$\sim$	yes	PSPACE-complete [29,48]	PSPACE-hard	undecidable [28]
$\sim_{hp}$	no	P [20,36]	?	undecidable
$\sim_{hhp}$	no	P [20,21]	?	undecidable

Fig. 5. Summary

Example 2. Fig. 4 demonstrates the unfolding of the BPP $\mathcal{E} = (\Delta, X_1)$, where $\Delta = \{X_1 \stackrel{\text{def}}{=} (t_1.X_2 \parallel t_2.X_3) + t_3.\mathbf{0}; X_2 \stackrel{\text{def}}{=} (t_4.X_1 + t_5.\mathbf{0}) \parallel t_6.X_2; X_3 \stackrel{\text{def}}{=} (t_7.X_1 \parallel t_8.\mathbf{0}) + (t_9.\mathbf{0} \parallel t_{10}.X_3)\}$.

PA and SPA have not been equipped with causal semantics yet. But one would expect that an appropriately defined unfolding semantics would display a regular structure of fork and join of 'chunks of independent behaviour'.

Problem 1. Define causal semantics for SPA, and PA respectively.

3.4 Infinite-State Results

Fig. 5 gives an overview of results and open questions in the process hierarchy up to PA and PN.

PN. For *Petri nets* all equivalences are undecidable. This follows from Jančars's reduction from the halting problem of counter machines, which proves that bisimilarity as well as trace equivalence is undecidable [28]: first observe that Petri nets can simulate counter machines, but only in a weak way since they cannot check for 0; given a counter machine C one constructs two variations of the Petri net that weakly simulates C such that the difference between these two nets can only be exposed by faithfully simulating C and reaching the halting state (in one of the nets); the two nets are non-equivalent iff C halts. The proof carries over to hp-b and hhp-b. The undecidability of language equivalence was first proved by Hack [24], but Jančar's proof is stronger in that it only requires 5 unbounded places.

BPP. Building on Jančar's technique Hirshfeld managed to resolve that trace equivalence is undeciable for communication-free Petri nets, and hence BPP [25]. The result does not carry over to pomset trace equivalence on (S)BPP. In contrast, Sunesen and Nielsen prove that pomset trace equivalence is decidable for BPP [49]. The proof first shows that in the linear-time world SBPP and BPP coincide in that every BPP can effectively be translated into a SBPP such that they are pomset trace equivalent. The decidability result then follows by a reduction to the equivalence problem of recognizable tree languages. The complexity of the algorithm is left open (but should not be hard to resolve).

Problem 2. Resolve the complexity of pomset trace equivalence on (S)BPP.

While for BPP deciding classical bisimilarity is PSPACE-complete [29] we obtain polynomial-time decision procedures for both hp-b [36] and hhp-b [21]. The two bisimilarities coincide for SBPP [15] but they do *not* coincide for BPP in general. This follows by the standard example of [3]. The decision procedures rely on different techniques. However, they can both be decided due to good decomposition properties.

Hp-b has the *unique decomposition property*: every BPP can be expressed, up to hp-b, as a parallel composition of prime processes, where a process is prime when it cannot be represented as a non-trivial parallel composition, up to hp-b. Moreover, this decomposition is unique up to the permutation of the primes. Then hp-b can be decided using the general scheme of Hirshfeld, Jerrum, and Moller [26] of deciding classical bisimilarity on normed BPP [22,36]. The fastest algorithm for hp-b on BPP runs in $O(n^6)$ [20], and is based on the technique of the *distance-to-disabling functions* introduced in [29].

Hhp-b has even stronger decomposition properties: modulo trivial choices it fully reflects the structure of BPP expressions. It can then be decided similarly to the standard algorithm for solving tree isomorphism [20,21].

In [20] the decision algorithms for both hp-b and hhp-b are presented in a unified framework. Hhp-b is solved in time $O(n^3 log\ n)$, and hp-b in time $O(n^6)$ respectively. In particular, both algorithms use the fact that on BPP hp-b and hhp-b have a a fixpoint characterization in terms of local games played over BPP processes of causal depth 1.

To sum up, for SBPP and BPP we can confirm the trend that as soon as we look at system classes that have a 'tame' interplay between causality and conflict causal equivalences are better behaved than their interleaving counterparts. This concerns both the linear-time as well as the branching-time equivalences. All algorithms make use of the fact that under causal semantics BPP and SBPP have tree-like structure.

