12-2022

Compliance through model checking

Avishkar MAHAJAN

STRECKER Martin

Seng Joe WATT

Meng Weng (HUANG Mingrong) WONG
Singapore Management University, mwwong@smu.edu.sg

Citation

MAHAJAN, Avishkar; STRECKER Martin; WATT, Seng Joe; and WONG, Meng Weng (HUANG Mingrong). Compliance through model checking. (2022). *International Workshop on AI Compliance Mechanism WAICOM 2022*.
Available at: https://ink.library.smu.edu.sg/cclaw/3

Compliance through Model Checking

Avishkar Mahajan[0000−0002−9925−1533], Martin Strecker⋆[0000−0001−9953−9871], Watt Seng Joe[0000−0002−6883−4736], and Meng Weng Wong[0000−0003−0419−9443]

Singapore Management University

Abstract. In this short note, we describe part of a case study about Singapore's Personal Data Protection Act, which we first presented informally, then formally as interacting Timed Automata. From these, we derive desiderata on a language and verification framework for reasoning about compliance.

Keywords: Compliance, Knowledge representation and reasoning, Computational Law, Model Checking

1 Introduction

With the present paper, we intend to provide solutions to the problem of specifying and enforcing compliance in particular in state-based, time-dependent systems. Our solutions are far from complete or conclusive – our short note is meant to present a snapshot of our current work and to contribute to a discussion about relevant issues: when are legal requirements coherent (so that they can be implemented)? How can compliance be technically enforced? How can violations be detected and error scenarios be communicated?

Our contribution first presents a case study that we have been working on at Singapore's Centre for Computational Law, CCLAW[1], dealing with a particularly relevant fragment of publicly available legislation and regulation: Singapore's Personal Data Protection Act. It will be described informally in Section 2. We will then proceed to a formal analysis in Section 3, highlighting some problems of the current legislation that could be termed an internal inconsistency and that has in part been revealed by the formal analysis. We then conclude with a description of desiderata on a language and verification framework for reasoning about compliance, in Section 4.

Our approach continues work on "contracts as automata" [7] and has similarities with other automata-based approaches for reasoning about contracts, in particular [3,10].

⋆ part of the work carried out at Toulouse University

[1] https://cclaw.smu.edu.sg/

2 Case Study: PDPA

Singapore's Personal Data Protection Act [11] consists of a set of rules governing the use of personal data in private and public organizations, and describes actions to be taken if a data breach is detected. It consists on the one hand of *constitutive* rules, *i.e.* definitions specifying what is considered a data breach and under which conditions it is deemed a notifiable data breach. On the other hand, *regulative* rules prescribe which actions need to be taken if a notifiable data breach is detected.

The constitutive rules are sufficiently voluminous and complex to warrant the development of an expert system assisting organizations in assessing whether a data breach is notifiable. In the CCLAW project, we have developed such a system. The regulative rules are seemingly less involved but sufficiently intricate that actors that precisely follow the rules may wind up in a state where they have breached the law. The complexity results from the interplay of temporal conditions that remain implicit and that would have to be explicated to guarantee lawful behavior.

We here give an abridged account of the relevant rules; for details, see §§ 26A to E of [11] that leads to and motivates our formalization in Section 3. The PDPA identifies three actors in a data breach scenario:

- the *organization* in which a data breach has occurred;
- the Personal Data Protection Commission (PDPC, henceforth only called the *commission*), the governmental authority that has to be notified in case of a breach;
- the *individual* affected by a data breach. The abstraction of the multitude of affected individuals to a single entity is already done in the law text and seems appropriate as there is no interaction among the individuals.

The temporal requirements are as follows:

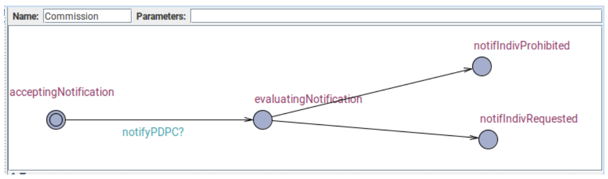
- When a data breach is detected, the organization has up to thirty days to assess whether the data breach is notifiable; if it is not, no further action is required, and the process stops there.
- If the breach is notifiable, the organization is obliged to inform the commission within three days of having recognized the breach as notifiable.
- If the breach is notifiable, any affected individual also has to be informed within the three day period.
- The organization must not notify an affected individual if the commission so directs.

