On the Parameterized Verification of Abstract Models of Contact Tracing Protocols

Sylvain Conchon[1], Giorgio Delzanno[2] and Arnaud Sangnier[3]

[1]*LRI, Universitè Paris, France*
[2]*DIBRIS, University of Genova, Italy*
[3]*IRIF, Universitè Paris Denis Diderot*

Abstract
We present an automata-based formal model of distributed systems specifically devised to formalise abstractions of Contact Tracing Protocols that combine Bluetooth and network communication. The model combines pure names, read/write operations on first-order and higher-order variables and synchronous communication primitives. The transition system models the interaction between devices in the same physical location and between a single device and a notification server. To automatically validate protocols in our formalism, we resort to an extension of the Cubicle SMT-based infinite-state model checker, in which the monotonic abstraction applied during the predecessor computation is strengthen by introducing abstract predicates obtained via Counting Abstraction.

1. Introduction

Contact tracing protocols, e.g., smartphone apps, are aimed at investigating who could have been contaminated by a positively diagnosed person and alert them, see e.g. [1]. After the COVID-19 pandemic, several protocols have been proposed as an automated support for this task su as PEPP-PT, DP3T, ROBERT4 and NTK5. In general the system architecture is based on a smart app, a notification server as in Publish/Subscribe architectures, and a possibly trusted server. In order to reduce the risk of a malicious use of user data (see e.g. the attacks described in [2, 1]), the protocols are based on frequent rotations of identifiers emitted via bluetooth. In centralized versions of the protocol ephemeral identifiers are downloaded from a trusted server. In decentralized versions ephemeral identifiers are generated by user apps. The user app installed on the user device (e.g. Immuni in Italy) broadcasts the currently valid ephemeral identifier. The same user app also collects all identifiers emitted in the same bluetooth range and stores them locally. When users are diagnosed to be infected, they release a report to the server. One possible implementation consists in sending the report to a central log database. Users can then consult the database to check if they crossed their way with infected users, i.e., they share some ephemeral identifier with one of them.

Based on a preliminary study present at Overlay 2020 [3], in this paper we present an automata-based formal model of distributed systems specifically devised to formalise abstractions of Contact Tracing Protocols that combine Bluetooth and network communication. The model

OVERLAY 2021: 3rd Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis, September 22, 2021, Padova, Italy

combines pure names, read/write operations on first-order and higher-order variables and synchronous communication primitives. The transition system models the interaction between devices in the same physical location and between a single device and a notification server. Violations of safety properties for this kind of protocols typically require universally quantified conditions on global configurations (e.g. one user detects a contact while no user is positive).

To automatically validate protocols in our formalism, we resort to an extension of the Cubicle SMT-based infinite-state model checker. Cubicle is a model checker that can be applied to verify safety properties of array-based systems, a syntactically restricted class of parameterized transition systems with states represented as arrays indexed by an arbitrary number of processes [4, 5]. Cubicle integrates the SMT solver Alt-Ergo [6]. Cubicle input language is a typed version of Murϕ [7]. A system is describe in Cubicle by: a set of type, variable, and array declarations; a formula for the initial states; and a set of transitions. A system execution is defined by an infinite loop that at each iteration non-deterministically chooses a transition instance whose guard is true in the current state; and updates state variables according to the action of the fired transition instance.

To validate instances of our model, we will exhibit a general encoding in the Cubicle array-based language and propose to refine the overapproximation of universally quantified preconditions applied in the Cubicle backward procedure by introducing abstract predicates obtained via Counting Abstraction.

2. Contact Tracing Systems

In this section we introduce a possible general formal model for contact tracing protocol. To specify the protocol rules and behavior (computation), we will use a transition system defined on configurations consisting of a set of states of individual users. User configurations contain a local memory needed to store the list of emitted and received identifiers. System configurations contain a global memory that can be used to gather and a set of user states that corresponds to the current number of registered indivuals.