PA. While BPP and PN are well-investigated hardly anything is known for PA. Naturally, the hardness results of BPP carry over. Tackling PA in the interleaving world has turned out to be difficult. The only known positive result is that bisimilarity on normed PA is decidable in 2-NEXPTIME [27]. The proof is technically involved and 63 pages long. It is based on an exhaustive case analysis which investigates when a seqential and a parallel composition can be equivalent. This might be much easier for causal equivalences. We believe that in particular hhp-b on simple PA could have very strong decomposition properties analogously to those for BPP.

Problem 3. Does the positive trend for causal equivalences extend to SPA, and PA respectively?

4 Summary and Outlook

To sum up, we have the following trend for causal equivalences:

The computational dichotomy of causality: For finite-state systems causal equivalences are often computationally much harder than their interleaving counterparts. However, as soon as we consider classes with a restricted interplay between causality and conflict this trend may be reversed. In particular, this includes standard infinite-state classes such as BPP, and might extend to PA.

To use causal equivalences for equivalence-based security properties it is necessary to lift them into the applied pi-calculus. In particular, this means we have to introduce asynchronous message input and output. While this would destroy the causal structure enforced by the discipline of the composition operators we might be able to regain this structure when we restrict ourselves to tagged protocols. Say a composition is homogeneous when the components relate to the same protocol process, and heteregenous when the components relate to different protocol processes. Then, roughly speaking, one could say the results of [6,19] suggest:

Wrt homogeneous compositions of *tagged* protocols, atoms only have a bounded range of decisive influence, and messages only have decisive influence up to a bounded size.

Thereby we could hope to achieve decidability of reachability- and equivalence-based properties for tagged protocols when verified in isolation. Moreover, roughly speaking, one could say the results of [2,7] indicate:

Wrt hetergeneous compositions of *tagged* protocols, messages have only a local range of decisive influence as long as the tagging is disjoint.

One could define an appropriate calculus of tagged protocols, for which one could hope that reachability-based properties would be decidable, and causal equivalence-based properties would also be decidable by composition results based on the local effect of messages and the insights from the 'pure' causal equivalences. Altogether this should also lead to an efficient verification method as long as the component protocols are small.

It is nontrivial to put this down more formally, and it will be even less trivial to prove it (or disprove if it turns out not to be true!). However, it might be most difficult of all to carry such 'design for verification' paradigms like working with tagged protocols into the various standards. Striving to do so is essential: interpreted the other way around, this just implements the general engineering principle to only interconnect systems in a way that does not create any side effects.

Acknowledgments. Part of this work has been conducted while I was a member of Ernst-Rüdiger Olderog's group working towards my habilitation. I would like to express my sincere gratitude for his guidance and mentoring during this time. It is due to his influence that I have adopted the verification viewpoint and realized the necessity to address cyberphysical systems security.