The inconsistency, that was in part revealed by the formalization and then confirmed by model checking, arises from the lack of temporal coordination between the action of informing the commission and the individual, and possibly the interdiction by the commission to inform the individual.

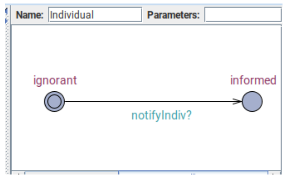
3 Formal Analysis

3.1 Formal Model

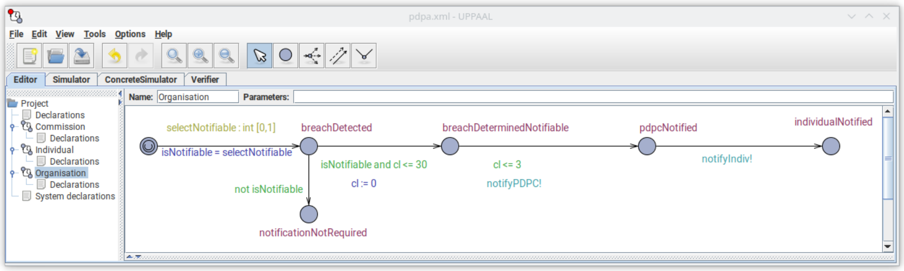
We have carried out a formal analysis of this scenario with Timed Automata (TA) in the Uppaal [9] model checker. The global setup is shown in Figure 1. There are three interacting automata, one for each of the above-mentioned actors. The states carry names (in mauve), the initial states are marked with a double circle. The automata synchronize via messages (in turquoise), where a send action is indicated by an exclamation and a receive action by a question mark. Transitions may depend on Boolean conditions (here: variables such as `isNotifiable`) and on temporal conditions modeled with the aid of *clocks*. In this example, there is one clock, `cl`, that is zero in the initial state, and that is again reset to zero in the transition from the state `breachDetected` to `breachDeterminedNotifiable`.



(a) Commission



(b) Individual



(c) Organization

Fig. 1: Automata modeling the PDPA scenario

According to the TA semantics, there are two kinds of transitions: the passage of time, at the same rate in all automata, or discrete state changes along enabled transitions, *i.e.* transitions whose conditions are satisfied. In case several transitions are enabled, one of them is chosen non-deterministically. This happens for example in the Commission automaton after receiving a notification (`notifyPDPC?`), modeling the fact that the commission's decision to prohibit or request the notification of the affected individual is autonomous and not influenced by external factors. The Individual automaton only provides two states corresponding to whether the individual has not yet / has been notified.

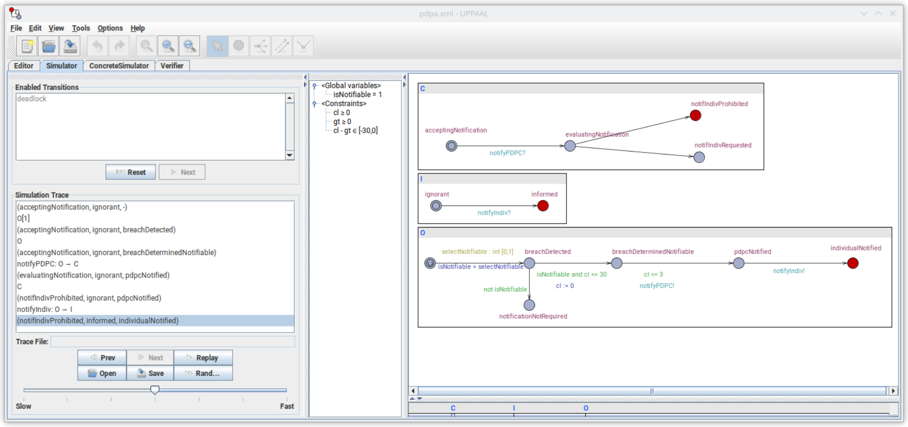
The Organization automaton is the most complex one: the very first transition is a modeling artifact for giving a random Boolean value to the variable `isNotifiable`. If the data breach is not notifiable, the process ends. If the breach is determined to be notifiable within 30 days, the timer is reset to allow for a 3 day period before the notification is carried out. We have here made the choice to sequentialize the notifications (the commission, then the individual). An interleaved execution would be preferable but is hampered by limitations of Uppaal (no nested parallel automata).