We will use the following general definitions: $\mathcal{I}$ is a denumerable set of pure names (identifiers) containing the undefined label $\perp$; M is a finite set of message labels $(a, b, c, \dots)$; V is a finite set of first order variables (identifiers) that are part of the local state of users $(x, y, \dots)$; S is a finite set of second order variables (set of identifiers) that are part of the local state of users $(s, t, \dots)$. We then define $\texttt{Act}(M, V, S)$ the set of actions defined as follows: $eq(x, y)$ and $noteq(x, y)$, where $x, y \in V$, $in(x, m)$ and $notin(x, m)$, where $x \in V$ and $m \in M$ (Conditions); $local$, $new(x)$, where $x \in V$, $bcast(m, x)$ and $rec(m, x)$, where $m \in M, x \in V$; $add(x, s)$, where $x \in V, s \in S$, $reset(s)$, where $s \in S$, $reg(m, s)$, where $m \in M, s \in S$ (Operations).

A protocol is defined as a Data Automaton $P = \langle Q, M, V, S, R, q_0 \rangle$, where $q_0 \in Q$ and $R \subseteq Q \times \texttt{Act}(M, S, V) \times Q$. We assume here that $bcast$ and reg actions are always defined on distinct labels in M. Consider a protocol $P = \langle Q, M, V, S, R, q_0 \rangle$. A process configuration of P is a triple $pc = \langle q, \delta, \gamma \rangle$ where: $q \in Q$ is the current user state; $\delta : V \to \mathcal{I}$ is the current evaluation of first order variables; $\gamma : S \to \mathcal{P}(\mathcal{I})$ is the current evaluation of second order variables. We denote by $\mathcal{PC}$ the set of process configurations. A global configuration of P is then a pair (K, Θ), where $K \in \mathbb{N}^{\mathcal{PC}}$ is a finite multiset of process configurations and $\Theta : M \mapsto \mathcal{P}(\mathcal{I})$

corresponds to a centralized memory. $\mathcal{GC}$ denotes the set of global configurations. For $\sigma \in \mathcal{GC}$, we will use $ids(\sigma)$ to denote the set of pure names occurring in σ. The operational semantics is then defined as the minimal relation $\rightarrow \subseteq \mathcal{G} \times \mathcal{G}$ that satisfies the following property definitions in which will use $\uplus$ to denote multiset union:

- $\sigma = \langle \{\langle q, \delta, \gamma \rangle\} \uplus K, \Theta \rangle \rightarrow \langle \{\langle q', \delta, \gamma \rangle\} \uplus K, \Theta \rangle$ if one of the following conditions is satisfied: $\langle q, local, q' \rangle \in R$, $\langle q, eq(x,y), q' \rangle \in R$ and $\delta(x) = \delta(y)$ [for $noteq(x,y)$,$\delta(x) \neq \delta(y)$], $\langle q, in(x,m), q' \rangle \in R$ and $\delta(x) \in \Theta(m)$ [for $notin(x,y)$ $\delta(x) \notin \Theta(m)$].

- $\sigma = \langle \{\langle q, \delta, \gamma \rangle\} \uplus K, \Theta \rangle \rightarrow \langle \{\langle q', \delta', \gamma \rangle\} \uplus K, \Theta \rangle$ if $\langle q, new(x), q' \rangle \in R$ and $\delta'(x) = id \notin ids(\sigma), id \neq \bot$, and $\delta'(y) = \delta(y)$ for $x \neq y$.

- $\sigma = \langle \{\langle q, \delta, \gamma \rangle\} \uplus K, \Theta \rangle \rightarrow \langle \{\langle q', \delta, \gamma' \rangle\} \uplus K, \Theta \rangle$ if $\langle q, add(x,s), q' \rangle \in R$ and $\gamma'(s) = \gamma(s) \cup \{\delta(x)\}$ and $\gamma'(t) = \gamma(t)$ for $t \neq s$.