References

1. Abadi, M., Needham, R.: Prudent engineering practice for cryptographic protocols. IEEE Trans. Softw. Eng. **22**(1), 6–15 (1996)
2. Arapinis, M., Cheval, V., Delaune, S.: Composing security protocols: from confidentiality to privacy. In: Focardi, R., Myers, A. (eds.) POST 2015. LNCS, vol. 9036, pp. 324–343. Springer, Heidelberg (2015)
3. Bednarczyk, M.: Hereditary history preserving bisimulation or what is the power of the future perfect in program logics. Technical report, Polish Academy of Sciences, Gdańsk (1991)
4. Blanchet, B., Abadi, M., Fournet, C.: Automated verification of selected equivalences for security protocols. Journal of Logic and Algebraic Programming **75**(1), 3–51 (2008)
5. Blanchet, B., Podelski, A.: Verification of Cryptographic Protocols: Tagging Enforces Termination. Theoretical Computer Science **333**(1–2), 67–90 (2005). Special issue FoSSaCS 2003
6. Chréetien, R., Cortier, V., Delaune, S.: Typing messages for free in security protocols: the case of equivalence properties. In: Baldan, P., Gorla, D. (eds.) CONCUR 2014. LNCS, vol. 8704, pp. 372–386. Springer, Heidelberg (2014)
7. Ciobâcă, Ş., Cortier, V.: Protocol composition for arbitrary primitives. In: Proceedings of the 23rd IEEE Computer Security Foundations Symposium (CSF 2010), Edinburgh, Scotland, UK, pp. 322–336. IEEE Computer Society Press, Edinburgh, July 2010
8. Cremers, C.J.F.: The scyther tool: verification, falsification, and analysis of security protocols. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 414–418. Springer, Heidelberg (2008)
9. Cremers, C.: Key exchange in ipsec revisited: formal analysis of IKEv1 and IKEv2. In: Atluri, V., Diaz, C. (eds.) ESORICS 2011. LNCS, vol. 6879, pp. 315–334. Springer, Heidelberg (2011)
10. Darondeau, P., Degano, P.: Causal trees. In: Ausiello, G., Dezani-Ciancaglini, M., Rocca, S.R.D. (eds.) ICALP 1989. LNCS, vol. 372, pp. 234–348. Springer, Heidelberg (1989)
11. ETSI. TS 102 731 V1.1.1: ITS; security; security services and architecture, 09 2010
12. ETSI. TS 102 940 V1.1.1: ITS; security; ITS communications security architecture and security management, 06 2012
13. Fábrega, F.J.T., Herzog, J.C., Guttman, J.D.:. Strand spaces: Why is a security protocol correct? In: Symposium on Security and Privacy. IEEE Computer Society (1998)
14. Fröschle, S.: Decidability and Coincidence of Equivalences for Concurrency. PhD thesis, University of Edinburgh (2004)
15. Fröschle, S.: Composition and decomposition in true-concurrency. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 333–347. Springer, Heidelberg (2005)
16. Fröschle, S.: The decidability border of hereditary history preserving bisimilarity. Information Processing Letters **93**(6), 289–293 (2005)
17. Fröschle, S.: The insecurity problem: tackling unbounded data. In: Proceedings of the 20th IEEE Computer Security Foundations Symposium, pp. 370–384 (2007)
18. Fröschle, S.: Causality in Security Protocols and Security APIs: Foundations and Practical Verification, Habilitation thesis, Universität Oldenburg (2012)
19. Fröschle, S.: Leakiness is decidable for well-founded protocols. In: Focardi, R., Myers, A. (eds.) POST 2015. LNCS, vol. 9036, pp. 176–195. Springer, Heidelberg (2015)

20. Fröschle, S., Jančar, P., Lasota, S., Sawa, Z.: Non-interleaving bisimulation equivalences on basic parallel processes. Information and Computation **208**(1), 42–62 (2010)
21. Fröschle, S., Lasota, S.: Decomposition and complexity of hereditary history preserving bisimulation on BPP. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 263–277. Springer, Heidelberg (2005)
22. Fröschle, S., Lasota, S.: Normed processes, unique decomposition, and complexity of bisimulation equivalences. In: Proceedings of INFINITY 2006-2009, vol. 239, pp. 17–42. Elsevier (2009)
23. Fröschle, S., Sommer, N.: Reasoning with past to prove PKCS#11 keys secure. In: Degano, P., Etalle, S., Guttman, J. (eds.) FAST 2010. LNCS, vol. 6561, pp. 96–110. Springer, Heidelberg (2011)
24. Hack, M.: The equality problem for vector addition systems is undecidable. Theoret. Comput. Sci. **2**(1), 77–95 (1976)
25. Hirshfeld, Y.: Petri nets and the equivalence problem. In: Meinke, K., Börger, E., Gurevich, Y. (eds.) CSL 1993. LNCS, vol. 832, pp. 165–174. Springer, Heidelberg (1994)
26. Hirshfeld, Y., Jerrum, M., Moller, F.: A polynomial time algorithm for deciding bisimulation equivalence of normed Basic Parallel Processes. Mathematical Structures in Computer Science **6**, 251–259 (1996)
27. Hirshfeld, Y., Jerrum, M.: Bisimulation equivalence is decidable for normed process algebra (Extended abstract). In: Wiedermann, J., Van Emde Boas, P., Nielsen, M. (eds.) ICALP 1999. LNCS, vol. 1644, pp. 412–421. Springer, Heidelberg (1999)
28. Jančar, P.: Undecidability of bisimilarity for Petri nets and some related problems. Theoretical Computer Science **148**(2), 281–301 (1995). STACS 1994
29. Jančar, P.: Bisimilarity of basic parallel processes is PSPACE-complete. In: Proc. LICS 2003, pp. 218–227. IEEE Computer Society (2003)
30. Jategaonkar, L., Meyer, A.R.: Deciding true concurrency equivalences on safe, finite nets. Theoretical Computer Science **154**(1), 107–143 (1996)
31. Joyal, A., Nielsen, M., Winskel, G.: Bisimulation from open maps. Information and Computation **127**(2), 164–185 (1996)
32. Jurdziński, M., Nielsen, M., Srba, J.: Undecidability of domino games and hhp-bisimilarity. Inform. and Comput. **184**, 343–368 (2003)
33. Kamil, A., Lowe, G.: Analysing tls in the strand spaced model. Journal of Computer Security **19**(5), 975–1025 (2011)
34. Kanellakis, P.C., Smolka, S.A.: CCS expressions, finite state processes, and three problems of equivalence. Information and Computation **86**(1), 43–68 (1990)
35. Künnemann, R.: Automated backward analysis of PKCS#11 v2.20. In: Focardi, R., Myers, A. (eds.) POST 2015. LNCS, vol. 9036, pp. 219–238. Springer, Heidelberg (2015)
36. Lasota, S.: A polynomial-time algorithm for deciding true concurrency equivalences of basic parallel processes. In: Rovan, B., Vojtáš, P. (eds.) MFCS 2003. LNCS, vol. 2747, pp. 521–530. Springer, Heidelberg (2003)
37. Lowe, G.: Towards a completeness result for model checking of security protocols. Journal of Computer Security **7**(1), 89–146 (1999)
38. Madhusudan, P., Thiagarajan, P.S.: Controllers for discrete event systems via morphisms. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 18–33. Springer, Heidelberg (1998)
39. Milner, R. (ed.): A calculus of communicating systems. LNCS, vol. 92. Springer, Heidelberg (1980)

