Timed Automata for Security of Real-Time Systems

Vasilikos, Panagiotis

Publication date:
2019

Document Version
Publisher's PDF, also known as Version of record

Citation (APA):
Vasilikos, P. (2019). *Timed Automata for Security of Real-Time Systems*. Technical University of Denmark.

Timed Automata for Security of Real-Time Systems

Panagiotis Vasilikos

Kongens Lyngby 2019
ISSN 0909-3192

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Matematiktorvet, building 303B,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3351
compute@compute.dtu.dk
www.compute.dtu.dk

Summary (English)

Ensuring information security is a fundamental problem in computing environments of modern society. Information flow theory provides key techniques for guaranteeing that certain information security goals are met by a computing system. However, the focus of the researchers has been mainly around the security of programming languages, whereas, little effort has been put in the security analysis of cyber-physical systems (CPS) and internet of things (IoT) devices. CPS and IoT involve computations which have unstructured control flow and can be both data- and continuous time-dependent, making the programming language approaches inadequate. This thesis makes a breakthrough to the problem of information security in such systems.

To this end, we model systems using timed automata, a formalism that has established itself for analyzing safety properties of CPS and IoT devices. We then leverage approaches from the theory of information flow, and we develop novel techniques for both the qualitative, and the quantitative security analysis of our models.

Our main contributions include

- the development of a language, whose semantics is given using timed automata, and a sound type system, which enforces a non-interference security condition on programs written in our language.
- a sound algorithm which traverses a timed automaton and enforces a non-interference condition, which permits locality-based information release.
- a logic for the specification of time- and data-dependent access control policies for networks of timed automata, and techniques for translating policies in our

logic to a logic which can be handled by standard model-checkers for timed automata such as UPPAAL. We also provide an implementation of our translation.

- the first principled information-flow analysis of information leakage on systems that implement the countermeasure clocks with reduced resolution. Our analysis relies on a novel translation of timed automata to information-theoretic channels, which we then use to derive insights into the effectiveness of this countermeasure and the existing attacks that can bypass it.

Summary (Danish)

At sikre informationssikkerhed er et grundlæggende problem i computermiljøer i det moderne samfund. Informationsteorien indeholder nøgleteknikker til at garantere, at visse informationssikkerhedsmål opfyldes af et computersystem. Forskernes fokus har dog primært været omkring sikkerheden ved programmeringssprog, hvorimod der er gjort en lille indsats i sikkerhedsanalysen af cyber-fysiske systemer (CPS) og internet of things- (IoT) enheder. CPS og IoT involverer beregninger, der har ustruktureret kontrolstruktur og kan være både afhængig af data og kontinuerlig tid, hvilket gør programmeringssprogtilgangene utilstrækkelige. Denne afhandling gør et gennembrud i problemet med informationssikkerhed i sådanne systemer.

Med henblik herpå modellerer vi systemer ved hjælp af tidsautomater, en formalisme, der er etableret til at analysere sikkerhedsegenskaber for CPS og IoT-enheder. Vi udnytter derefter tilgange fra informationsteorien, og vi udvikler nye teknikker til både den kvalitative og den kvantitative sikkerhedsanalyse af vores modeller.

Vores vigtigste bidrag inkluderer

- udviklingen af et sprog, hvis semantik er givet ved hjælp af tidsautomater, og et korrekt typesystem, der håndhæver en ikke-interferens-sikkerhedsbetingelse på programmer skrevet i vores sprog.
- en korrekt algoritme, der gennemløber en tidsautomat og håndhæver en ikke-interferensbetingelse, som tillader lokalitetsbaseret informationsfrigivelse.
- en logik til specifikation af tids- og datafhængige adgangskontrolpolitikker for netværk med tidsautomater, og teknikker til at oversætte politikker i vores logik

til en logik, der kan håndteres af standardmodeltjekkere for tidsautomater som UPPAAL. Vi leverer også en implementering af vores oversættelse.

- den første systematiske information flow-analyse af informationslækage på systemer, der implementerer modforanstaltningsurene med reduceret opløsning. Vores analyse er afhængig af en ny oversættelse af tidsautomater til informationsteoretiske kanaler, som vi derefter bruger til at udlede indsigt i effektiviteten af denne modforanstaltning og de eksisterende angreb, der kan omgå den.

Preface

This thesis was prepared at the department of Applied Mathematics and Computer Science at the Technical University of Denmark in fulfillment of the requirements for acquiring a PhD degree in Computer Science.

The research has been performed under the supervision of Professor Hanne Riis Nielson and Professor Flemming Nielson in the period from September 2016 to August 2019.

The core of the developments presented in this thesis are based on four published papers. In particular, Chapter 3, Chapter 4, Chapter 5 and Chapter 6 are based on the publications of [NNV17], [VNN18], [VNN17] and [VNNK] respectively.

Lyngby, 31-August-2019

Panagiotis Vasilikos

[illegible]

Acknowledgements

I am especially grateful to my supervisors Hanne Riis Nielson and Flemming Nielson for their constant support, guidance and teaching through those years. Thanks to them, I am the researcher who I am today, and without them, this thesis would have not been possible. I would also like to thank Boris Köpf for being my supervisor during my external research stay at IMDEA Software. His work on quantitative information flow has been really inspiring, and our collaboration has been beneficial for my career.

I would also like to thank my assessment committee Michael Reichhardt Hansen, Luca Vigano and Luca Aceto for their valuable comments on this work.

Being part of the formal methods section at DTU has been a rewarding and fun experience. I am thankful to all the people who shared the office, had lunches and discussions with me. My special thanks go to Alberto Lluch Lafuente for his priceless support.

On a personal note, I want to thank all of my friends in Copenhagen, and especially my girlfriend Sonia for our special moments, which made me forget and overcome my research frustrations. Most importantly, I want to thank my sister Sofia and my parents Pantazis and Kariofyllia, for their love, constant support and always believing in me.

Contents

CHAPTER 1

Introduction

Information security is essential to the privacy and safety of modern society. Over the last few years, cyber-attacks have successfully compromised information such as credit card numbers, login credentials, and personal data, leading to significant financial loss and privacy violations of companies and individuals [ATT19]. More importantly, other cyber-attacks have demonstrated that information can be manipulated in such a way that could result in catastrophic explosions at a petrochemical plant [SAU19], or bring down the power and heat in a war-torn country, during winter [LJC16].

Although the task of information security is vital, the exponential growth of technology makes it complicated and daunting. In particular, latest technology trends have introduced the internet of things (IoT) and cyber-physical systems (CPS) to automate many commercial and industrial applications respectively. Both IoT and CPS involve increasingly distributed and interconnected devices, such as sensors, actuators, controllers and processors, whose functionality is complex, and highly dependent on both discrete and continuous variables, such as *data* and *time*.

Those variables may introduce *information channels* that could allow an adversary to compromise sensitive information. For instance, consider a scenario of a distributed system that consists of a sensor and a controller. The sensor continually computes some data and communicates it to the controller. For ensuring data integrity, the sensor always encrypts (signs) the data with its private key. Consider also an adversary who is able to sniff messages sent on the link used by the sensor and the controller, and can also measure the delay between the captured messages. A faulty implementation of the encryption algorithm could introduce vast information flows from the sensor's private key to the observations of the adversary, that is the time of the sniffed messages and the data in them. Consequently, the adversary could infer the private key of the sensor, impersonate it, and then deliver false information to the controller, taking full control of the entire system.

Information flow theory [Sha01, VSI96, CT06] studies the way information flows between different variables of a system. There exist two main approaches in information flow theory, the *qualitative* and the *quantitative* ones.

In the qualitative approach, *information flow security policies* are used to specify the desired information flows in a system. For instance, for the example of the sensor and the controller, a security policy could specify that there should be no flow of information between the private key of the sensor and the observations of the adversary. Next, a *security condition* describes formally when a system fulfills the security policy, and an *information flow control mechanism* enforces the security policy on the system, proving that the security condition is satisfied.

In the quantitative approach, mathematical techniques are used to calculate the exact amount of information flowing between the variables of the system. Going back to the example of the sensor and the controller, if the sensor's key is 1024-bit long, then one could calculate the information leaked from the key to the observations of the adversary. For instance, a 2-bit leak is tolerable; however, if 1000 bits are leaked, then the implementation of the encryption should be refined. The amount of information leakage is often calculated with the use of an information-theoretic measure called *entropy* [Sha01, ACPS12, CT06, Smi09].

In the field of programming languages, information flow theory has been a fundamental approach to ensure the confidentiality and integrity of information. In particular, for the qualitative approach, there is a significant body of work from the literature that allows one to express complex information security policies, which can be data-dependent [LNN15, LNNF15], and permit intentional information leakage [SS09, AS07, GLMS14, MSZ04, MSZ06, ML97, ZM01]. One approach to enforce security policies is *access control*, which however, only protects against explicit flows between variables (e.g reading or writing to a variable). Another approach is with the use of *non-interference* [VSI96, SM03] style security conditions, which requires the absence of flows between the sensitive variables of the system and the ones that could be manipulated by or observed from the adversary. Finally, a common information flow mechanism for enforcing non-interference is via a *type-system* [VSI96, SM03, Aga00]. The quantitative approach has found many applications in measuring the amount of information leaked through the timing behaviour of a program [DFK$^+$13, KB07, BK15, MKP$^+$18], and also in evaluating the effectiveness of countermeasures deployed to limit such information leaks [CRS83, KD09, ZAM11].

Unfortunately, many of those approaches are not suitable for the security analysis of IoT and CPS. In particular, most of the programming language approaches [Aga00, KD09, KB07, PHW08, BK15] model time as a discrete variable. On the one hand, this simplification makes the analysis of information flow easier, since time is treated as any other discrete data variable of the system. However, as illustrated recently in [BP18], this simplification can introduce some false sense of security, allowing for some information flows to be undetected. Another limitation of most programming language approaches is that they allow only structured control flow, which is not always suited for the modelling of systems such as IoT and CPS. We show the latter with an example in Chapter 4.

Contributions This thesis contributes to the problem of information security in systems such as IoT and CPS. To this end, we model systems as *timed automata* [AILS07, AD94], a formalism that has established itself for analyzing safety properties of systems with hard *real-time* constraints. We then study the problem of information security in the following contexts: (a) detection of information leakage through information channels, constructed using both the data and the timing properties of the system, (b) specification and enforcement of security policies which permit intentional information release, (c) specification and enforcement of data- and time-dependent security policies, and (d) quantification of the effectiveness of countermeasures against information leakage through information channels constructed by measuring the timing properties of the system.

In addressing (a), we take a language-based approach by defining the language of timed commands, whose semantics is given using timed automata. We then develop a type system, and we prove that type-checked commands leak no information under adversaries that can observe both some of the data and the execution time of the command. The latter result is formulated via a non-interference theorem.

In addressing (b), we define locality-based security policies, which allow intentional information release at specific locations of the automaton. We then develop an algorithm which certifies a timed automaton with respect to a security policy. Certified automata are proved to satisfy the security policy, and we formulate this via a bisimulation-based relaxed non-interference.

In addressing (c), we present a logic in which one can express data- and time-dependent security policies for access control on networks of timed automata. We show how a fragment of our logic can be reduced to a logic that current model checkers for timed automata such as UPPAAL [UPP] can handle, and we implement a translator that performs this reduction. We then show how the enforcement of the policies can be expressed as a reachability problem, and consequently checked by UPPAAL [UPP].

In addressing (d), we present the first information flow analysis of the countermeasure of reducing clock resolution. Our analysis is based on translations of timed automata models to information-theoretic channels, and using it we achieve the following: (1) we show that contrary to the popular belief a coarse-grained clock might leak more than a fine-grained one, (2) we give sufficient conditions for when increasing the granularity of the clock we achieve less information leakage, and (3) we show that the attack techniques used to bypass this countermeasure form a strict hierarchy in terms of the information an adversary can extract using them.

Our main contributions to (a), (b), (c), and (d) have been published in the following papers

- (a) Information flow for timed automata by Flemming Nielson, Hanne Riis Nielson and Panagiotis Vasilikos. *Appeared in Models, Algorithms, Logics and Tools - Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday.* [NNV17]
- (b) Secure information release in timed automata by Panagiotis Vasilikos, Flemming Nielson and Hanne Riis Nielson. *Appeared in Principles of Security and Trust - 7th International Conference, POST 2018.* [VNN18]
- (c) Time dependent policy-based access control by Panagiotis Vasilikos, Flemming Nielson and Hanne Riis Nielson. *Appeared in 24th International Symposium on Temporal Representation and Reasoning, TIME 2017.* [VNN17]
- (d) Timing leaks and coarse-grained clocks by Panagiotis Vasilikos, Flemming Nielson, Hanne Riis Nielson and Boris Köepf. *Appeared in 32nd IEEE Computer Security Foundations Symposium, CSF 2019.* [VNNK]

Thesis Organization In Chapter 2, we recall information security notions from the theory of information flow, which we later extend to our developments for timed automata. We also present some motivating examples which will be analysed in following chapters. Next, Chapters 3, 4, 5, and 6 present our contributions to the above-mentioned problems (a), (b), (c) and (d) respectively. Chapter 7 discusses our conclusions, and Appendix A, B, C and D include proofs of the results presented in the Chapters 3, 4, 5 and 6 respectively. Finally, in Appendix E we give detailed calculations for a case study presented in Chapter 6.

CHAPTER 2

Information Flow Theory

Formalisms of information security developed in this dissertation build on certain notions of *information flow theory*, which have been widely studied in language-based settings. Information flow theory studies the way information flows between the variables in a system. There exist two main approaches of information flow theory – the qualitative and the quantitative.

The qualitative approach consists of three main steps: (1) the specification of *security policies*, which precisely describe the desired flows in a system, (2) a *security condition* that defines when a system respects a security policy, and (3) an *information flow control mechanism* that checks the system against the security condition. Main developments of the qualitative approach include (a) *access control*, which protects against direct flows in a system (e.g reading or writing on a file), (b) *non-interference*, which requires no flow of information from sensitive to public variables of the system, and has been the standard approach to the defeat of *covert channels*, and finally, (c) *declassification*, which is a relaxed version of non-interference, permitting conditional information flows.

On the other hand, in the quantitative approach, mathematical techniques are used to calculate the exact amount of information flowing between the variables of the system. Those techniques are usually based on a measure called *entropy*. Main entropy definitions include *Shannon-entropy*, *Min-entropy*, and *g-entropy*.

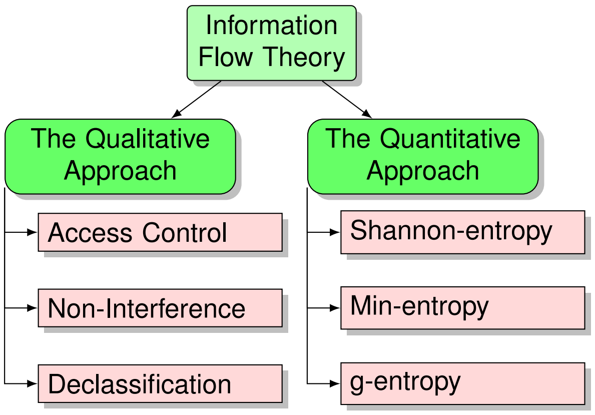


Figure 2.1: Main approaches of information flow theory.

In this Chapter, we describe and compare the qualitative and quantitative approaches in more detail. We also give motivating examples that will be analysed later using our developments.

Chapter organization We start with the qualitative approach and in Section 2.1, 2.2, and 2.3 we discuss the access control, non-interference, and declassification developments respectively. Finally, in Section 2.4 we discuss the quantitative approach.