- $\sigma = \langle \{\langle q, \delta, \gamma \rangle\} \uplus K, \Theta \rangle \rightarrow \langle \{\langle q', \delta, \gamma' \rangle\} \uplus K, \Theta \rangle$ if $\langle q, reset(s), q' \rangle \in R$ and $\gamma'(s) = \emptyset$ and $\gamma'(t) = \gamma(t)$ for $t \neq s$.

- $\sigma = \langle \{\langle q, \delta, \gamma \rangle\} \uplus K, \Theta \rangle \rightarrow \langle \{\langle q', \delta, \gamma \rangle\} \uplus K, \Theta' \rangle$ if $\langle q, reg(m,s), q' \rangle \in R$ and $\Theta'(m) = \Theta(m) \cup \gamma(s)$ and $\Theta'(n) = \Theta(n)$ for $n \neq m$.

- $\sigma = \langle \{\langle q, \delta, \gamma \rangle, \langle q_1, \delta_1, \gamma_1 \rangle, \ldots, \langle q_k, \delta_k, \gamma \rangle_k\} \uplus K, \Theta \rangle \rightarrow \langle K', \Theta \rangle$ where $K' = \{\langle q', \delta, \gamma \rangle, \langle q'_1, \delta'_1, \gamma_1 \rangle, \ldots, \langle q'_k, \delta'_k, \gamma_k \rangle\} \uplus K$ if $k \geq 0$ and $\langle q, bcast(m,x), q' \rangle \in R$ and for all $i \in [1,k]$, we have $\langle q_i, rec(m, x_i), q'_i \rangle \in R$ and $\delta'_i(x_i) = \delta(x)$ and $\delta'_i(y) = \delta_i(y)$ for $y \neq x_i$.

For the resulting model, we are interested in safety properties for parameterized version of a protocol instance. More specifically, given a protocol definition P and a given process state $q_e \in Q$ that denotes an error state, we would like to verify that global configurations that contain occurrences of q_e are unreachable starting from any possibile initial configuration (i.e. independently from the number of process instances). A similar property can be formulated by adding an error flags to the server state instead of selecting a special error state in Q. We are also interested in properties expressed by violations in which global configurations contain a process in state q_e while all other processes are in states contained in These properties are referred to as control state reachability and constrained control state reachabiliy, respectively. Violations of this kind of properties can be defined using quantified assertions, such as there exists a process i in control state q_e and, for constrained properties, universally quantified conditions on the control states of other processes. In the rest of the paper we will illustrate how to encode our formalism in Cubicle and how to verify control state reachability problems using the corresponding symbolic exploration procedure provided by the tool.

3. Case Studies: Contact Tracing Protocols

We present below a formal specifications of the protocol scheme of Contact Tracing applications abstracting away identifier creation, i.e., assuming that user app generate fresh identifiers, and reasoning within the validity epoch of identifiers, i.e., those maintained in local memory. The system is composed of a finite but arbitrary number of user apps and of a server modeled via a centralized memory. Each user apps has local variables $V = \{x, y, z\}$ and $S = \{in, out\}$. Furthermore, $M = \{b, m\}$ where b denotes beacon messages broadcasted by user apps, and

m denotes report messages sent to the centralized memory. The user app starts their part of protocol in state q_0 and operate in three different modes.