Let us note that our automaton model is not complete: not every execution of the automata will run to completion, *i.e.* wind up in one of the end nodes of the automata. Instead, an execution can get stuck in an intermediate state. This choice is deliberate, but open to debate: adding failure states for every undesirable run would clutter up the automata; some runs may legitimately be infinite without clearly identified end states; and, as seen below, such deadlock states can be detected.

3.2 Checking the Model

The model can be validated in several respects. The simplest form corresponds to testing in traditional program development. For this, Uppaal (and similar tools) provides a *simulation* environment whose purpose is to step through the model depending on user-selected criteria, for example the values of Boolean variables or the duration of staying in particular system states. We will not further dwell on simulation, also because error traces (see below) can be run in the simulator.

Contrasting with simulation is a systematic state space exploration by model checking. Model checking can be used in legal drafting mode, to discover incoherences in a law. It can also be used in "production" mode to identify illegal behaviors and possible contrary-to-duty repair actions. We will present examples of the two kinds below.



```
deadlock
```

```
<Global variables>
  isNotifiable = 1
<Constraints>
  cl ≥ 0
  gt ≥ 0
  cl - gt ∈ [-30,0]
```

```
(acceptingNotification, ignorant, -)
O[1]
(acceptingNotification, ignorant, breachDetected)
O
(acceptingNotification, ignorant, breachDeterminedNotifiable)
notifyPDPC: O → C
(evaluatingNotification, ignorant, pdpcNotified)
C
(notifIndivProhibited, ignorant, pdpcNotified)
notifyIndiv: O → I
(notifIndivProhibited, informed, individualNotified)
```

Fig. 2: Failure trace

Model checking tries to verify if a system's behavior conforms to requirements that are typically stated in a formal logic, here the temporal logic CTL. We give a few examples in our scenario:

– A desirable property is that it is possible to reach a state such that the individual is informed and the commission requests informing the individual. This property is written as

  ```
  E<>I.informed and C.notifIndivRequested
  ```

  Here, `E<>` means: "there exists a run eventually leading to". and `I.informed` and `C.notifIndivRequested` refer to the corresponding states in the Individual and Commission automata. This property is indeed satisfied.
– An undesirable situation is that the individual is informed in spite of the commission having prohibited it, expressed by the formula

  ```
  E<> I.informed and C.notifIndivProhibited
  ```

  . This property is also satisfied, and Uppaal produces a trace (Figure 2) leading to this error situation. It comes about because the organization has no clue at which point the commission's interdiction to inform the individual could intervene, and is therefore entitled to inform the individual as soon as a data breach is identified.
– Another undesirable behavior is when the execution of the process gets stuck, or, more precisely, when there is a deadlock, in a system state that is not perceived as final. Note that in a technical sense, according to the terminology of labelled transition systems, every system with final states is deadlocked in these states. Identifying deadlocks in "intermediate" states may contribute to finding leaks in the formal model, or states that correspond to breaches of the law. For example, the intermediate state `breachDeterminedNotifiable` is identified by the query

  ```
  E<> O.breachDeterminedNotifiable and deadlock
  ```

  as a deadlock state. Indeed, in this state, no action is possible when the notification deadline of 3 days has been exceeded.

4 Discussion: Extensions and Refinements

4.1 Refining the Model

The formal model could be perceived as "incomplete" in several respects. As shown by the formal analysis of Section 3, some behaviors are *undefined* (and lead to deadlocks), others are *under-specified*. In particular, a precise interaction between the commission and the organization is not made explicit in the law text, which might question the tenet of "isomorphism" [4] between law texts and formal representations: the formal representation possibly has to make completions to avoid undesirable situations.

One such completion would be the following: the commission communicates its decision back to the organization before the organization informs the individual. Such a provision is currently not foreseen in the law text. In order for the deadline of informing the individual (call it d_i) to be satisfied, the commission's response (call it d_c) would have to arrive before: $d_c < d_i$. These additional constraints can be incorporated into the automaton model and would avoid the above error scenario.

4.2 Realizability

Laws are requirements that constrain processes adopted by companies and organizations and that are often described by business processes [12].

We take an automaton such as in Figure 1c as an abstract specification of a business process. Depending on an organization's internal functioning, such an abstract description will be further refined, typically by adding new states (for example to guide the investigations to determine if a breach is notifiable). We will then have two automata, the abstract one (such as in Figure 1c) and its refinement (the business process, possibly specified in a dedicated language such as BPMN). Since tools such as Uppaal do not offer support for refinement checking, our project currently works on a language for describing refinements and for verifying them.