2.1 Access Control

Historically, *access control policies* [dVSS14] have been the standard way for defining the desired information flows in a system. They do that by specifying who and how one can access data and resources. The security condition in access control specifies that a direct access or an *explicit flow* of information should occur only when it is allowed by the access control policy. Access requests to resources or data by users or processes are then either denied, or authorized by a *monitor* [Fag78] that enforces the policies.

The literature offers a vast number of access control models, where among all, the most used in practice are the *discretionary access control* (DAC) [Lam74, BSJ93], *mandatory access control* (MAC) [Thu09], and *role-based access control* (RBAC) [OSM00], while lately, great attention has been given to the *attribute-based access control* (ABAC) model.

Discretionary access control is one of the oldest forms of access control and was first defined by Lampson [Lam74] in 1970. The core idea in DAC is that a system is defined by a set of subjects S (e.g the users of the system or the processes which run on behalf of them) a set of objects O (e.g files, sockets etc.) and a set of permissions R (e.g read, write etc.). An access control policy $\mathcal{L} : S \times O \rightarrow \mathbb{P}(R)$ defines the set of permissions a subject has on an object. For example, if u is a user of the system, and f a file of the system, then a policy $\mathcal{L}$ such that $\mathcal{L}(u, f) = \{\text{read}\}$ specifies that the user u can only read the file f. Any other type of action on f by u will be denied. DAC policies are known for their simplicity and flexibility, which has made them applicable to a variety of systems, both commercial and industrial. However, they are also known to be weak on the information flow constraints that they put in a system. In particular, once access has been granted, they do not impose any restrictions on subsequent use of an object by a subject. For example, a trojan-horse process p running on behalf of a fully privileged user u could maliciously copy contents from a restricted-read permission file f_1 (e.g only u can read f_1) to another file f_2, which could be read by any other user. This problem raised the need for a new model of access control policies, and a solution was given by the mandatory access control (MAC) model.

Mandatory access control policies [Thu09] constrain also manipulations of objects that can occur inside the execution of a process. Those constraints are expressed with the use of a security lattice $(L, \sqsubseteq)$, whose elements are security clearance levels, and the ordering $\sqsubseteq$ represents the desired flows between them. This lattice-based access control model was first introduced by Denning [Den76]. A security policy $\mathcal{L} : S \cup O \rightarrow L$ is a mapping that assigns a security level to each one of the subjects and objects of the system. For instance, going back to our trojan-horse example, let $L = \{\mathtt{L}, \mathtt{H}\}$ be the set of security clearances, with only two elements, low $\mathtt{L}$, and high $\mathtt{H}$, and with $\mathtt{L} \sqsubseteq \mathtt{H}$. The latter means that a flow of information is allowed only from the low to the high security level. The security policy $\mathcal{L} = [p \mapsto \mathtt{H}, f_1 \mapsto \mathtt{H}, f_2 \mapsto \mathtt{L}]$ would have disallowed p to copy the files of f_1 to f_2, since $\mathcal{L}(f_1) = \mathtt{H} \not\sqsubseteq \mathtt{L} = \mathcal{L}(f_2)$. MAC policies have found their way in military systems and intelligence agencies; however, they are not so appealing for commercial applications, since under certain conditions a user may desire to declassify his data to a lower security level.

Role-based access control (RBAC) is a much broader model of access control, allowing one to express both DAC and MAC policies [OSM00]. The core of this model is based on a set of roles which are assigned to a set of permissions. Users are then assigned to a particular role. This simple model implements DAC policies on the role's level using the set of permissions, while by defining a hierarchy on the roles, one can also specify MAC policies. In the context of an organization, roles describe job-specific functions, and role permissions determine the permitted actions or tasks a role should have in order to do its job. For example, a doctor in a hospital will be assigned to the role Doctor, and this role may have permissions such as writing, modifying prescriptions and reading patient's medical records. If now there exists another role for the nurses of the hospital, and assume that this role is Nurse, we can define a hierarchy between them, by allowing the role Nurse only to read patients prescriptions and nothing more. Stan-

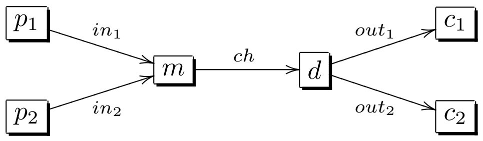


Figure 2.2: The processes and channels of the gateway example.

dard RBAC models are very intuitive and flexible when it comes to role administrator, and thus they have been widely accepted by many commercial organizations; however, they are not suitable for specifying dynamic policies.

Attribute-based access control (ABAC) [MS08, BBF01, MSA11, JBLG05, GBO12, CWW$^+$10, PSL$^+$15] is a broader policy model that offers the flexibility of data declassification and dynamic policy specification. Policies in ABAC are specified with the use of conditions which talk about attributes of the system's environment. Such attributes are the current time, the location of the subject that tries to perform an access, the contents of the object the subject is trying to have access etc. Going back to the example of the hospital, with the use of an ABAC model one can express that doctors can have access to the medical history of a patient only during a certain period of the patient's treatment and not after (i.e here, the attribute of interest is time). In addition to that, a doctor may have no access to any resource when it is not in the hospital (i.e here, the attribute of interest is location). The expressive power of ABAC models allows for the specification of real-world policies; ABAC is more general and flexible than DAC, MAC, and RBAC.

The work presented in Chapter 5 deals with the problem of expressing and enforcing ABAC policies in distributed real-time systems of processes. In particular, in our context, the subjects will be processes, objects will be data variables, while the attributes of interest are the content of the data that is communicated between the processes and the current system's time.

This work is motivated by the world of avionics software in distributed real-time computer networks consisting of processing modules [MPTB12]. Every processing module hosts different software functions that share computing resources (e.g. I/O, execution time) that need to be partitioned based on the content of some variables or the time of the system. The separation of the resources is essential in order to ensure that untrusted processes such as passenger devices can have access to on-board communication systems, without alerting the safe operation of the aircraft. The separation is then implemented by a kernel which is used as a monitor that authorizes the accesses of the processes based on data- and time-dependent security policies.

EXAMPLE 2.1 *Such a system is depicted in Figure 2.2. A gateway with two processes p_1, p_2 called the producers, each of them producing data for different targets c_1 and c_2*

resp., called the consumers. The gateway uses a multiplexer m, and a demultiplexer d to successfully deliver the data to the intended target. The data is communicated through channels whose names label the edges in Figure 2.2.

The security goal of the gateway is that the producers p_1, p_2 talk only with the consumers c_1, c_2 resp., and this happen only within certain time intervals. We will see more details of the example in Chapter 5.

As we have seen, access control offers a variety of models, which allow one to express, and enforce rich security policies, meeting the needs of industry, military and any modern organization of today. However, poor system implementations allow for unintended transfer of information. This could happen even without the use of common operations such as read or write, resulting in violation of security policies, which becomes undetected by the access control monitor. This leads to the development of new ways and methods of enforcing security policies which we explore in the next section.

2.2 Non-Interference

Unintended information flows can occur whenever a system builds a *covert channel. Covert channels* are mechanisms used to transfer information via the control-flow of a system, its termination behaviour, its execution time, power consumption, temperature etc. Information flows arising from covert channels are called *implicit flows*. Covert channels are not intended by the design of the system, and thus, they are hard to detect. We now give some examples of covert channels which are investigated in this thesis.

Control-flow covert channels are one of the most common types of covert channels. They occur whenever a system's control flow depends on secret data. For example, consider the program `if y > 0 x := 1 else x := 0`. If an adversary observes the final value of `x` he deduces partial information (i.e that `y` is positive when `x` is 1 and that `y` is not positive when `x` is 0) of the variable `y` without explicitly reading it. This kind of covert channel has been extensively studied in the literature, with Denning [DD77] being the first to formally specify when a program is free of control-flow covert channels.

Systems with non-deterministic semantics give rise to new control-flow covert channels. Such covert channels are particularly interesting for the kind of systems we study in this thesis.

Example 2.2 *To see an example of such a covert channel consider a Dijkstra's* Guarded Command *[Dij75] like language, which allows for statements of the form*

$\mathtt{g} \rightarrow \mathtt{C}$, *where the command* C *can be executed when the guard* g *evaluates to true. Next, consider the program* $\mathtt{x} := \mathtt{0}; (\ \mathtt{y} > \mathtt{0} \rightarrow \mathtt{skip}\ [] \ \mathtt{tt} \rightarrow \mathtt{x} := \mathtt{1})$, *which first sets the value of* x *to 0 and then makes a non-deterministic choice which may modify the value of* x. *Notice that, even if the variable* x *does not appear in the branch which depends on the guard* $\mathtt{y} > \mathtt{0}$, *the value of* x *is still dependent on* y. *In particular, observing that the final value of* x *is not* 1 *allows the adversary to deduce that* y *is positive.*

This kind of implicit flow cannot be detected with the approach of [BBM94], which deals with information flows in a guarded command-like language. In Chapter 3, we show how we can detect those kind of flows.

Another common type of a covert channel is a timing channel. A *timing channel* transfers secret information through the execution-time of the system. In particular, the information conveyed by timing channels has been used by adversaries to recover cryptographic keys, where the timing channel is built by measuring cryptographic or cache-dependent operations, and by malicious websites which correlate this information with the internal state of a victim who visits the website [Koc96, SWT01, BT11, VK17, OKSK15, FS00, AKM+15].

EXAMPLE 2.3 *As an example of a system that builds a timing channel consider a program that implements the RSA encryption algorithm using the modular exponentiation, which computes* $\mathtt{x}^{\mathtt{k}} \bmod \mathtt{n}$ *for the secret key* k, *the plaintext* x *and the constant modulus* n. *The implementation of* $\mathtt{x}^{\mathtt{k}} \bmod \mathtt{n}$ *is given by the following piece of code*

```
m := (1 * 1) mod n;
for (j = 0; j < len(k); j++) {
    m := (m * m) mod n;
    if (k[j] == 1) then
      m := (m * x) mod n;
}
```

where the secret bits of the key are stored in the array k[]. *Next, consider an abstract model of this program, whose execution time* t *is given based on the number of modular multiplications it performs. Now if an adversary observes* t, *we have a timing channel since the conditional execution (*$\mathtt{if}\ \mathtt{k[j]} == \mathtt{1}$*) of the modular multiplication operation* $\mathtt{m} = (\mathtt{m} * \mathtt{x}) \bmod \mathtt{n}$ *reveals information about the entries of k (i.e one extra modular multiplication is performed when* k[j] *is 1).*

Non-interference is the prevailing security condition that defeats completely covert channels, and was first introduced by Goguen and Meseguer [GM82]. The non-interference definition of [GM82] is fundamentally about higher-level processes not interfering with lower-level processes of the system. In a more general context, non-interference is usually expressed with a lattice-based access control policy, (as we saw in the mandatory access control policies model), and requires that entities of the system that have security classification that appears higher in the lattice do not interfere with the ones that have a lower security classification.

To give a better understanding of the non-interference condition, we consider a program P which takes as input some *secret* from the set I and produces a *public* output from the set O. Consider also the security lattice $L = \{\mathtt{L}, \mathtt{H}\}$ from the MAC policies example, and let all the elements from the set I have $\mathtt{H}$ security level, and all the ones from O have $\mathtt{L}$ security level. We want to ensure that running the program P on some secret input does not produce an information flow from the secret input to its corresponding output, or to put it otherwise, we want that the set of inputs does not interfere with the set of outputs.

If now the program P is deterministic it can be seen as a function $\mathsf{P} : I \mapsto O$ mapping inputs to outputs. The non-interference property is formally expressed as

$$\forall i_1, i_2 \in I : \mathsf{P}(i_1) = \mathsf{P}(i_2)$$

that is that independently of the secret input, the observation to the adversary is the same, and thus there is no interference between the secret inputs and the public outputs.

EXAMPLE 2.4 *Consider now the modular exponentiation program from Example 2.3. If we fix the message* $\mathtt{x}$ *then the program can be seen as a function* $\mathsf{P} : K \mapsto T$ *which maps a secret key* $\mathtt{k} \in K$ *to its execution time* $\mathtt{t} \in T$*. In particular, assume that* K *contains all the possible keys of size 1024-bits, and for a key* $\mathtt{k} \in K$ *we have that its execution time is* $P(\mathtt{k}) = 1025 + \mathsf{Ham}(\mathtt{k})$*, where* $\mathsf{Ham}(\mathtt{k})$ *is the Hamming weight of the key (i.e the number of non-zero bits). This program does not satisfy the non-interference property between the secret input set of keys* K*, and the output set of execution times* T*. To see this, take the key* $\mathtt{k}_0$ *with all bits equal to* 0*, and we have that* $\mathsf{P}(\mathtt{k}_0) = 1025 \neq 2049 = \mathtt{P}(\mathtt{k}_1)$*, where* $\mathtt{k}_1$ *is the key with all bits equal to 1.*

If now the program P is non-deterministic, the program is not a function from inputs to outputs anymore. To define the non-interference condition, let $\mathsf{P}(i) \subseteq O$ be the set of possible outputs of the input $i \in I$. In this case, the non-interference condition can be described by the following condition

$$\forall i_1, i_2 \in I : \mathsf{P}(i_1) \subseteq \mathsf{P}(i_2)$$

which by symmetry implies that $\mathsf{P}(i_1) = \mathsf{P}(i_2)$ as above.

EXAMPLE 2.5 *Let now* P *be the program of the non-deterministic guarded command program of Example 2.2. Let also* $Y = \mathbb{Z}$ *be the input set of the program and let*

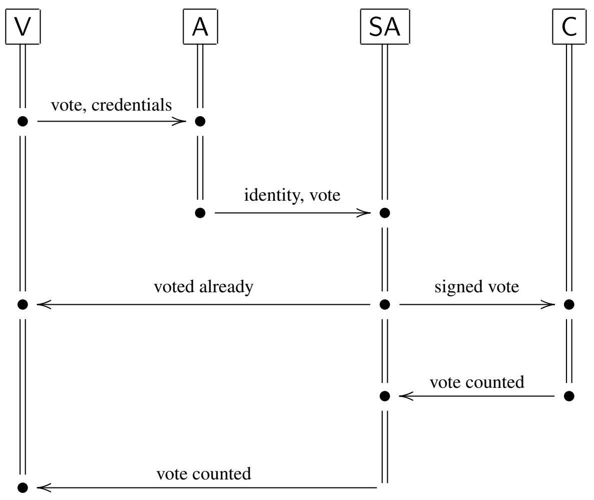


Figure 2.3: A diagram that illustrates the messages exchanged between the voter V, the authenticator A, the signing authority SA and the counting mechanism C.

$X = \{0, 1\}$ *be the output set of the program. We then have that for* $\mathrm{y} \in Y : y \leq 0 \Rightarrow P(\mathrm{y}) = \{1\}$*, whereas for* $\mathrm{y} \in Y : \mathrm{y} > 0 \Rightarrow P(\mathrm{y}) = \{0, 1\}$ *and thus the input set* Y *is interfering with the output set* X.

Non-interference conditions have also been formulated for programs with probabilistic semantics [III90, McL90, SI95, SS00], concurrency [SV98, BBM95, BB93, SS00], and communication [LNNF15, LNN15]. For a literature review of non-interference notions we refer to [SM03].

In Chapter 3, we deal with the problem of enforcing non-interference in systems with real-time semantics and adversaries who observe both the execution time of the system and some public output upon termination.