In emitter mode the app generates and emits fresh beacons: $\langle q_0, new(x), q_1\rangle$, the user app generates a fresh identifier and stores it in the local variable x; $\langle q_1, add(x, out), q_2\rangle$, in q_1 the fresh identifier stored in x is added to the local memory out; $\langle q_2, bcast(b, x), q_2\rangle$, $\langle q_2, local, q_0\rangle$, in q_2 the user app either emits the message (b, x) or returns to the initial state in order to apply a rotation scheme for the beacon identifiers or perform other steps. In reception mode the app, while in any state $l_0 \in Q \setminus \{r_1, s_2\}$, receives a beacon, stores it in the local memory in, and then returns to the current state: $\langle l_0, rec(b, y), l_1\rangle$ in state l_0 the user is always ready to receive beacons; $\langle l_1, add(y, in), l_0\rangle$, in state l_1 the beacon is stored in the local memory in. In report mode, enabled only in state $r_0 = q_0$, the app sends its local memory out to the server and moves to the halt state r_1: $\langle q_0, send(m, out), r_1\rangle$. In query mode, enabled only in state $s_0 = q_0$, the user app selects a beacon z from the local memory in, i.e., $\langle s_0, sel(in, z), s_1\rangle$, sends it to the server to check whether it has been reported by another app or not. In the former case it moves to the halting state s_2, i.e, $\langle s_1, in(m, x), s_2\rangle$. In the latter case it returns to state s_0, i.e, $\langle s_1, notin(m, x), s_0\rangle$.

4. From Data Automata to Cubicle

Let us now discuss how to model a protocol definition $P = \langle Q, M, V, S, R, q_0\rangle$ in Cubicle. A global configuration is a pair (K, Θ), where K is a finite multiset of process configurations and Θ corresponds to a centralized memory. The encoding if Θ is quite immediate since we can represent the global memory as a finite set of unbounded arrays $A_{m_1}, \ldots, A_{m_n}$ with cells of Boolean type one for each $m_i \in M$. The array A_m is such that $A_m[x] = True$ if and only if $x \in \Theta(m)$ for each pair x, m. For the encoding of K we can proceed in several different ways. Remember that a process configuration is a triple $pc = \langle q, \delta, \gamma\rangle$, in which $q \in Q$ but is the current user state, δ is the current evaluation of first order variables; and γ is the current evaluation of second order variables. One possible encoding is defined as follows: To represent control states, we introduce an unbounded array $state$ that associates an enumerative type associated to Q to represent the current state of each process, i.e., $state[i] = q$ if process i is in state q. To represent the current state of a first order variable v we can introduce an unbounded array val_v such that $val_v[i] = d$ if $\delta(v) = d$ holds in pc_i (in process i). To represent the current state of a second order variable s we can introduce an unbounded bidimensional array val_s such that $val_s[i, d] = True$ if and only if $d \in \gamma(s)$ holds in pc_i (in process i). Transition rules can then be expressed via Cubicle transition rules. In particular, to model send operation involving second order variables, we can use global operations on arrays, in which all cells of an arrays are copied into another one (whole-place operations in Petri net languages).

The Cubicle verification engine is based on symbolic backward exploration. Cubicle operates over sets of existentially quantified formulas called cubes. Formulas containing universally quantified formulas (generated during the computation of predecessors) are over-approximated by existentially quantified formulas. In other words Cubicle applies monotone abstraction [8] and over-approximates predecessors via upward-closed sets of configurations. Infinite sets of unsafe states (bad configurations) are symbolically defined by using $unsafe$ constraints in which all variables are implicitly existentially quantified. In our case studies we are interested in checking the consistency between local and global information. More specifically, when user U queries the server, she/he supposed to receive a notification only in presence of at least one positive user who emitted a beacon that has been received by U. The server does not maintain any association between identities and beacons collected in the log. Consistency of global and local states combined with the privacy preservation property of the protocol can be tested by adding an Error state and a special rule in which we only compare local states of different users. Namely, the rule generates an error whenever there exist a user U in the `Contact` state without any positive user in the global system. If this happens the protocol does not trace physical contacts in correct way. The property is expressed by adding a global `Error` flag

```
Error: bool
```

together with the following rule:

```
transition bad(v)
requires { Contact[v] = True && forall_other u. (Pos[u] = False) }
{ Error := True; }
```

To validate the considered properties, we define the following assertion that specifies bad configurations in the sense of control state reachability,

```
unsafe () { Error = True }
```

i.e., configurations of any size in which `Error` is `True`. Due to the presence of universally quantified conditions both in the protocol rule and in the precondition of the error rule, there is no termination guarantee on the execution of the verification task in Cubicle. Furthermore, when terminating the tool may return spurious traces as in our case study. Indeed , Cubicle, after visiting 41 cubes, detects a spurious trace of the form: start(#3, #2) -> in(#1, #2) -> end(#1, #2) -> report(#3) -> query(#1, #2) -> bad(#1) -> unsafe.