Of particular interest for the legal drafter in this context is the question of *realizability*: can laws be implemented under realistic conditions? To continue the example of Section 4.1: formally, the requirement $d_c < d_i$ on the deadlines is not a sufficient guarantee for realizability under realistic conditions: the difference $d_i - d_c$ can become arbitrarily small. Detecting such inconsistencies is part of our ongoing work.

4.3 Modularization through Rely-Guarantee Reasoning

The previous discussion highlights another desideratum: modularization. The organization can only satisfy requirements imposed on them when other actors (such as the commission) satisfy theirs. For separate refinement (in the sense of Section 4.2) to work, an (abstract) automaton should be annotated with requirements it expects its environment to provide (the *rely* part of its specification) in order to deliver the promise it makes about its behavior (the *guarantee* of the specification).

Without having clearly identified solutions, we are interested in exploring the duality of notions such as permissions (and *reliance* statements in a specification) and obligations (*guarantees*) known from deontic logics, notions that have been largely explored, among others, in LogiKEy [6].

A whole line of work is concerned with formal contracts [5], and in particular notions of modularization and refinement [2,8]. In a legal context, similar ideas have in particular been proposed as a temporal logic of normative systems [1].

Acknowledgements. This research / project is supported by the National Research Foundation, Singapore under its Industry Alignment Fund – Pre-positioning (IAF-PP) Funding Initiative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

References

1. Ågotnes, T., van der Hoek, W., Rodríguez-Aguilar, J.A., Sierra, C., Wooldridge, M.: A temporal logic of normative systems. In: Towards Mathematical Philosophy, pp. 69–106. Springer (2009)
2. de Alfaro, L., Henzinger, T.A.: Interface-based design. In: Broy, M., Gruenbauer, J., Harel, D., Hoare, C. (eds.) Engineering Theories of Software-intensive Systems. NATO Science Series, vol. 195, pp. 83–104. Springer (2005)
3. Azzopardi, S., Pace, G.J., Schapachnik, F., Schneider, G.: Contract automata: An operational view of contracts between interactive parties. Artificial Intelligence and Law **24**(3), 203–243 (September 2016). https://doi.org/10.1007/s10506-016-9185-2
4. Bench-Capon, T.J.M., Gordon, T.F.: Isomorphism and argumentation. In: The 12th International Conference on Artificial Intelligence and Law, Proceedings of the Conference, June 8-12, 2009, Barcelona, Spain. pp. 11–20. ACM (2009). https://doi.org/10.1145/1568234.1568237, https://doi.org/10.1145/1568234.1568237
5. Benveniste, A., Caillaud, B., Nickovic, D., Passerone, R., Raclet, J., Reinkemeier, P., Sangiovanni-Vincentelli, A.L., Damm, W., Henzinger, T.A., Larsen, K.G.: Contracts for system design. Found. Trends Electron. Des. Autom. **12**(2-3), 124–400 (2018). https://doi.org/10.1561/1000000053, https://doi.org/10.1561/1000000053
6. Benzmüller, C., Farjami, A., Fuenmayor, D., Meder, P., Parent, X., Steen, A., van der Torre, L., Zahoransky, V.: LogiKEy workbench: Deontic logics, logic combinations and expressive ethical and legal reasoning. Data in Brief **33**, 106409 (2020)
7. Flood, M.D., Goodenough, O.R.: Contract as automaton: representing a simple financial agreement in computational form. Artif. Intell. Law **30**(3), 391–416 (2022). https://doi.org/10.1007/s10506-021-09300-9, https://doi.org/10.1007/s10506-021-09300-9
8. Grumberg, O., Long, D.E.: Model checking and modular verification. ACM Trans. Program. Lang. Syst. **16**(3), 843–871 (1994). https://doi.org/10.1145/177492.177725, https://doi.org/10.1145/177492.177725
9. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. International journal on software tools for technology transfer **1**(1), 134–152 (1997), http://user.it.uu.se/~yi/pdf-files/2017/nutshell.pdf
10. Parvizimosaed, A., Roveri, M., Rasti, A., Amyot, D., Logrippo, L., Mylopoulos, J.: Model-checking legal contracts with symboleopc. In: Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems. pp. 278–288 (2022)
11. Singapore Statutes Online: Personal data protection act (2012), https://sso.agc.gov.sg/Act/PDPA2012
12. Van der Aalst, W.M.: Business process management: a comprehensive survey. International Scholarly Research Notices **2013** (2013), http://www.padsweb.rwth-aachen.de/wvdaalst/publications/p712.pdf