EXAMPLE 2.6 *To illustrate our development in Chapter 3, we consider a voting system in which votes are signed by an authority to ensure their authenticity.*

The system consists of n *voters* V_1*,...,*V_n *and three parties, an authentication mechanism* A*, a signing authority* SA *and a counting mechanism* C*. The voting protocol of the system is described as follows: a voter first authenticates itself to* A*. If the authentication succeeds,* A *forwards the vote and a unique id for the voter to* SA*.* SA *checks if the voter has already voted, and in this case, it sends a message, notifying him that* *he has already voted; otherwise, it signs the vote and forwards it to* C. C *then counts the vote, informs* SA *about it, and subsequently* SA *notifies the voter about it. The messages exchanged by the different parties of the system are depicted in Figure 2.3.*

For signing the votes, the signing authority SA *uses an implementation of the RSA algorithm similar to the one given in the Example 2.3. The security goal of the system is to ensure that whenever a voter observes the messages received from* SA*, and their arrival time, he cannot infer anything about the RSA key used by* SA.

Non-interference provide strong guarantees that no information is leaked in case of confidentiality, and that untrusted data does not influence trusted data in case of integrity. However, many real systems need to allow some interference in order to achieve their purposes. This need leads to the development of relaxed notions of non-interference which permit data declassification.

2.3 Declassification

Without the possibility of leaking some information some systems would have not been useful. For example, at the end of an election a voting protocol reveals the sum of all votes, a password authentication mechanism reveals some information of the password in case of a failed log-in attempt, and the medical history of the patient will become public to the doctor in case of the patient getting a disease. Systems with intentional leak lead to the development of security policies which allow *data declassification*. Security policies for data declassification are partitioned in three main categories based on *what* data can be declassified, *who* can declassify data, and *where* data can be declassified. We explain those categories in the context of confidentiality.

What data can be declassified? Property-based information flow policies [SS09, AS07, GLMS14] control *what* information or property of the secret can be deduced by an adversary. One way of expressing property-based information flow policies is with the use of equivalence relations [SS09]. This approach generalizes non-interference that requires that any two secrets must be indistinguishable to the adversary, and only demands that two secrets are indistinguishable to the adversary if they have the same property specified by the security policy.

Example 2.7 *For instance, for the RSA program* $\mathsf{P} : K \mapsto T$ *of the Example 2.4, a property-based policy could be defined by the equivalence relation*

$$\equiv_{\mathsf{Ham}} = \{(\mathtt{k}_1, \mathtt{k}_2) \mid \mathtt{k}_1,\ \mathtt{k}_2 \in K : \ \mathsf{Ham}(\mathtt{k}_1) = \mathsf{Ham}(\mathtt{k}_2)\}$$

which specifies that two keys should be indistinguishable to the adversary if and only if they have the same Hamming weight. Next, notice that equivalent keys under $\equiv_{\mathsf{Ham}}$

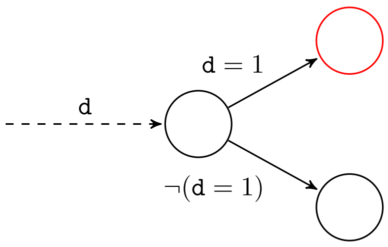


Figure 2.4: The part of the smart grid system, where it receives the declassification decision d and based on that decides if it should declassify data or not. Information should be declassified only when the red state is reached.

result in the same execution time in P *and thus they are indistinguishable to the adversary. Therefore* P *satisfies the security policy* $\equiv_{\mathsf{Ham}}$.

Who can declassify data? Ownership-based information flow policies [SS09, MSZ04, MSZ06, ML97] are essential in many applications that require control over *who* can declassify data. The most widely accepted framework that allows expressing such policies is the *decentralized label model* [ML97], where data is annotated with ownership labels. Declassification of some data is then only allowed if it is performed from the owner indicated in the label. The decentralized label model has been implemented in the Jif compiler [MZZ$^+$06], which is used to enforce information flow policies for Java programs. One open issue with this formalism was that there was no formalization of a semantic condition that proves that an adversary cannot influence the declassification. Later work of [MSZ06, ZM01] solved this issue by introducing the notion of *robust-declassification*.

Where can data be declassified? Locality-based information flow policies [AS07, GLMS14, MS04, Pin95, RG99] describe *wherein* the system (or program) data declassification is allowed. In particular, components of the system, or code fragments of the program are annotated with declassification labels. Then, the security policy specifies that reaching such a labelled component, or executing a labelled code fragment may result in some information leakage. The most standard semantic notion for enforcing locality-based information flow policies is the one of *intransitive non-interference* [MS04, Pin95, RG99]. For a system or program that satisfies intransitive non-interference, it is guaranteed that only labelled components or code fragments declassify data.

The work in Chapter 4 considers the problem of enforcing locality-based policies on real-time systems. The work is motivated by data declassification problems in smart power grid systems. In its very basic form, a smart grid system consists of a meter that measures the electricity consumption in a customer's C house, and then sends this data to the utility company UC. The detailed measurements of the meter provide more accurate billings for UC, while C receives energy management plans that optimize his energy consumption. Although this setting seems to be beneficial for both UC and C, it has been shown that high-frequent monitoring of the power flow poses a major threat to the privacy of C [SF15, GA17, MR17]. To deal with this problem many smart grid systems introduce a trusted third-party TTP, on which both UC and C agree on [SF15]. Now, the data of the meter is collected by the TTP, and by the end of each month, the TTP charges C depending on the tariff prices defined by UC. In this protocol, UC trusts TTP for the accurate billing of C, while C trusts TTP with its sensitive data. However, in some cases, C may desire an energy management plan by UC, and consequently, he makes a clear statement to TTP, allowing the latter to release its private data to UC.

Example 2.8 *Figure 2.4 illustrates part of the smart grid system that we will see later in Chapter 4. In particular, Figure 2.4 depicts the part of the system where the trusted party* TTP *receives the decision* d *of the client* C*, and based on that it moves on its next location, deciding if data should be declassified.*

The locality-based policy that we wish to enforce here is that information is released only when the system reaches the red location in Figure 2.4.

Security policies that allow data declassification offer a flexible way to overcome the strictness of non-interference conditions, and for this reason they are more appealing in real-world applications. However, their drawback is that sometimes it is difficult to understand the implication of a declassification for the security of a system. For example, a declassification may reveal some property of the secret, but this does not say if the leakage is big or small with respect to the size of the secret, resulting in a situation where an adversary could infer the entire secret based on its property. *Quantitative information flow* provides mathematical mechanisms for dealing with such cases.

2.4 Quantitative Information Flow

Quantitative information flow studies the problem of measuring the correlation between the secret components and the observable ones in a system. The quantity of correlation is usually calculated based on a mathematical measure called *entropy*. Information entropy measures are used to describe the *initial uncertainty* the adversary has regarding the secret, and his *posterior uncertainty* or *remaining uncertainty* after making his observations. In particular, the initial uncertainty is calculated based on a probability distribution on the secret, while the posterior uncertainty is calculated based on the information channel of the adversary. Formally, the information channel is defined as a matrix, and for each secret and observation it contains the probability of the observation

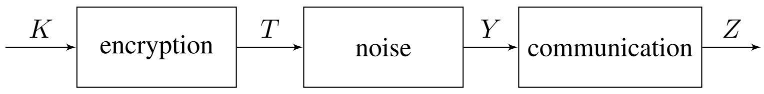


Figure 2.5: The functionality of the sensor.

conditioned on the secret. Whenever the observations of the adversary are based only on the timing of the system then the information channel is called a timing channel.

EXAMPLE 2.9 *The timing channel* $\mathsf{TC} : K \times T \mapsto [0, 1]$ *of the program* $\mathsf{P} : K \mapsto T$ *from the Example 2.4 is*

$$\mathsf{TC}(\mathtt{k}, \mathtt{t}) = \begin{cases} 1 & \textit{if}\, \mathtt{t} = \mathsf{P}(\mathtt{k}) \\ 0 & \textit{otherwise} \end{cases}$$

Since the program P *is deterministic, then for a key* $\mathtt{k} \in K$*, the probability of a timing observation* $\mathtt{t} \in T$ *conditioned on* $\mathtt{k}$ *is 1, if and only if,* $\mathtt{t} = \mathsf{P}(\mathtt{k})$*, otherwise it is 0.*

The *leakage* of the system is then defined as the difference between the initial uncertainty and the remaining uncertainty of the adversary i.e

Leakage = Initial Uncertainty - Remaining Uncertainty

The literature offers a large body of information entropy measures [CT06, Smi09, Sha01, Rn61]. We briefly discuss three of them, *Shannon-entropy*, *Min-entropy*, and *g-entropy*.

Shannon-entropy is a foundational concept of information theory, introduced by the American mathematician Shannon [Sha01]. It is a measure that explains the average uncertainty of the adversary about the secret variable of the system. In particular, it represents the optimal number of bits needed on average to describe the secret. Similarly, the *conditional Shannon-entropy* is used to describe the remaining average uncertainty of the secret when making some observations of the system's state. Denning's book [Den82] gives the first attempt to use Shannon's measures of entropy to quantify leakage in programs written in an imperative language, while later work of Clark et al. [CHM05] provided a static analysis that could compute *Shannon-leakage* in programs. Although Shannon-entropy was one of the first measures used in order to quantify leakage in programs, Smith noticed that it is not an appropriate security measure when an adversary has a high probability of guessing the secret with one try, and consequently, he suggested *Min-entropy* [Smi09].

Min-entropy is a special case of Rényi's Entropy [Rn61]. It expresses precisely the secret's *vulnerability* to being guessed correctly after one try, while the *conditional min-entropy* gives the *expected* min-entropy of the secret after observing some output [Smi09]. This entropy measure has been in particular interesting for computing leakage in information channels of deterministic programs. For example, the information leaked from a deterministic program $\mathsf{P} : I \mapsto O$ that maps secrets inputs from I to public outputs in O is equal to $\log_2 |O|$, whenever the secret is uniformly distributed [Smi09]. Therefore any over-approximation of the set of outputs O can be used directly to over-approximate the leakage of the program.

EXAMPLE 2.10 *For the program* $\mathsf{P} : K \mapsto T$ *from the Example 2.4, we have that*

$$|T| = |\{1025 + \mathsf{Ham}(\mathtt{k}) \mid \mathtt{k} \textit{ is a 1024-bit key}\}| = 1025$$

and, if we assume a uniform distribution on the set of the keys K*, then the min-leakage is* $\log_2 |T| \approx 10$ *bits.*

Later on, Alvin et al. [ACPS12] showed that min-entropy is not enough to capture the adversary's threat in all applications. For example, an adversary may benefit if he can guess some property of the secret and not the entire secret. For this reason, Alvin et al. [ACPS12] introduced g-entropy.

g-entropy [ACPS12] is a generalization of min-entropy. The core mechanism of this entropy measure is the use of *gain functions* that describe the benefit the adversary gets after guessing a specific part of the secret. Similarly to the previous entropy measures, *conditional g-entropy* is the *expected benefit* an adversary gains after making his observation. Although, g-entropy can be seen as a rich framework for expressing different attack scenarios, it hasn't received a lot of attention in practice yet.

Quantitative information flow has found many applications in measuring information leakage conveyed by timing channels [DFK$^+$13, KB07, BK15, MKP$^+$18], and in evaluating the effectiveness of countermeasures against timing channels [CRS83, KD09, ZAM11].

In Chapter 6, we investigate the effectiveness of a widely deployed countermeasure against timing channels, that is the countermeasure of reducing the accuracy of the system's clocks provided to the adversary. In particular, we use min-entropy to measure the information leakage of timing channels in real-time systems with reduced accuracy clocks.

EXAMPLE 2.11 *As an example, consider a scenario of a distributed system that consists of a sensor and a controller. In particular, the sensor continually computes some data and communicates it to the controller. For ensuring data integrity, the sensor* *always encrypts (signs) the data with his RSA private key. The RSA encryption is implemented using the modular exponentiation algorithm which is given in Example 2.3. To decrease the correlation between the encryption time and the secret bits of the key, the sensor adds noise to the encryption time by delaying for some additional period after each encryption, and then it communicates the data to the controller. Finally, on the side of the controller, we assume an adversary who runs malicious code and measures the time needed for the sensor to send its data, trying to infer bits of the sensor's private key. The measurements of the adversary are affected from the countermeasure of reducing clock resolution.*

What we are interested here is to measure how much information about the secret key $k \in K$ *is leaked, when the adversary observes the time* $z \in Z$*. This functionality of the sensor is given in Figure 2.5.*

CHAPTER 3

A Type System for Non-Interference

In this chapter, we take a language-based approach to enforce a non-interference style security property for timed automata. In particular, we adapt the guarded command language of Dijkstra [Dij75] to more closely correspond to the primitives of the timed automata formalism – resulting in the *timed command* language – and we show how to obtain timed automata from programs in timed commands. We use mandatory access control policies (MAC) to specify which components of the timed automaton are secret, and which are public. We then, develop a type system for enforcing non-interference on programs in timed commands. In particular, the type system generates a set of constraints, which over-approximate the possible flows between the components of the automaton. We prove the soundness of the type system, that is that, type-checked programs imply non-interference. Our approach is illustrated In Figure 3.1.

Chapter Organisation In Section 3.1, we give the model and the semantics of timed automata. Next, in Section 3.2, we define our non-interference condition for timed automata. In Section 3.3, we present the timed command language and we show how a timed command results in a timed automaton. In Section 3.4, we give our type system, and in Section 3.5 our soundness result. We finish with related work and our conclusions in Section 3.6, and Section 3.7 respectively.