The imprecision here is due to the over-approximation introduced via monotone abstraction. Preconditions with universally quantified guards are transformed into post-conditions, and thus they are not propagated through different steps of predecessor computations. Since in the correctness property of the protocol local states are put in relation via hidden data (not explicitly mentioned in control states) stored in the server memory, it seems necessary to strengthen the symbolic reasoning procedure with some form of propagation of universally quantified conditions. One solutions comes from the combination of the existentially quantified formulas used to represent sets of states in Cubicle (cubes) with other forms of constraints on the global configuration e.g. constraints inspired to counting abstraction used in Petri nets. More in detail, we enrich our assertions by adding counters that keep track of the number of users in a given state. For our purposes, we will focus only on a counter associated to the `Pos` flag. We now add a global variable `Count` whose initial state is defined as `Count=0`. The counter is updated in the report rule ($Count \geq 0$ is added to the guard and $Count := Count + 1$ is added to the body off the transition) and used as precondition in the error rule as specified below.

```
transition bad(v)
requires { Pos[v] = False  && Contact[v] = True && Count = 0 }
{ Error := True; }
```

The enriched assertional language combining array formulas (for expressing precise properties of individual processes and of global variables) and Presburger constraints (for expressing global constraints via the counting abstraction) is the key point in order to enforce termination and prove correctness for the desired property. When executed on the refined model, Cubicle proves the model SAFE after visiting 10 cubes and performing 138 fixpoint tests and 41 SMT solver calls.

Although not yet implemented, the above mentioned strategy could be incorporated into the Cubicle algorithm for instance by associating counters to specific control states and by generating counter manipulation operations in each transition rule via a preliminary static analysis of the Cubicle specification as done in our example via human ingenuity.

References

[1] S. Vaudenay, Centralized or decentralized? the contact tracing dilemma, IACR Cryptol. ePrint Arch. 2020 (2020) 531. URL: https://eprint.iacr.org/2020/531.

[2] G. Avitabile, V. Botta, V. Iovino, I. Visconti, Towards defeating mass surveillance and sars-cov-2: The pronto-c2 fully decentralized automatic contact tracing system, IACR Cryptol. ePrint Arch. 2020 (2020) 493.

[3] P. A. Abdulla, M. F. Atig, G. Delzanno, M. Montali, A. Sangnier, On the formalization of decentralized contact tracing protocols, in: Proceedings of the 2nd Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis hosted by the Bolzano Summer of Knowledge 2020 (BOSK 2020), September 25, 2020, volume 2785 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2020, pp. 65–70.

[4] S. Conchon, A. Goel, S. Krstic, A. Mebsout, F. Zaïdi, Cubicle: A parallel smt-based model checker for parameterized systems - tool paper, in: Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings, 2012, pp. 718–724.

[5] A. Mebsout, Inférence d'invariants pour le model checking de systèmes paramétrés. (Invariants inference for model checking of parameterized systems), Ph.D. thesis, University of Paris-Sud, Orsay, France, 2014.

[6] https://alt-ergo.ocamlpro.com/, 2021.

[7] D. L. Dill, The mur*phi* verification system, in: Computer Aided Verification, 8th International Conference, CAV '96, New Brunswick, NJ, USA, July 31 - August 3, 1996, Proceedings, volume 1102 of *Lecture Notes in Computer Science*, 1996, pp. 390–393.

[8] P. A. Abdulla, G. Delzanno, N. B. Henda, A. Rezine, Monotonic abstraction: on efficient verification of parameterized systems, Int. J. Found. Comput. Sci. 20 (2009) 779–801. doi:10.1142/S0129054109006887.