40. Nielsen, M., Plotkin, G.D., Winskel, G.: Petri nets, event structures and domains, part i. Theor. Comput. Sci. **13**, 85–108 (1981)
41. Paige, R., Tarjan, R.E.: Three partition refinement algorithms. SIAM J. Comput. **16**(6), 973–989 (1987)
42. Park, D.: Concurrency and automata on infinite sequences. In: Deussen, P. (ed.) Theoretical Computer Science. LNCS, vol. 104, pp. 167–183. Springer, Heidelberg (1981)
43. Ramanujam, R., Suresh, S.P.: A decidable subclass of unbounded security protocols. In: WITS 2003, pp. 11–20 (2003)
44. Ramanujam, R., Suresh, S.P.: Tagging makes secrecy decidable with unbounded nonces as well. In: Pandya, P.K., Radhakrishnan, J. (eds.) FSTTCS 2003. LNCS, vol. 2914, pp. 363–374. Springer, Heidelberg (2003)
45. Ramanujam, R., Suresh, S.P.: Decidability of context-explicit security protocols. Journal of Computer Security **13**(1), 135–165 (2005)
46. Schmidt, B., Sasse, R., Cremers, C., Basin, D.: Automated verification of group key agreement protocols. In: Proceedings of the 2014 IEEE Symposium on Security and Privacy SP 2014, DC, USA, Washington, pp. 179–194 (2014)
47. Song, D.X.: Athena: A new efficient automatic checker for security protocol analysis. In: CSFW 1999, pp. 192–202. IEEE Computer Society (1999)
48. Srba, J.: Strong bisimilarity and regularity of basic parallel processes is PSPACE-hard. In: Alt, H., Ferreira, A. (eds.) STACS 2002. LNCS, vol. 2285, pp. 535–546. Springer, Heidelberg (2002)
49. Sunesen, K., Nielsen, N.: Behavioural equivalence for infinite systems—partially decidable!. In: Billington, J., Reisig, W. (eds.) ICATPN 1996. LNCS, vol. 1091, pp. 460–479. Springer, Heidelberg (1996)
50. van Glabbeek, R., Goltz, U.: Equivalence notions for concurrent systems and refinement of actions. In: Kreczmar, A., Mirkowska, G. (eds.) MFCS 1989. LNCS, vol. 379, pp. 237–248. Springer, Heidelberg (1989)
51. Vogler, W.: Deciding history preserving bisimilarity. In: Albert, J.L., Monien, B., Artalejo, M.R. (eds.) Automata, Languages and Programming. LNCS, vol. 510, pp. 495–505. Springer, Heidelberg (1991)