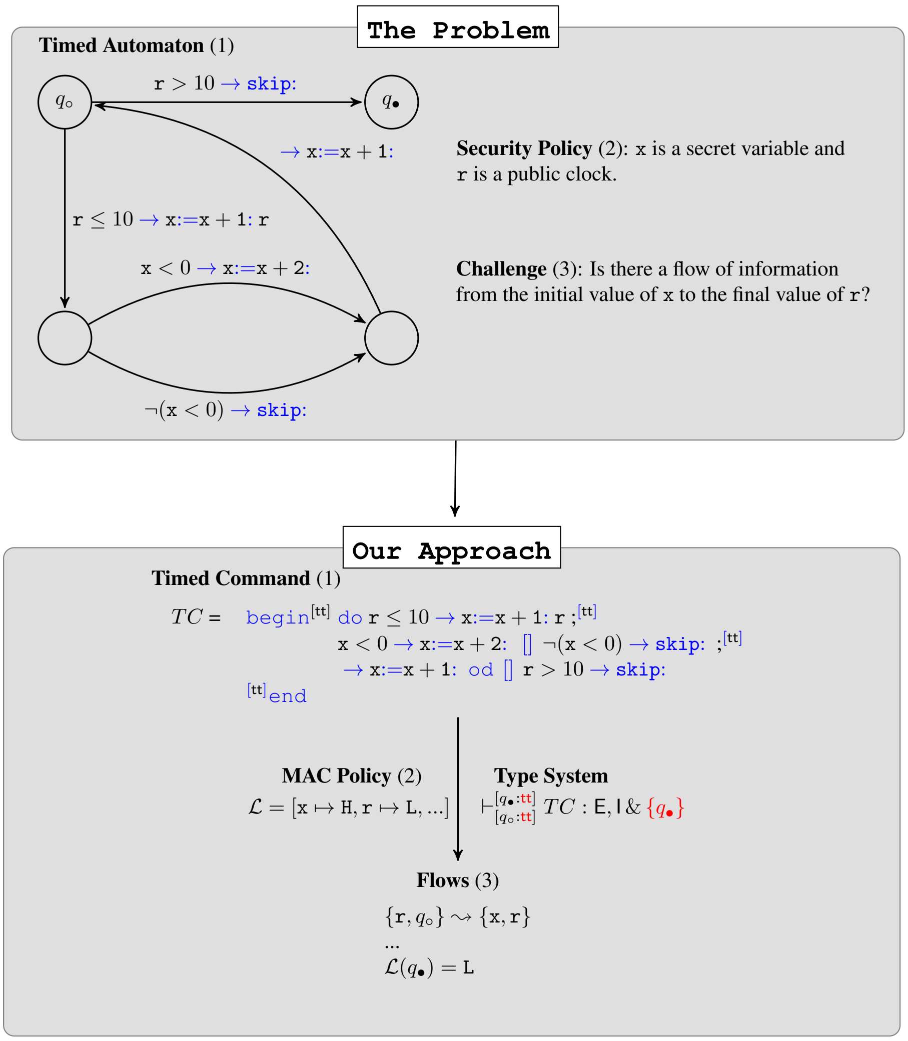

The Problem
Timed Automaton (1)
$q_\circ$
$q_\bullet$
$\mathtt{r} > 10 \rightarrow \mathtt{skip}:$
$\rightarrow \mathtt{x}{:=}\mathtt{x}+1:$
$\mathtt{r} \leq 10 \rightarrow \mathtt{x}{:=}\mathtt{x}+1: \mathtt{r}$
$\mathtt{x} < 0 \rightarrow \mathtt{x}{:=}\mathtt{x}+2:$
$\neg(\mathtt{x} < 0) \rightarrow \mathtt{skip}:$
Security Policy (2): x is a secret variable and r is a public clock.
Challenge (3): Is there a flow of information from the initial value of x to the final value of r?
Our Approach
Timed Command (1)
$TC = \mathtt{begin}^{[\mathsf{tt}]}\ \mathtt{do}\ \mathtt{r} \leq 10 \rightarrow \mathtt{x}{:=}\mathtt{x}+1: \mathtt{r}\ ;^{[\mathsf{tt}]}$
$\mathtt{x} < 0 \rightarrow \mathtt{x}{:=}\mathtt{x}+2:\ [\!]\ \neg(\mathtt{x} < 0) \rightarrow \mathtt{skip}:\ ;^{[\mathsf{tt}]}$
$\rightarrow \mathtt{x}{:=}\mathtt{x}+1:\ \mathtt{od}\ [\!]\ \mathtt{r} > 10 \rightarrow \mathtt{skip}:$
$^{[\mathsf{tt}]}\mathtt{end}$
MAC Policy (2)
$\mathcal{L} = [\mathtt{x} \mapsto \mathtt{H}, \mathtt{r} \mapsto \mathtt{L}, \ldots]$
Type System
$\vdash^{[q_\bullet:\mathsf{tt}]}_{[q_\circ:\mathsf{tt}]} TC : \mathsf{E}, \mathsf{I}\ \&\ \{q_\bullet\}$
Flows (3)
$\{\mathtt{r}, q_\circ\} \rightsquigarrow \{\mathtt{x}, \mathtt{r}\}$
...
$\mathcal{L}(q_\bullet) = \mathtt{L}$

Figure 3.1: The idea of our development in Chapter 3.

3.1 Timed Automata

In this section, we give the model and semantics of timed automata. We describe time with the set of non-negative real numbers $\mathbb{R}_{\geq 0}$. Timed automata [AILS07, AD94] are finite automata extended with real-valued variables called *dense clocks* or simply *clocks*, that are used to record the elapse of time, and integer data variables, which we simply call variables.

Dense clocks are being increased at the same rate, have infinite precision, and can reset. The transitions of the automaton are guarded with constraints over dense clocks and/or variables, restricting in that way the possible timing behaviour of the automaton.

Formally now, let **Clocks** be a finite set of *dense clocks* taking values from $\mathbb{R}_{\geq 0}$, and **Var** a finite set of *variables* taking values from $\mathbb{Z}$. A *timed automaton* [AD94, AILS07] $\mathsf{TA} = (\mathsf{Q}, \mathsf{E}, \mathsf{I}, q_\circ, q_\bullet)$ consists of a set of nodes Q, a set of annotated edges E, and a labelling function I on nodes. The node $q_\circ \in \mathsf{Q}$ will be the initial node and the node $q_\bullet$ is a final node. The mapping I maps each node in Q to a guard (to be introduced below) that will be imposed as an invariant at the node. Finally, we sometimes write (E, I) for TA.

The edges are annotated with actions and take the form $(q_s, g \rightarrow \boldsymbol{x} := \boldsymbol{a} \colon \boldsymbol{r}, q_t)$, where $q_s \in \mathsf{Q}$ is the source node, $q_t \in \mathsf{Q}$ is the target node, and $\boldsymbol{x}$, $\boldsymbol{a}$ and $\boldsymbol{r}$ are finite sequences of variables, arithmetic expressions, and clocks respectively. The action $g \rightarrow \boldsymbol{x} := \boldsymbol{a} \colon \boldsymbol{r}$ consists of a guard g that has to be satisfied in order for the multiple assignments $\boldsymbol{x} := \boldsymbol{a}$ to be performed and the clock variables $\boldsymbol{r}$ to reset. We shall assume that the sequences $\boldsymbol{x}$ and $\boldsymbol{a}$ of program variables and expressions, respectively, have the same length and that $\boldsymbol{x}$ does not contain any repetitions. To cater for special cases we shall allow to write `skip` for the assignments of $g \rightarrow \boldsymbol{x} := \boldsymbol{a} \colon \boldsymbol{r}$ when $\boldsymbol{x}$ (and hence $\boldsymbol{a}$) is empty; also we shall omit the guard g when it equals $\mathtt{tt}$, and omit the clock resets when $\boldsymbol{r}$ is empty.

The arithmetic expressions a, the boolean expressions b, and guards g are defined as follows:

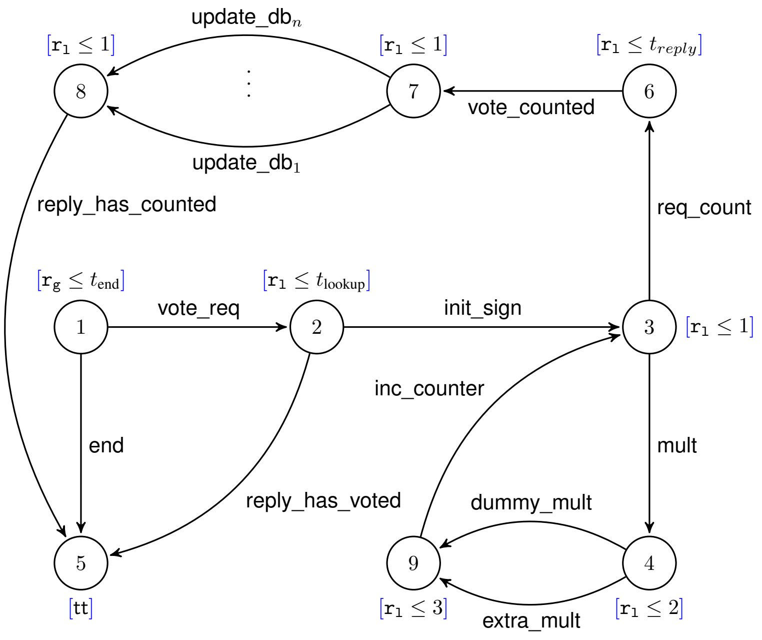
$$\begin{array}{lcl} a & ::= & a_1 \, \mathsf{op_a} \, a_2 \mid x \mid n \\ b & ::= & \mathsf{tt} \mid \mathsf{ff} \mid a_1 \, \mathsf{op_r} \, a_2 \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \mid \neg b \\ g & ::= & b \mid r \, \mathsf{op_r} \, n \mid g_1 \wedge g_2 \end{array}$$

where n is an integer, x is a data variable, r is a clock, $\mathsf{op_a}$ is a finite set of total arithmetic operators (as usual), and we also have the relational operators $\mathsf{op_r} \in \{<, \leq, =, \geq, >\}$.

EXAMPLE 3.1 *We are interested in modelling the behaviour of the signing authority* SA *of the voting system given in Example 2.6 for one session initiated from some voter. Its timed automaton is given in Figure 3.2.*

Clocks *It uses a global clock* $\mathtt{r_g}$ *and a local clock* $\mathtt{r_1}$ *to measure the overall execution time and to control its local transitions respectively. The local transitions consist of assignments to variables, and we assume that they take 1 time unit each to be performed.*

Channels SA *is using channels to receive and communicate data. We model those channels using data variables. In particular, we have the variables* $\mathtt{in_1}$, $\mathtt{in_2}$ *which*



Casting Phase

vote_req	$\mathtt{r}_1 \geq 2 \rightarrow (\mathtt{id}, \mathtt{v}) := (\mathtt{in}_1, \mathtt{in}_2) \colon \mathtt{r}_1$
init_sign	$\mathtt{r}_1 \geq 3 \wedge \bigvee_{j=1}^{n} (\mathtt{id} = j \wedge \mathtt{d}_j = 0) \rightarrow (\mathtt{i}, \mathtt{s}, \mathtt{y}) := (1, 1, 1) \colon \mathtt{r}_1$
reply_has_voted	$\mathtt{r}_1 \geq 1 \wedge \neg \bigvee_{j=1}^{n} (\mathtt{id} = j \wedge \mathtt{d}_j = 0) \rightarrow \mathtt{out}_2 := 0 \colon \mathtt{r}_1$

Signing Phase

mult	$\mathtt{r}_1 \geq 1 \wedge \mathtt{i} < 1025 \rightarrow \mathtt{s} := \mathtt{s} \cdot \mathtt{s} \colon$
extra_mult	$\mathtt{r}_1 \geq 2 \wedge \bigvee_{j=1}^{1024} (\mathtt{i} = j \wedge \mathtt{k}_j = 1) \rightarrow \mathtt{s} := \mathtt{s} \cdot \mathtt{v} \colon$
dummy_mult	$\mathtt{r}_1 \geq 2 \wedge \neg \bigvee_{j=1}^{1024} (\mathtt{i} = j \wedge \mathtt{k}_j = 1) \rightarrow \mathtt{y} := \mathtt{y} \cdot \mathtt{v} \colon$
inc_counter	$\mathtt{r}_1 \geq 3 \rightarrow \mathtt{i} := \mathtt{i} + 1 \colon \mathtt{r}_1$

Counting Phase

req_count	$\mathtt{r}_1 \geq 1 \wedge \neg(\mathtt{i} < 1025) \rightarrow \mathtt{out}_1 := \mathtt{s} \colon \mathtt{r}_1$
vote_counted	$\mathtt{r}_1 \geq 1 \rightarrow \mathtt{x} := \mathtt{in}_3 \colon \mathtt{r}_1$
update_vote_db$_j$	$\mathtt{r}_1 \geq 1 \wedge \mathtt{id} = j \rightarrow \mathtt{d}_j := 1 \colon \mathtt{r}_1$
reply_has_counted	$\mathtt{r}_1 \geq 1 \rightarrow \mathtt{out}_2 := 1 \colon \mathtt{r}_1$

Ending Phase

end	$\mathtt{r}_\mathtt{g} \geq t_{\text{end}} \rightarrow \colon \mathtt{r}_1$

Figure 3.2: The timed automaton of the signing authority SA, and the abbreviations of its actions grouped up based on the different phases (casting, signing, counting, ending) of the voting system.

store the unique identity of a user, and its vote, whenever the authenticator A *writes to them. The channel* $\mathtt{out}_1$ *is used to send the signed vote from* SA *to the counting mechanism* C*, while the channel* $\mathtt{in}_3$ *stores the reply from* C *when the vote of the voter has been counted. Finally, the channel* $\mathtt{out}_2$ *is used to send the messages of the signing* *authority* SA *to the voter.*

Variables *The variables* `id` *and* `v` *are used to store the identity and the vote of a voter, received from the channels* $\mathtt{in}_1$ *and* $\mathtt{in}_2$ *respectively. In particular, if the voter* V_j *wants to vote then the authenticator will send the identity* j *through the channel* $\mathtt{in}_1$*. The variables* $\mathtt{d}_1,\ldots,\mathtt{d}_\mathtt{n}$ *are used to record if a voter has not voted yet by holding the value 0, or any other value otherwise. For the RSA signature the automaton is using an implementation of the algorithm given in Example 2.3, introducing a dummy variable* `y` *for eliminating the timing channel created due to the extra multiplication. The bits of the 1024-bit RSA key are represented by the variables* $\mathtt{k}_1,\ldots,\mathtt{k}_{1024}$*, the signature of the vote* `v` *is stored in the variable* `s`*, and* `i` *is the index variable used for the loop iteration. Finally, the variable* `x` *is used to store the reply from the counting mechanism* C*.*

Transitions *The signing authority* SA *starts at the initial location 1, and waits for votes from the authenticator* A *until the end of the voting period denoted by the invariant* $\mathtt{r}_\mathtt{g} \leq t_{end}$ *(*t_{end} *is a constant greater than 2). If the value of* $\mathtt{r}_\mathtt{g}$ *becomes equal to* t_{end} *the automaton terminates by moving to its final location 5 (the ending phase). The edge from 1 to 2 is taken whenever a vote request occurs (the casting phase begins).*

Next, at location 2, the authority checks if the user has already voted (this could take up to $t_{lookup} \geq 3$*). If the check fails, it replies to the voter with the constant 0 using the channel* $\mathtt{out}_2$*. Otherwise, it moves to location 3 by performing an initialization of the variables needed for the signature (the signing phase begins).*

The loop transition starting at 3, moving to 4, 9 and back to 3 is the loop of the RSA encryption algorithm. Once it has been completed the automaton moves to the location 6 by sending the signature `s` *to* C *using the channel* $\mathtt{out}_1$ *(the counting phase begins). Next, it waits (up to* $t_{reply} > 1$*) for a reply from* C*, and once it receives it, it reads it from channel* $\mathtt{in}_3$*, stores the reply in variable* `x`*, and moves to location 7. Finally, it updates the variable* $\mathtt{d}_j$ *to 1 (if the voter has* $\mathtt{id}{=}j$*) since now the voter has voted, and notifies him by sending the constant 1 through the channel* $\mathtt{out}_2$*.*

Finally, the $\cdot$ *operator is the modulo* n *multiplication.*

3.1.1 Timed Automata Semantics

To specify the semantics of timed automata let σ be a state mapping variables to their integer values, and let δ be a clock assignment mapping clocks to non-negative reals. We then have total semantic functions $[\![\cdot]\!]$ for evaluating the arithmetic expressions, boolean tests, and guards; the values of the arithmetic expressions and boolean expressions only depend on the states, whereas that of guards also depend on the clock assignments.

The semantics of timed automata is given by a transition system whose configurations have the form $\langle q, \sigma, \delta\rangle \in$ **Config**, where $[\![\mathsf{I}(q)]\!](\sigma,\delta)$ is true, and the transitions are described by an initial delay (possibly none) that increases the values of all the clocks followed by an action. Therefore, whenever $(q_s, g \rightarrow \boldsymbol{x} := \boldsymbol{a}\colon \boldsymbol{r}, q_t)$ is in E we have the rule:

$$\langle q_s, \sigma, \delta\rangle \xrightarrow{t} \langle q_t, \sigma', \delta'\rangle \begin{cases} t \geq 0 \\ [\![\mathsf{I}(q_s)]\!](\sigma, \delta + t) = \mathsf{tt}, \\ [\![g]\!](\sigma, \delta + t) = \mathsf{tt}, \\ \sigma' = \sigma[\boldsymbol{x} \mapsto [\![\boldsymbol{a}]\!]\sigma], \delta' = (\delta + t)[\boldsymbol{r} \mapsto \mathbf{0}], \\ [\![\mathsf{I}(q_t)]\!](\sigma', \delta') = \mathsf{tt} \end{cases}$$

where t corresponds to the initial delay. The rule ensures that after the initial delay the invariant and the guard are satisfied in the starting configuration, and updates the mappings σ and δ. Here $\delta + t$ abbreviates $\lambda r.\, \delta(r) + t$. Finally, it ensures that the invariant is satisfied in the resulting configuration. Initial configurations are the ones of the node $q_\circ$.

We define a *trace* from $\langle q_s, \sigma, \delta\rangle$ to q_t in a timed automaton TA to have one of three forms. It may be a finite "successful" sequence

$$\begin{array}{ll} \langle q_s, \sigma, \delta\rangle = \langle q'_0, \sigma'_0, \delta'_0\rangle \xrightarrow{t_1} \cdots \xrightarrow{t_n} \langle q'_n, \sigma'_n, \delta'_n\rangle & \qquad (n > 0) \\ \text{such that } \{n\} = \{i \mid q'_i = q_t \ \wedge \ 0 < i \leq n\}. & \end{array}$$

in which case at least one step is performed. It may be a finite "unsuccessful" sequence

$$\begin{array}{ll} \langle q_s, \sigma, \delta\rangle = \langle q'_0, \sigma'_0, \delta'_0\rangle \xrightarrow{t_1} \cdots \xrightarrow{t_n} \langle q'_n, \sigma'_n, \delta'_n\rangle & \qquad (n \geq 0) \\ \text{such that } \langle q'_n, \sigma'_n, \delta'_n\rangle \text{ is stuck and } q_t \notin \{q'_1, \cdots, q'_n\} & \end{array}$$

where $\langle q'_n, \sigma'_n, \delta'_n\rangle$ is stuck when there is no transition starting from $\langle q'_n, \sigma'_n, \delta'_n\rangle$. Finally, it may be an infinite "unsuccessful" sequence

$$\begin{array}{l} \langle q_s, \sigma, \delta\rangle = \langle q'_0, \sigma'_0, \delta'_0\rangle \xrightarrow{t_1} \cdots \xrightarrow{t_n} \langle q'_n, \sigma'_n, \delta'_n\rangle \xrightarrow{t_{n+1}} \cdots \\ \text{such that } q_t \notin \{q'_1, \cdots, q'_n, \cdots\}. \end{array}$$

Finally, for a configuration $\langle q_s, \sigma, \delta\rangle$ and the node q_t we define the *trace behaviour* $\mathsf{Final}[\![\mathsf{TA} : q_s \mapsto q_t]\!](\sigma, \delta)$ as

$$\begin{array}{l} \mathsf{Final}[\![\mathsf{TA} : q_s \mapsto q_t]\!](\sigma, \delta) = \\ \qquad \{(\sigma', \delta') \mid \text{a successful trace from } \langle q_s, \sigma, \delta\rangle \text{ in } \mathsf{TA} \text{ ends in } \langle q_t, \sigma', \delta'\rangle\} \\ \qquad \cup \{\bot \mid \text{there is an unsuccessful trace from } \langle q_s, \sigma, \delta\rangle \text{ in } \mathsf{TA} \text{ to } q_t\ \} \end{array}$$

In particular, it is the set that contains all the *final pairs of states* of successful traces that end at q_t, or the $\bot$ element in case of the existence of an unsuccessful trace.

3.2 Non-Interference in Timed Automata

In this section, we give our non-interference semantic notion for security in timed automata. In particular, we assume a victim which is modelled as a timed automaton, operating under some secret input information. For the adversary we assume that it knows (1) the timed automaton (i.e the system of the victim), (2) some public input information, and (3) it observes some public output information at the final node of the automaton. The goal of the adversary is to deduce information about the initial secret information. Security is then defined by a non-interference condition between the initial secret and the final public information i.e that there is no flow of information from the initial secret to the final public information.

EXAMPLE 3.2 *Returning to the voting system of Example 3.1 we shall assume that the variables* $\mathtt{k}_1, ..., \mathtt{k}_{1024}$ *that store the bits of the secret key, the variables* `s` *and* `y` *(i.e the dummy variable) which store the signature of the vote, and the channel variable* $\mathtt{out}_1$ *which communicates the signed vote to the counting authority are secret, whereas the rest of the variables and clocks are public.*

To this end, we envisage that there is a security lattice expressing the permissible flows [DD77]. Formally, this is a complete lattice and the permitted flows go in the direction of the partial order. In our development it will contain just two elements, `L` (for *low* or *public*) and `H` (for *high* or *secret*), and we set $\mathtt{L} \sqsubseteq \mathtt{H}$ so that only the flow from `H` to `L` is disallowed.

A *security policy* is then expressed by a mapping $\mathcal{L}$ that assigns an element of the security lattice to each program variable, clock, and node. An entity is called *high* or *secret*, if it is mapped to `H` by $\mathcal{L}$, and it is said to be *low* or *public* if it is mapped to `L` by $\mathcal{L}$.

EXAMPLE 3.3 *Returning to the voting system of Examples 3.1 and 3.2 we shall let the security policy* $\mathcal{L}$ *map the variables* $\mathtt{k}_1, ..., \mathtt{k}_{1024}$, `s`, `y`, *and* $\mathtt{out}_1$ *to the high security level (*`H`*), whereas the remaining variables, and all the clocks and nodes to the low security level (*`L`*).*

To express *adherence to the security policy* we use the binary operation $\rightsquigarrow$ defined on sets χ and χ' (of variables, clocks and nodes):

$$\chi \rightsquigarrow \chi' \Leftrightarrow \forall u \in \chi : \forall u' \in \chi' : \mathcal{L}(u) \sqsubseteq \mathcal{L}(u')$$

This expresses that all the entities of χ may flow into those of χ'; note that if one of the entities of χ has a high security level then it must be the case that all the entities of χ' have high security level. Note also that $\rightsquigarrow$ is transitive but not reflexive.

In our development, a timed automaton gives rise to constraints of the form $\{y\} \rightsquigarrow \{x\}$ whenever the value of y may somehow influence (or flow into) that of x. Those flows are either *explicit* or *implicit* flows (i.e from covert channels, see Section 2.2).

As an example of an *explicit* flow in a timed automaton consider a simple assignment of the form $x{:=}y + z$. This gives rise to a condition $\mathsf{fv}(y + z) \rightsquigarrow \{x\}$ so as to to indicate that the *explicit* flow from the variables y, z to the variable x must adhere to the security policy. In particular, if $\{y, z\}$ contains a variable with high security level then x must also have high security level.

For an example of an *implicit* flow in a timed automaton consider a conditional assignment $g \rightarrow x{:=}0$ where x is assigned the constant value 0 in case g evaluates to true. This gives rise to a condition $\mathsf{fv}(g) \rightsquigarrow \{x\}$ so as to to indicate that the *implicit* flow from the variables of g to the variable x must adhere to the security policy. In particular, if g contains a variable with high security level then x also must have high security level.

Before we introduce our notion of non-interference, we define a relation on pairs of variables and clock states, which expresses that an adversary is able to distinguish two pairs, whenever they differ on variables or clocks with low security level. Formally, we write $(\sigma, \delta) \equiv (\sigma', \delta')$ to indicate that the two pairs are equal on low variables and low clocks:

$$(\sigma, \delta) \equiv (\sigma', \delta') \quad \text{iff} \quad \begin{array}{l} \forall x : \mathcal{L}(x) = \mathsf{L} \Rightarrow \sigma(x) = \sigma'(x) \wedge \\ \forall r : \mathcal{L}(r) = \mathsf{L} \Rightarrow \delta(r) = \delta'(r) \end{array}$$

To cater for the $\perp$ behaviour produced by the trace behaviour we shall allow to write $\perp \equiv \perp$ and take it for granted that $\perp \not\equiv (\sigma, \delta)$ and $(\sigma, \delta) \not\equiv \perp$. It is immediate that this definition of $\equiv$ gives rise to an equivalence relation.

We next lift the operation $\equiv$ to work on sets:

$$H \equiv H' \quad \text{iff} \quad \begin{array}{l} \forall \eta \in H : \exists \eta' \in H' : \eta \equiv \eta' \wedge \\ \forall \eta' \in H' : \exists \eta \in H : \eta \equiv \eta' \end{array}$$

Here η ranges over pairs (σ, δ) as well as $\perp$, and it is immediate that this definition of $\equiv$ gives rise to an equivalence relation.

We can now express our non-interference condition for timed automata.

Definition 3.1 (**Non-Interference**) For a timed automaton $\mathsf{TA} = (\mathsf{Q}, \mathsf{E}, \mathsf{I}, q_\circ, q_\bullet)$, we will say that TA satisfies non-interference with respect to the security policy $\mathcal{L}$ whenever:

$$\begin{array}{l} (\sigma, \delta) \equiv (\sigma', \delta') \;\wedge\; [\![\mathsf{I}(q_\circ)]\!](\sigma, \delta) \;\wedge\; [\![\mathsf{I}(q_\circ)]\!](\sigma', \delta') \\ \Downarrow \\ \mathsf{Final}[\![(\mathsf{E}, \mathsf{I}) : q_\circ \mapsto q_\bullet]\!](\sigma, \delta) \equiv \mathsf{Final}[\![(\mathsf{E}, \mathsf{I}) : q_\circ \mapsto q_\bullet]\!](\sigma', \delta') \end{array}$$

This condition caters for a passive adversary that observes the public part of final configurations, and tries to deduce secret information of the initial configurations. In particular, it says that if we consider two initial configurations that only differ on high variables and clocks then the final configurations are also only allowed to differ on high variables and clocks. Otherwise an adversary observing the final configurations could infer information about the initial secret variables or clocks. In other words, there is no information flow from the initial values of high variables and clocks to the final values of low variables and clocks.

The fact that the trace behaviour produces a set of configurations means that we take due care of non-determinism, and the fact that the trace behaviour may contain $\perp$ means that we take due care of non-termination (because of looping or because of getting stuck). In addition to that, because we allow the adversary to observe the final values of low clocks, it also means that we take care of potential timing channels. Finally, our semantic condition is more involved than in classical papers like [VSI96] which consider only deterministic programs, due to the highly non-deterministic nature of timed automata (i.e our notion also caters for covert channels as the one given in Example 2.2).

3.3 Timed Commands

The semantic condition for non-interference is undecidable in general [Cas09]. To obtain a sound and decidable enforcement mechanism, the traditional approach is to develop a type system for a suitable programming language or process calculus.

To this end we introduce the language TC of *timed commands*. It is strongly motivated by Dijkstra's language of guarded commands [Dij75] but is designed so that it combines guards and assignments in the manner of timed automata. The syntax is given by:

$$\begin{array}{rcl} TC & ::= & \texttt{begin}^{[g_\circ]}\, C\ ^{[g_\bullet]}\texttt{end} \\ C & ::= & g \rightarrow \boldsymbol{x} := \boldsymbol{a} \colon \boldsymbol{r} \ \mid\ C_1;^{[g]}C_2 \ \mid\ \texttt{do}\, T_1 \,[]\, \cdots \,[]\, T_n\ \texttt{od}\, []\, T_{n+1} \,[]\, \cdots \,[]\, T_m \\ T & ::= & g \rightarrow \boldsymbol{x} := \boldsymbol{a} \colon \boldsymbol{r} \ \mid\ T;^{[g]}C \end{array}$$

A timed command TC specifies a guard condition $g_\circ$ that must hold initially and a condition $g_\bullet$ that must hold if the command terminates. The command C itself can have one of three forms. One possibility is that it is an action of the form $g \rightarrow \boldsymbol{x} := \boldsymbol{a} \colon \boldsymbol{r}$. Another possibility is that it is a sequence of commands and then the condition g must be satisfied when moving from the first command to the second. The third possibility is that it is a looping construct with a number of branches $T_1, \cdots, T_n$ that will loop and a number of branches $T_{n+1}, \cdots, T_m$ that will terminate the looping behaviour. In case

$$\vdash_{q_s}^{q_t} g \to \boldsymbol{x} := \boldsymbol{a} \colon \boldsymbol{r} : \{(q_s, g \to \boldsymbol{x} := \boldsymbol{a} \colon \boldsymbol{r}, q_t)\}, [\,]$$

$$\frac{\vdash_{q_s}^{q} C_1 : \mathsf{E}_1, \mathsf{I}_1 \qquad \vdash_{q}^{q_t} C_2 : \mathsf{E}_2, \mathsf{I}_2}{\vdash_{q_s}^{q_t} C_1;^{[g]} C_2 : \mathsf{E}_1 \cup \mathsf{E}_2, \mathsf{I}_1 \cup \mathsf{I}_2 \cup [q \mapsto g]} \qquad \text{where } q \text{ is fresh}$$

$$\frac{\bigwedge_{i=1}^{n} \vdash_{q_s}^{q_s} T_i : \mathsf{E}_i, \mathsf{I}_i \qquad \bigwedge_{i=n+1}^{m} \vdash_{q_s}^{q_t} T_i : \mathsf{E}_i, \mathsf{I}_i}{\vdash_{q_s}^{q_t} \texttt{do}\, T_1 \,[]\, \cdots \,[]\, T_n \,\texttt{od}\, []\, T_{n+1} \,[]\, \cdots \,[]\, T_m : \bigcup_i \mathsf{E}_i, \bigcup_i \mathsf{I}_i}$$

$$\frac{\vdash_{q_\circ}^{q_\bullet} C : \mathsf{E}, \mathsf{I}}{\vdash \texttt{begin}^{[g_\circ]}\, C\, ^{[g_\bullet]}\texttt{end} : \mathsf{E}, \mathsf{I}', q_\circ, q_\bullet} \qquad \text{where } \begin{cases} \mathsf{I}' = \mathsf{I}[q_\circ \mapsto g_\circ; q_\bullet \mapsto g_\bullet] \\ q_\circ, q_\bullet \text{ are fresh} \end{cases}$$

Table 3.1: From Timed Commands to Timed Automata.

$n = 0$ and $m > 1$ we dispense with the `do od`. Here T is a special form of command that starts with an action and potentially is followed by a number of commands. Guards and expressions are defined as in Section 3.1.

EXAMPLE 3.4 *The automaton of the signing authority* SA *of Example 3.1 is given by the following timed command*

$$\texttt{begin}^{[\mathtt{r_g} \le t_{end}]}\ T_{casting}\ []\ T_{ending}\ ^{[\mathtt{tt}]}\ \texttt{end}$$

where the command T_{ending}=end *describes the ending phase of the voting system, the command*

$$T_{casting} = \ \text{vote\_req};^{[\mathtt{r_1} \le t_{lookup}]} \\ (\ \text{init\_sign};^{[\mathtt{r_1} \le 1]}\ \texttt{do} T_{signing}\ \texttt{od}\ []\ T_{counting})\ []\ (\ \text{reply\_has\_voted})$$

describes the casting phase, while

$$T_{signing} = \ \text{mult};^{[\mathtt{r_1} \le 2]}\ (\ \text{extra\_mult}\ []\ \text{dummy\_mult}\ );^{[\mathtt{r_1} \le 3]}\ \text{inc\_counter}$$

corresponds to the signing phase, and finally,

$$T_{counting} = \ \text{req\_count};^{[\mathtt{r_1} \le t_{reply}]}\ \text{vote\_counted};^{[\mathtt{r_1} \le 1]} \\ (\ \text{update\_db}_1\ []\ \ldots\ []\ \text{update\_db}_n);^{[\mathtt{r_1} \le 1]}\ \text{reply\_has\_counted}$$

describes the counting phase.

Transformational Semantics We shall define the semantics of a timed command by mapping it into a timed automaton. Consider $\texttt{begin}^{[g_\circ]}\ C\ ^{[g_\bullet]}\texttt{end}$ and let $q_\circ$ and $q_\bullet$ be two distinct nodes; they will be the initial and final node of the resulting timed automaton and we shall ensure that $\mathsf{I}(q_\circ) = g_\circ$ and $\mathsf{I}(q_\bullet) = g_\bullet$. Additional nodes will be created during the construction using a judgment of the form:

$$\vdash_{q_s}^{q_t} C : \mathsf{E}, \mathsf{I}$$

Here C is a timed command, q_s and q_t are nodes, E is a set of edges, and the judgment will introduce additional nodes whose invariants are given by the labelling function I. This defines a timed automaton with initial node q_s, final node q_t, edges E, and labelling function I.

The judgment is specified by the axioms and rules of Table 3.1. In the axiom we simply create the edge $(q_s, g \rightarrow \boldsymbol{x} := \boldsymbol{a} \colon \boldsymbol{r}, q_t)$ starting in q_s and ending in q_t and indicating the action to be performed; the resulting labelling function is empty as no new nodes are created in the construct.

In the first rule we create a *fresh* node q to be used to glue the timed automata for C_1 and C_2 together; the node q has the invariant g and is used as target node for C_1 as well as source node for C_2. The resulting set of edges is the union of the two sets; the two branches will create disjoint sets of nodes so the two mappings I_1 and I_2 will have disjoint domains and we write union for their combination.

In the rule for the looping construct we achieve the looping of the branches $T_1, \cdots, T_n$ by using q_s as source as well as target node, whereas for $T_{n+1}, \cdots, T_m$ we use q_t as target node. The overall set of edges is obtained as the union of the edges E_i and as in the previous case the domains of the mappings I_i will be disjoint so the mappings are easily combined.

Recall that T is a special form of timed command and hence timed automata can be constructed using the judgments of Table 3.1. The timed automata constructed from T always have exactly one edge leaving the initial node and do not contain any edge back to the initial node unless the initial and final nodes coincide. The timed automata constructed from C may have more than one edge leaving the initial node and may contain edges back to the initial node even when the initial and final nodes are distinct.

For the overall timed command $\texttt{begin}^{[g_\circ]}\ C\ ^{[g_\bullet]}\texttt{end}$ we can now obtain a timed automaton with initial node $q_\circ$, final node $q_\bullet$, edges E, and labelling function I' given by the last inference rule of Table 3.1.

Example 3.5 *The transformation applied to the timed command of Example 3.4 gives rise to the timed automaton of Figure 3.2.*

3.4 Type System

The information flow type system is specified using judgments of the form

$$\vdash^{[q_t : g_t]}_{[q_s : g_s]} C : \mathsf{E}, \mathsf{I} \,\&\, \chi$$

This is an extension of the judgments $\vdash^{q_t}_{q_s} C : \mathsf{E}, \mathsf{I}$ of the previous section for constructing timed automata from commands. The new judgments maintain information about the invariants g_s and g_t associated with the nodes q_s and q_t and a set χ of *latent variables and nodes* that influence the termination of the command; the influence of χ on q_t remains to be enforced. The type system is specified in Table 3.2 and explained below.

Assignment Consider the first axiom of Table 3.2. The second line of the side condition expresses all the explicit flows from components of the sequence of expressions to corresponding components of the sequence of variables. The first line of the side condition expresses that the modifications of variables and clocks as well as the termination relies on having started the action. The third line of the side condition expresses our knowledge that g_s holds and the implicit flows arising from testing the guard g in the pre-state and the condition g_t in the post-state before performing the modifications of variables and clocks. (We are using the insight from Hoare logic [Apt81] that evaluating g_t in the post-state is the same as evaluating $g_t[\boldsymbol{a}/\boldsymbol{x}][\boldsymbol{0}/\boldsymbol{r}]$ in the pre-state.) Rather than also expressing the implicit flow for termination (in the form of a side condition $\mathsf{fv}(g_s \wedge g \wedge g_t[\boldsymbol{a}/\boldsymbol{x}][\boldsymbol{0}/\boldsymbol{r}]) \rightsquigarrow \{q_t\}$) we produce the latent set of variables and nodes $\{q_s\} \cup \mathsf{fv}(g_s \wedge g \wedge g_t[\boldsymbol{a}/\boldsymbol{x}][\boldsymbol{0}/\boldsymbol{r}])$ as listed after the ampersand in the axiom. (We shall see the flexibility offered by this approach shortly.)

EXAMPLE 3.6 *Consider the command*

$$\mathsf{extra\_mult} = \mathtt{r}_1 \geq 2 \wedge \bigvee_{j=1}^{1024} (\mathtt{i} = j \wedge \mathtt{k}_j = 1) \rightarrow \mathtt{s} := \mathtt{s} \cdot \mathtt{v}:$$

appearing in the timed command $T_{signing}$ in Example 3.4, and the automaton from Figure 3.2. It is the case that $q_s = 4$, $q_t = 9$, $g_s = \mathtt{r}_1 \leq 2$, and $g_t = \mathtt{r}_1 \leq 3$. The type system imposes the following constraints on the flows:

$$\{4\} \rightsquigarrow \{9, \mathtt{s}\}, \quad \{\mathtt{s}, \mathtt{v}\} \rightsquigarrow \{\mathtt{s}\}, \quad \{\mathtt{r}_1, \mathtt{i}, \mathtt{k}_1, ..., \mathtt{k}_{1024}\} \rightsquigarrow \{\mathtt{s}\}$$

It is easy to check that they are fulfilled for the security assignment of Example 3.3. The latent set of variables is $\{4, \mathtt{r}_1, \mathtt{i}, \mathtt{k}_1, ..., \mathtt{k}_{1024}\}$.

$$\vdash^{[q_t:g_t]}_{[q_s:g_s]} g \rightarrow \boldsymbol{x} := \boldsymbol{a}\colon \boldsymbol{r} : \{(q_s, g \rightarrow \boldsymbol{x} := \boldsymbol{a}\colon \boldsymbol{r}, q_t)\}, [\,] \;\&\; \{q_s\} \cup \mathsf{fv}(g_s \wedge g \wedge g_t[\boldsymbol{a}/\boldsymbol{x}][\boldsymbol{0}/\boldsymbol{r}])$$

$$\text{if}\quad \begin{array}{l} \{q_s\} \leadsto \{q_t, \boldsymbol{x}, \boldsymbol{r}\} \\ \bigwedge_i \mathsf{fv}(a_i) \leadsto \{x_i\} \\ \mathsf{fv}(g_s \wedge g \wedge g_t[\boldsymbol{a}/\boldsymbol{x}][\boldsymbol{0}/\boldsymbol{r}]) \leadsto \{\boldsymbol{x}, \boldsymbol{r}\} \end{array}$$

$$\dfrac{\vdash^{[q:g]}_{[q_s:g_s]} C_1 : \mathsf{E}_1, \mathsf{I}_1 \;\&\; \chi_1 \qquad \vdash^{[q_t:g_t]}_{[q:g]} C_2 : \mathsf{E}_2, \mathsf{I}_2 \;\&\; \chi_2}{\vdash^{[q_t:g_t]}_{[q_s:g_s]} C_1 ;^{[g]} C_2 : \mathsf{E}_1 \cup \mathsf{E}_2, \mathsf{I}_1 \cup \mathsf{I}_2 \cup [q \mapsto g] \;\&\; \chi_2}$$

$$\text{if}\quad \begin{array}{l} q \text{ is fresh} \\ \mathsf{fv}(g) \cup \{q\} \leadsto \textbf{Clocks} \cup \{q\} \\ \chi_1 \leadsto \{q\} \end{array}$$

$$\dfrac{\bigwedge_{i=1}^{n} \vdash^{[q_s:g_s]}_{[q_s:g_s]} T_i : \mathsf{E}_i, \mathsf{I}_i \;\&\; \chi_i \qquad \bigwedge_{i=n+1}^{m} \vdash^{[q_t:g_t]}_{[q_s:g_s]} T_i : \mathsf{E}_i, \mathsf{I}_i \;\&\; \chi_i}{\vdash^{[q_t:g_t]}_{[q_s:g_s]} \texttt{do}\, T_1 \,[]\, \cdots \,[]\, T_n \,\texttt{od}\, [] \,T_{n+1} \,[]\, \cdots \,[]\, T_m : \bigcup_i \mathsf{E}_i, \bigcup_i \mathsf{I}_i \;\&\; \{q_t\}}$$

$$\text{if}\quad \begin{array}{l} \{q_s\} \leadsto \{q_t\} \\ \bigwedge_{i=1}^{n} \chi_i \leadsto \{q_s\} \\ \Phi^{T_1,\cdots,T_n}_{T_{n+1},\cdots,T_m}\left[{}^{q_t:g_t}_{q_s:g_s}\right] \Rightarrow \bigwedge_{i=n+1}^{m} \chi_i \leadsto \{q_t\} \\ \bigwedge_{i,j \mid i \neq j, \underline{\mathrm{sat}}(\mathsf{fst}^{\zeta_i}_{g_s}(T_i) \wedge \mathsf{fst}^{\zeta_j}_{g_s}(T_j))} \chi_i \leadsto \mathsf{ass}(T_j) \\ \qquad \text{where } \zeta_l \text{ is } g_s \text{ if } l \leq n \text{ and } \zeta_l \text{ is } g_t \text{ if } l > n \\ \bigwedge_{i=n+1}^{m} \left(\forall r \in \mathsf{fv}(\mathsf{fst}^{g_t}_{g_s}(T_i)) \cap \textbf{Clocks} : \mathcal{L}(r) = \mathtt{L}\right) \wedge \\ \qquad \left(\bigwedge_{j=n+1}^{m} (\overline{\mathsf{fst}^{g_t}_{g_s}(T_i)} \Leftrightarrow \overline{\mathsf{fst}^{g_t}_{g_s}(T_j)}) \vee (\forall x \in \mathsf{fv}(\mathsf{fst}^{g_t}_{g_s}(T_i)) \cap \textbf{Var} : \mathcal{L}(x) = \mathtt{L})\right) \end{array}$$

$$\dfrac{\vdash^{[q_\bullet:g_\bullet]}_{[q_\circ:g_\circ]} C : \mathsf{E}, \mathsf{I} \;\&\; \chi}{\vdash \texttt{begin}^{[g_\circ]}\, C \,^{[g_\bullet]}\texttt{end} : \mathsf{E}, \mathsf{I}', q_\circ, q_\bullet} \qquad \text{where} \left\{ \begin{array}{l} \mathsf{I}' = \mathsf{I}[q_\circ \mapsto g_\circ; q_\bullet \mapsto g_\bullet] \\ \mathsf{fv}(g_\circ) \cup \{q_\circ\} \leadsto \textbf{Clocks} \cup \{q_\circ\} \\ \mathsf{fv}(g_\bullet) \cup \{q_\bullet\} \leadsto \textbf{Clocks} \cup \{q_\bullet\} \\ \chi \leadsto \{q_\bullet\} \\ \mathcal{L}(q_\bullet) = \mathtt{L} \\ q_\circ, q_\bullet \text{ are fresh} \end{array} \right.$$

Table 3.2: Type System for Timed Commands.

Sequence The first inference rule of Table 3.2 deals with the sequential composition of two commands. The second line of the side condition expresses the explicit flow possible due to the delay at the node q separating the two commands. Recall that **Clocks** is the set of all clock variables and it is included to mimic the effect of the potential delay. The third line of the side condition takes care of imposing the latent effect of the first command on the node q following immediately after it.

EXAMPLE 3.7 *Let us consider the sequence of commands*

$$(\mathsf{extra\_mult}\ [\!]\ \mathsf{dummy\_mult});^{[\mathtt{r_1} \leq 3]}\ \mathsf{inc\_counter}$$

from Example 3.4, and the automaton from Figure 3.2. The latent set of variables from the first non-deterministic command will simply be {9}, and the two constraints will amount to $\{\mathtt{r_1}, 9\} \rightsquigarrow \{\mathtt{r_1}, \mathtt{r_g}, 9\}$, *and* $\{9\} \rightsquigarrow \{9\}$, *which are satisfied for the security assignment of Example 3.3.*

Auxiliary Operations Before approaching the last inference rule in Table 3.2 we shall introduce three auxiliary operations.

The auxiliary operation $\mathsf{ass}(C)$ over-approximates the set of variables and clocks modified by the command (ignoring any initial and final delays):

$$\begin{array}{rcl}
\mathsf{ass}(g \rightarrow \boldsymbol{x} := \boldsymbol{a} \colon \boldsymbol{r}) & = & \{\boldsymbol{x}, \boldsymbol{r}\} \\
\mathsf{ass}(C_1;^{[g]} C_2) & = & \mathsf{ass}(C_1) \cup \mathsf{ass}(C_2) \cup \mathbf{Clocks} \\
\mathsf{ass}\left(\begin{array}{c} \mathsf{do}\, T_1\ [\!]\ \cdots\ [\!]\ T_n\ \mathsf{od} \\ [\!]\ T_{n+1}\ [\!]\ \cdots\ [\!]\ T_m \end{array}\right) & = & \left\{\begin{array}{ll} \mathsf{ass}(T_1) \cup \cdots \cup \mathsf{ass}(T_m) \cup \mathbf{Clocks} & \text{if } n > 0 \\ \mathsf{ass}(T_1) \cup \cdots \cup \mathsf{ass}(T_m) & \text{if } n = 0 \end{array}\right.
\end{array}$$

where recall that **Clocks** is the set of all clocks, and it is included to mimic the effect of the potential (internal) delays of the sequence and loop command. The correctness of our $\mathsf{ass}(.)$ is given with the following fact

FACT 1 *If* $\vdash_{g_s}^{g_t} C : \mathsf{E}, \mathsf{I}$ *and if* $(\sigma', \delta') \in \mathsf{Final}[\![(\mathsf{E}, \mathsf{I}[q_s \mapsto g_s][q_t \mapsto g_t] : q_s \mapsto q_t]\!](\sigma, \delta)$ *then* $\exists t \geq 0 : \{x \mid \sigma(x) \neq \sigma'(x)\} \cup \{r \mid \delta(r) + t \neq \delta'(r)\} \subseteq \mathsf{ass}(C)$, *where* t *corresponds to the initial delay.*

Next, the auxiliary operation $\mathsf{fst}_{g_s}^{g_t}(T)$ determines the initial guard and the invariant immediately following it (in the manner of the rule for assignment):

$$\begin{array}{rcl}
\mathsf{fst}_{g_s}^{g_t}(g \rightarrow \boldsymbol{x} := \boldsymbol{a} \colon \boldsymbol{r}) & = & g_s \wedge g \wedge g_t[\boldsymbol{a}/\boldsymbol{x}][\boldsymbol{0}/\boldsymbol{r}] \\
\mathsf{fst}_{g_s}^{g_t}(T;^{[g]} C) & = & \mathsf{fst}_{g_s}^{g}(T)
\end{array}$$

The inclusion of g_s is so as to get the strongest information for use in the rule for the looping construct in Table 3.2.

We shall need the auxiliary predicate $\Phi^{T_1,\cdots,T_n}_{T_{n+1},\cdots,T_m}\left[{}^{q_t:g_t}_{q_s:g_s}\right]$ that must be true whenever it is possible that the construct $\texttt{do}\ T_1\ [\!]\ \cdots\ [\!]\ T_n\ \texttt{od}\ [\!]\ T_{n+1}\ [\!]\ \cdots\ [\!]\ T_m$ does *not* terminate from a state satisfying g_s; we return to this below.

Looping We can now explain the inference rule in Table 3.2 for looping. The first line in the side condition expresses that the termination relies on having started the action as we saw in the axiom for assignment. The second line in the side condition takes care of imposing the latent effect χ_i of the looping commands on the loop header q_s.

The third line in the side condition takes care of imposing the latent effect of the terminating commands on the final node q_t. However, by using the predicate $\Phi^{T_1,\cdots,T_n}_{T_{n+1},\cdots,T_m}\left[{}^{q_t:g_t}_{q_s:g_s}\right]$ we dispense with imposing this latent effect in case termination of the looping construct is guaranteed. As an example this means that the type system will allow the following timed command

$$\big(\ (\mathtt{h} = 0 \rightarrow \mathtt{h}{:=}\mathtt{h}{:}\ )\ [\!]\ (\mathtt{h} \neq 0 \rightarrow \mathtt{h}{:=}\mathtt{h}{:}\ )\ \big)\ ;^{[\mathsf{tt}]}\mathsf{tt} \rightarrow \mathtt{l}{:=}\mathtt{l}{:}$$

that would otherwise be disallowed (assuming that $\mathtt{h}$ is a high variable and $\mathtt{l}$ is a low variable). Indeed it is in order to accommodate this kind of behaviour that the type system makes use of latent variables and nodes. This is essential for preventing unnecessary timing channels created from non-termination which depends on high variables.

Using the notation of Table 3.2 we can now clarify our demands on the auxiliary notation $\Phi^{T_1,\cdots,T_n}_{T_{n+1},\cdots,T_m}\left[{}^{q_t:g_t}_{q_s:g_s}\right]$ used in the third line:

$$\begin{array}{l} \perp \in \bigcup_{(\sigma,\delta)\mid[\![g_s]\!](\sigma,\delta)} \mathsf{Final}[\![(\cup_i \mathsf{E}_i, \cup_i \mathsf{I}_i[q_s \mapsto g_s][q_t \mapsto g_t]) : q_s \mapsto q_t]\!](\sigma,\delta) \\ \Downarrow \\ \Phi^{T_1,\cdots,T_n}_{T_{n+1},\cdots,T_m}\left[{}^{q_t:g_t}_{q_s:g_s}\right] \end{array}$$

The subscript $(\sigma,\delta) \mid [\![g_s]\!](\sigma,\delta)$ is intended to let (σ,δ) range over all possibilities that satisfy $[\![g_s]\!](\sigma,\delta)$. Note that we do not require to capture non-termination precisely but will allow any over-approximation.

Before explaining the fourth line in the side condition it is helpful to establish the following property of the type system as stated in Table 3.2.

LEMMA 3.2 *If* $\vdash^{[q_t:g_t]}_{[q_s:g_s]} C : \mathsf{E}, \mathsf{I} \,\&\, \chi$ *then we have that* $\{q_s\} \rightsquigarrow \mathsf{ass}(C) \cup \{q_t\}$ *and* $\forall \chi' : (\chi \rightsquigarrow \chi') \Rightarrow (\{q_s\} \rightsquigarrow \chi')$.

If $\vdash^{[q_t:g_t]}_{[q_s:g_s]} T : \mathsf{E}, \mathsf{I} \,\&\, \chi$ *then* $\forall \chi' : (\chi \rightsquigarrow \chi') \Rightarrow (\{q_s\} \cup \mathsf{fv}(\mathsf{fst}^{g_t}_{g_s}(T)) \rightsquigarrow \chi')$ *and* $\{q_s\} \cup \mathsf{fv}(\mathsf{fst}^{g_t}_{g_s}(T)) \rightsquigarrow \mathsf{ass}(T)$, *and* $\{q_s\} \rightsquigarrow \{q_t\}$.

(Note that the lack of reflexivity of $\rightsquigarrow$ means that we need to write slightly complex formula like $\forall\chi' : (\chi \rightsquigarrow \chi') \Rightarrow ((\cdots) \rightsquigarrow \chi')$ because the formula $((\cdots) \rightsquigarrow \chi$ is in general incorrect.)

This lemma shows that we have already taken care of the so-called *block labels* of [DD77] and thereby take care of the implicit flows due to testing guards in the manner of [VSI96]. However, the language considered in [VSI96] is deterministic and as we have already showed in Example 2.2 the presence of non-determinism poses a complication, giving rise to new information flows. For another example of such a flow, consider the following command:

$$\texttt{tt} \to \texttt{l:=0:}\ ;^{[\texttt{tt}]} \Big(\ (\texttt{h} = \texttt{0} \to \texttt{h:=h:}\ )\ []\ (\texttt{tt} \to \texttt{l:=1:}\ )\ \Big)$$

where $\texttt{l}$ is a low variable and $\texttt{h}$ is a high variable. Here the final value of $\texttt{l}$ will be 1 if $\texttt{h} \neq \texttt{0}$, however if $\texttt{h} = \texttt{0}$, the final value of $\texttt{l}$ may be either 0 or 1. This presents a violation for adherence to the non-interference condition.

The purpose of the fourth line in the side condition is to take care of this possibility and this is a novel contribution with respect to [DD77, VSI96, BBM94]. The notation $\underline{\text{sat}}(\cdots)$ is intended to express the satisfiability of the $\cdots$ formula. We are considering all terminating branches in the looping construct and whenever there are two branches that are not mutually exclusive (that is, where $\underline{\text{sat}}(\mathsf{fst}^{\zeta_i}_{g_s}(T_i) \wedge \mathsf{fst}^{\zeta_j}_{g_s}(T_j)))$ we make sure to record the information flow arising from *bypassing* the branch that would otherwise perform an assignment. This is essential for dealing with *non-determinism*.

Before explaining the fifth condition let us consider the following command operating on a low clock $\texttt{r}_l$, a high clock $\texttt{r}_h$ and a high variable $\texttt{h}$:

$$\texttt{r}_h \geq 50 \wedge \texttt{h} = 1 \to \texttt{skip:}\ []\ \texttt{r}_h \geq 100 \wedge \neg(\texttt{h} = 1) \to \texttt{skip:}$$

Take arbitrary σ, δ. Here we have that $(\sigma[\texttt{h} \mapsto 1], \delta[\texttt{r}_h \mapsto 50]) \equiv (\sigma[\texttt{h} \mapsto 0], \delta[\texttt{r}_h \mapsto 60])$ but running the command from $(\sigma[\texttt{h} \mapsto 1], \delta[\texttt{r}_h \mapsto 50])$ might produce $(\sigma[\texttt{h} \mapsto 1], \delta[\texttt{r}_h \mapsto 50])$ itself (and the initial and final value of $\texttt{r}_l$ are equal), whereas running the command from $(\sigma[\texttt{h} \mapsto 0], \delta[\texttt{r}_h \mapsto 60])$ can only produce $(\sigma[\texttt{h} \mapsto 0], \delta[\texttt{r}_h \mapsto 60] + t)$ for $t \geq 40$ (the initial and the final value of $\texttt{r}_l$ differ by at least 40 time units) in which case $(\sigma[\texttt{h} \mapsto 1], \delta[\texttt{r}_h \mapsto 50]) \not\equiv (\sigma[\texttt{h} \mapsto 0], \delta[\texttt{r}_h \mapsto 60] + t)$.

The purpose of the fifth line in the side condition is to take care of this possibility by enforcing that the terminating branches only test on low clocks, and that the conditions on clocks are the same whenever we test on high variables. This is essential to deal with *timing channels* (see Section 2.2).

To this end we define $\overline{g}$ as follows

$$\begin{array}{rcl} \overline{b} & = & \mathsf{tt} \\ \overline{r\,\mathsf{op_r}\,n} & = & r\,\mathsf{op_r}\,n \\ \overline{g_1 \wedge g_2} & = & \overline{g_1}\ \wedge\ \overline{g_2} \end{array}$$

and we write $g \Leftrightarrow g'$ to express the equivalence of the guards g and g'.

EXAMPLE 3.8 *Returning to Example 3.4 and the timed automaton of Figure 3.2, let us consider the command*

$$\mathsf{extra\_mult}\ []\ \mathsf{dummy\_mult}$$

From the first line we get the constraint $\{4\} \rightsquigarrow \{9\}$. *We have no contribution from the second line since there is no looping, and we also have no contribution from the third and forth line since the termination of the command is guaranteed and there is no pair of states satisfying the guards of both commands resp. Finally, from the fifth condition we get only that* $\mathcal{L}(\mathtt{r_1}) = \mathtt{L}$, *since* $\mathtt{r_1} \geq 2 \Leftrightarrow \mathtt{r_1} \geq 2$ *and thus*

$$\overline{\mathtt{r_1} \geq 2 \wedge \bigvee_{j=1}^{1024} (\mathtt{i} = j \wedge \mathtt{k}_j = 1)} \Leftrightarrow \overline{\mathtt{r_1} \geq 2 \wedge \neg \bigvee_{j=1}^{1024} (\mathtt{i} = j \wedge \mathtt{k}_j = 1)}$$

It is easy to check that the above conditions are fulfilled with the security assignment of Example 3.3.

Timed Commands Consider the last inference rule in Table 3.2. The first and last lines of the side condition are as in Table 3.1. The second and third lines of the side condition express the explicit flow possible due to the delay at the node $q_\circ$ and $q_\bullet$ and is analogous to our treatment of sequencing. The fourth line of the side condition takes care of imposing the latent effect of the command on the final node $q_\bullet$ and is analogous to our treatment of sequencing. The fifth line will allow us to invoke Theorem 3.3 of the next section.

EXAMPLE 3.9 *Consider the timed command*

$$\mathtt{begin}^{[\mathtt{r_g} \leq t_{end}]}\ T_{casting}\ []\ T_{ending}\ ^{[\mathsf{tt}]}\ \mathtt{end}$$

from Example 3.4 and its automaton of Figure 3.2. It is the case that $q_\circ = 1$, $q_\bullet = 5$, $g_\circ = \mathtt{r_g} \leq t_{end}$ *and* $g_\bullet = \mathsf{tt}$. *The type system imposes the following constraints on the flows:*

$$\{\mathtt{r_g}, 1\} \rightsquigarrow \{\mathtt{r_g}, \mathtt{r_1}, 1\}, \quad \{5\} \rightsquigarrow \{\mathtt{r_g}, \mathtt{r_1}, 5\}, \quad \{5\} \rightsquigarrow \{5\}, \quad \mathcal{L}(q_\bullet) = \mathtt{L}$$

It is easy to check that they are fulfilled for the security assignment of Example 3.3. Finally, using Example 3.6, Example 3.7 and Example 3.8 we can easily check that the body of the time command type checks.

3.5 Adequacy

To prove the adequacy of the type system we shall establish some terminology. A function like $\mathsf{Final}[\![\mathsf{TA} : q_s \mapsto q_t]\!]$ mapping a pair of state and clock assignment to a set of pairs of states and clock assignments, and possibly the symbol $\perp$ will be called a *semantic function*. Whenever F is a semantic function we define

$$F \models g_s \mapsto g_t \text{ iff } \forall(\sigma,\delta),(\sigma',\delta') : \begin{array}{c}(\sigma,\delta) \equiv_{g_s} (\sigma',\delta') \\ \Downarrow \\ F(\sigma,\delta) \equiv^{g_t} F(\sigma',\delta')\end{array}$$

where (using $\equiv$ as defined in Section 3.2)

$$\begin{array}{ll}(\sigma,\delta) \equiv_g (\sigma',\delta') \text{ abbreviates} & (\sigma,\delta) \equiv (\sigma',\delta') \ \wedge \ [\![g]\!](\sigma,\delta) \ \wedge \ [\![g]\!](\sigma',\delta') \\ H \equiv^g H' \text{ abbreviates} & H \equiv H' \ \wedge \\ & \forall(\sigma,\delta) \in H : [\![g]\!](\sigma,\delta) \ \wedge \ \forall(\sigma',\delta') \in H' : [\![g]\!](\sigma',\delta')\end{array}$$

The semantic condition for when a timed automaton $\mathsf{TA} = (\mathsf{Q},\mathsf{E},\mathsf{I},q_\circ,q_\bullet)$ satisfies non-interference with respect to a security policy $\mathcal{L}$ (see Definition 3.1) then amounts to $\mathsf{Final}[\![(\mathsf{E},\mathsf{I}) : q_\circ \mapsto q_\bullet]\!] \models \mathsf{I}(q_\circ) \mapsto \mathsf{I}(q_\bullet)$.

Finally, let us define the composition of two semantic functions F_1 and F_2 as follows:

$$\begin{array}{ll}F_1 \diamond F_2 & = \lambda(\sigma_0,\delta_0). \quad (F_1(\sigma_0,\delta_0) \cap \{\perp\}) \cup \\ & \qquad \bigcup_{(\sigma_1,\delta_1)\in F(\sigma_0,\delta_0)\setminus\{\perp\}} F_2(\sigma_1,\delta_1)\end{array}$$

Fact 2 *If $F_1 \models g_0 \mapsto g_1$ and $F_2 \models g_1 \mapsto g_2$ then $F_1 \diamond F_2 \models g_0 \mapsto g_2$.*

We are then ready to state a non-interference result in the manner of Definition 3.1:

Theorem 3.3 (Adequacy of Commands) *If $\vdash^{[q_t:g_t]}_{[q_s:g_s]} C : \mathsf{E},\mathsf{I} \,\&\, \chi$ and $\chi \rightsquigarrow \{q_t\}$ and $\mathcal{L}(q_t) = \mathsf{L}$ and $\mathsf{fv}(g_s) \rightsquigarrow \{q_s\}$ then we have $\mathsf{Final}[\![(\mathsf{E},\mathsf{I}[q_s \mapsto g_s][q_t \mapsto g_t]) : q_s \mapsto q_t]\!] \models g_s \mapsto g_t$.*

We can now establish our main result that the type system enforces a sufficient condition for the absence of information flows violating the security policy.

Corollary 3.4 (Adequacy) *If $\vdash \texttt{begin}^{[g_\circ]}\, C\, ^{[g_\bullet]}\texttt{end} : \mathsf{E},\mathsf{I},q_\circ,q_\bullet$ then we have that $\mathsf{Final}[\![(\mathsf{E},\mathsf{I}) : q_\circ \mapsto q_\bullet]\!] \models \mathsf{I}(q_\circ) \mapsto \mathsf{I}(q_\bullet)$.*

3.6 Related Work

There are many works dealing with information flow on systems with a notion of *discrete time*. The work of [FGM03] develops a non-interference property based on bisimulations of processes from a discrete-time process algebra. The works of [BMP19] and [ATM10] introduce a calculus as a semantic framework for dealing with information security in IoT devices and cyber-physical systems respectively. A language-based approach is taken in [Aga00], where a transformational type system is used to remove discrete timing as a covert channel for deterministic programs. However, as has already been noted in [BP18] discrete time may allow for some information flows to be undetected.

There are several papers that deal with information flow on systems with a notion of *dense time*. The work of [BFST02] and [BT03] define a notion of non-interference for timed automata with high-level (secret) and low-level (public) actions. Their notion of security is expressed as a non-interference property and it depends on a natural number m, representing a minimum delay between high-level actions such that the low-level behaviors are not affected by the high-level ones. The authors of [LMST10] define a notion of timed non-interference based on bisimulations for probabilistic timed automata which again is based on high-level and low-level actions. Similarly, the work of [GSB18] defines an action-based non-interference and uses model-checking techniques in order to enforce it on timed automata models. A somewhat different approach is taken in [GMR07] that studies the synthesis of controllers for achieving non-interference between high- and low-level actions. Our work differs from those works since we are concerned with timed automata that use data-variables and our non-interference notion is *state-based* instead.

There are other works that define security of timed-systems based on notions either different or somehow different from non-interference style conditions. The work of [LMMV19] propose a hybrid process calculus for the modelling of cyber-physical systems, and the security analysis of integrity and denial of service attacks. The work of [Cas09] presents timed-opacity as a generalization of non-interference in timed systems, and shows that verifying timed-opacity is in general undecidable; however the authors of [Cas09] do not consider approaches for enforcing timed-opacity. The work of [AS19] studies the following problem: given a timed automaton with timing parameters, a secret state and a final public state, synthesize the timing parameters and the execution times for which one cannot infer whether the system's execution has passed through the secret state. Finally, the work of [BP18] introduces a hybrid logic for verifying information flow security properties that depend both on discrete and continuous variables.

3.7 Conclusions

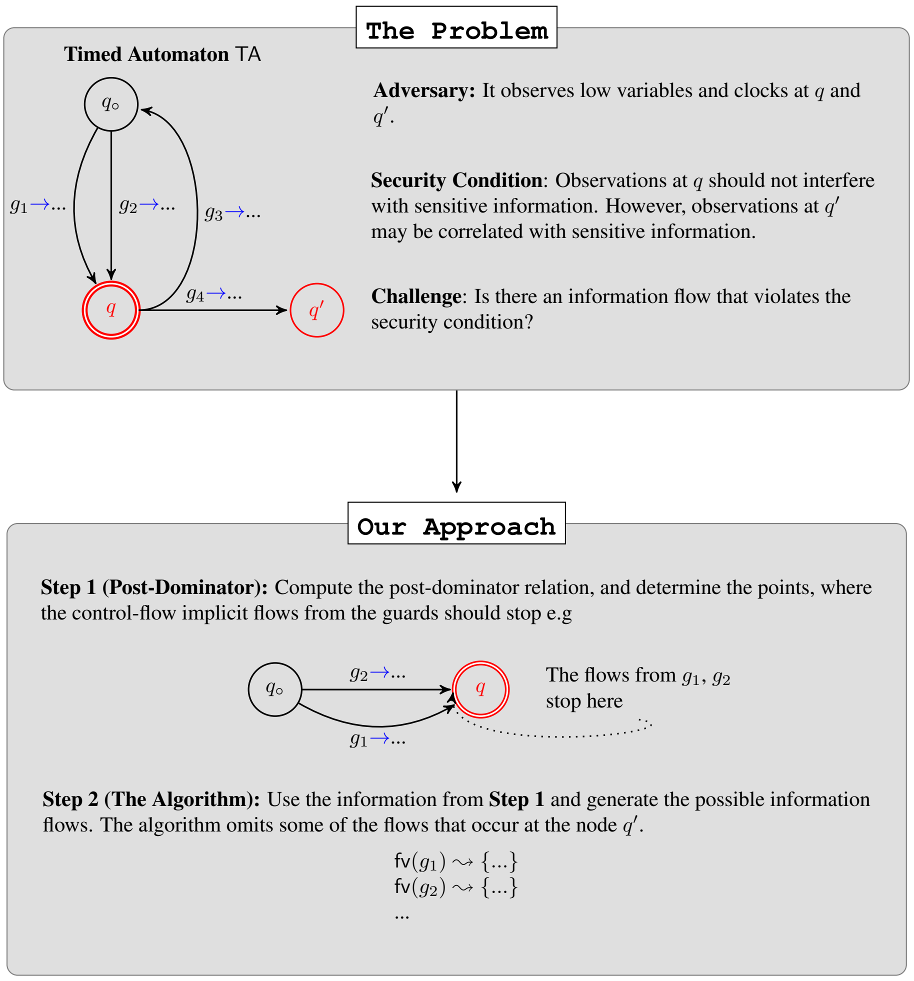
We have shown how to successfully merge timed automata with information flow and language-based security through the introduction of the timed commands language patterned after Dijkstra's guarded commands. This has facilitated developing a type system that prevents unnecessary *covert channels* and that deals with *non-determinism*, *non-termination* and *continuous real-time*. The type system has been proved adequate by means of a non-interference result.

CHAPTER 4

Secure Locality-Based Declassification

In this chapter, we develop a static analysis which allows one to enforce a security condition that permits locality-based declassification in timed automata. In particular, our security condition is a bisimulation relation on the configurations of the timed automaton, it captures adversaries that can make multiple observations, and permits the bypassing of a security policy whenever an execution reaches particular nodes. To check our security condition, we first define a general notion of the post-dominator relation [LT79], which allows us to determine the points (i.e the nodes) in the automaton where the propagation of an implicit flow should stop. We then develop a static analysis that is an algorithm which traverses a timed automaton and imposes information flow constraints using the post-dominator relation. Finally, we prove that whenever a timed automaton is certified by our algorithm then it satisfies our security condition. The idea of our approach is illustrated in Figure 4.1.

Chapter Organisation In Section 4.1, we model the smart grid system from Example 2.8 as a timed automaton, while we also introduce some auxiliary definitions for the traces of timed automata. In Section 4.2, we give our security condition, and in Section 4.3, we define our post-dominator relation. In Section 4.4, we give our static analysis. Finally, in Section 4.5 and Section 4.6, we finish with our related work and conclusions respectively.


The Problem

Timed Automaton TA

Adversary: It observes low variables and clocks at q and q'.

Security Condition: Observations at q should not interfere with sensitive information. However, observations at q' may be correlated with sensitive information.

Challenge: Is there an information flow that violates the security condition?

Our Approach

Step 1 (Post-Dominator): Compute the post-dominator relation, and determine the points, where the control-flow implicit flows from the guards should stop e.g

The flows from g_1, g_2 stop here

Step 2 (The Algorithm): Use the information from **Step 1** and generate the possible information flows. The algorithm omits some of the flows that occur at the node q'.

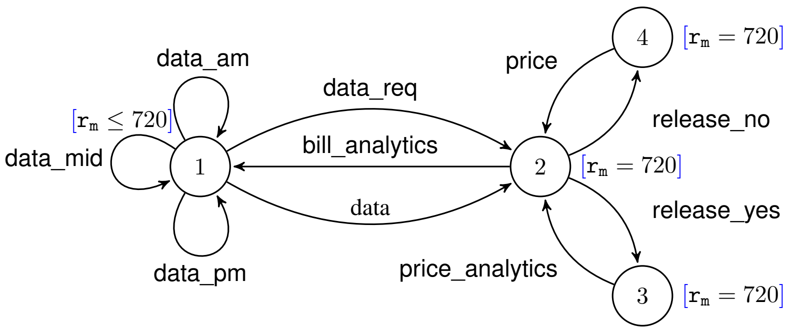
$\mathsf{fv}(g_1) \rightsquigarrow \{...\}$
$\mathsf{fv}(g_2) \rightsquigarrow \{...\}$
...

Figure 4.1: The idea of our development in Chapter 4.

4.1 Modelling the Smart Grid System

In Section 3.1 we introduced the model of a timed automaton, and we required that it has a final location $q_\bullet$. In this chapter, we drop this restriction, and a timed automaton is now given as quadruple $\mathsf{TA} = (\mathsf{Q}, \mathsf{E}, \mathsf{I}, q_\circ)$, where all of its components are defined as in Section 3.1.

We are now ready to give the timed automaton model of the smart grid system given



Data Aggregation

data_am:	$\mathtt{r_d} \leq 12 \wedge \mathtt{r_f} = 1 \rightarrow \mathtt{ed_{am}}{:=}\mathtt{ed_{am}} + \mathtt{ed}\text{: } \mathtt{r_f}$
data_pm:	$\mathtt{r_d} > 12 \wedge \mathtt{r_d} < 24 \wedge \mathtt{r_f} = 1 \rightarrow \mathtt{ed_{pm}}{:=}\mathtt{ed_{pm}} + \mathtt{ed}\text{: } \mathtt{r_f}$
data_mid:	$\mathtt{r_d} = 24 \wedge \mathtt{r_f} = 1 \rightarrow \mathtt{ed_{pm}}{:=}\mathtt{ed_{pm}} + \mathtt{ed}\text{: } (\mathtt{r_f}, \mathtt{r_d})$
data_req:	$\mathtt{r_m} = 720 \rightarrow (\mathtt{d}, \mathtt{c_{am}}, \mathtt{c_{pm}}){:=}(1, \mathtt{ed_{am}}, \mathtt{ed_{pm}})\text{:}$
data:	$\mathtt{r_m} = 720 \rightarrow (\mathtt{d}, \mathtt{c_{am}}, \mathtt{c_{pm}}){:=}(0, \mathtt{ed_{am}}, \mathtt{ed_{pm}})\text{:}$

Billing and Analytics

release_no	$\mathtt{d} = 0 \rightarrow \mathtt{skip}\text{:}$
release_yes	$\mathtt{d} = 1 \rightarrow (\mathtt{y_{am}}, \mathtt{y_{pm}}){:=}(\mathtt{c_{am}}, \mathtt{c_{pm}})\text{:}$
price_analytics	$\rightarrow (\mathtt{p_{am}}, \mathtt{p_{pm}}, \mathtt{a}, \mathtt{f}){:=}(\mathtt{v_{am}}, \mathtt{v_{pm}}, \mathtt{z}, 1)\text{:}$
price	$\rightarrow (\mathtt{p_{am}}, \mathtt{p_{pm}}, \mathtt{a}, \mathtt{f}){:=}(\mathtt{v_{am}}, \mathtt{v_{pm}}, 0, 1)\text{:}$
bill_analytics	$\mathtt{f} = 1 \rightarrow (\mathtt{b}, \mathtt{x}, \mathtt{ed_{am}}, \mathtt{ed_{pm}}, \mathtt{f}){:=}(\mathtt{p_{am}} \cdot \mathtt{c_{am}} + \mathtt{p_{pm}} \cdot \mathtt{c_{pm}}, \mathtt{a}, 0, 0, 0)\text{: } (\mathtt{r_m}, \mathtt{r_d}, \mathtt{r_f})$

Figure 4.2: The timed automaton SG of the smart grid system, and the abbreviations of its actions during the data aggregation phase, and during the billing and analytics phase.

in Example 2.8. Recall that it consists of a customer C, a utility company UC and a trusted third party TTP which is used for controlling the way UC is making use of C's data, in order to achieve data privacy.

EXAMPLE 4.1 *The timed automaton* SG *of the smart grid system is given in Figure 4.2*

Clocks*. The automaton uses three clocks. The clocks* $\mathtt{r_m}$ *and* $\mathtt{r_d}$ *measure the elapse of time within one month and one day respectively, while the clock* $\mathtt{r_f}$ *is used to regulate the frequency of the electricity measurements performed by the sensor, by allowing one measurement every full hour.*

Variables*. The variable* ed *contains the electricity data of the customer* C*, while the variables* $\mathtt{ed_{am}}$ *and* $\mathtt{ed_{pm}}$ *aggregate the data from* ed *between midnight and noon, and from noon to midnight respectively. The collectors* $\mathtt{c_{am}}$ *and* $\mathtt{c_{pm}}$ *are used by the trusted party* TTP *to collect the electricity data from* $\mathtt{ed_{am}}$ *and* $\mathtt{ed_{pm}}$ *respectively, while the variable* d *is used to hold the decision of the customer* C *with regards the analytics of his data. Whenever* C *decides for his data to be declassified to the utility* UC*, then the* *latter receives* C*'s data (i.e* $\mathtt{c_{am}}$ *and* $\mathtt{c_{pm}}$*) from* TTP*, and stores it in the variables* $\mathtt{y_{am}}$ *and* $\mathtt{y_{pm}}$ *(resp.). The bill for the customer is stored in the* b *variable, and it is calculated based on the electricity tariff values* $\mathtt{v_{am}}$ *and* $\mathtt{v_{pm}}$ *which are collected from the* TTP *in the price variables* $\mathtt{p_{am}}$ *and* $\mathtt{p_{pm}}$ *respectively. Finally, the flag* f *is used from the* TTP *to determine if it has collected all the information about the bill and the data analytics* a *of* C*, while* a *is stored in the variable* z *from* TTP *and in* x *from* C*.*

Transitions*. We begin at the initial location 1 with the data aggregation phase during the period of a month* ($\mathtt{r_m} \leq 720$)*. Here we have three different transitions that correspond to the two different time periods, midnight to noon (*$\mathtt{r_d} \leq 12$*), noon to midnight (*$\mathtt{r_d} > 12$*), and the exact moment of the end of a day (*$\mathtt{r_d} = 24$*). Within a day, a measurement is performed every one hour* ($\mathtt{r_f} = 1$)*. By the end of the day the last measurement is calculated, and the clock* $\mathsf{r_d}$ *is reset indicating the start of a new day. The aggregation phase ends at the end of the month* ($\mathtt{r_m} = 720$)*, and the trusted party* TTP *collects the electricity data from* C *together with his decision about analysing it. Next the automaton moves to location 2.*

At location 2, the billing and analytics phase starts. The TTP *requests from the* UC *the prices of the electricity tariffs for the two time periods of interest moving to location 4. Otherwise, if* C *has made a request for his data to be analysed (*$\mathtt{d} = 1$*),* TTP *also reveals the collected data to* UC *moving to location 3. Once the* TTP *receives everything* ($\mathtt{f} = 1$) *from* UC*, he calculates the bill for* C*, sends it to him together with the analysis result, the clocks and the variables of the meter are reset, the automaton moves to location 1, and a new month starts. For simplicity here we assume that all the calculations done by the* TTP *and the* UC *by the end of the month are being completed in zero time.*

The semantics of timed automata is the same as the ones defined in Section 3.1.1, while now we introduce some extra definitions that we will use in later sections of our development.

For a configuration $\langle q_s, \sigma, \delta\rangle$ and a node q_t, we write $\mathsf{Traces}[\![\mathsf{TA} : q_s \mapsto q_t]\!](\sigma, \delta)$ for the set of traces from $\langle q_s, \sigma, \delta\rangle$ to q_t. Let now the *length* of a trace be the number of transitions appearing in it (possibly this number could be ∞) we then make use of Lemma 1 of [HSW12] and we obtain the following

PROPOSITION 4.1 *For a pair* (σ, δ) *whenever* $\mathsf{Traces}[\![\mathsf{TA} : q_s \mapsto q_t]\!](\sigma, \delta)$ *contains only successful traces, then there exists a trace* $tr \in \mathsf{Traces}[\![\mathsf{TA} : q_s \mapsto q_t]\!](\sigma, \delta)$ *with maximal length.*

Proposition 4.1 shows that whenever a configuration $\langle q_s, \sigma, \delta\rangle$ is guaranteed to reach a configuration at the node q_t, then the number of transitions it takes is bounded.

Next, we define the *delay* $\Delta(tr)$ of a trace tr from $\langle q_s, \sigma, \delta\rangle$ to q_t, and we have that if tr is a successful trace

$$\langle q_s, \sigma, \delta\rangle = \langle q'_0, \sigma'_0, \delta'_0\rangle \xrightarrow{t_1} \cdots \xrightarrow{t_n} \langle q'_n, \sigma'_n, \delta'_n\rangle = \langle q_t, \sigma', \delta'\rangle$$

then

$$\Delta(tr) = \sum_{i=1}^{n} t_i$$

In the case of tr being an unsuccessful (finite or infinite) trace we have that

$$\Delta(tr) = \infty$$

One may think here that the delay of an unsuccessful *finite* trace from $\langle q_s, \sigma, \delta\rangle$ to q_t shouldn't be ∞, since we have a finite number of transitions, and we could have instead taken the sum of the times appearing in the transition arrows of the trace. However, our intention here is to capture that q_t is never reached, and thus "waiting" to reach q_t takes ∞ time.

Finally, for two pairs (σ_1, δ_1), (σ_2, δ_2), and two nodes q_s, q_t, whenever

$$\begin{aligned}&\forall tr_1 \in \mathsf{Traces}[\![\mathsf{TA} : q_s \mapsto q_t]\!](\sigma_1, \delta_1) : \\ &\quad \forall tr_2 \in \mathsf{Traces}[\![\mathsf{TA} : q_s \mapsto q_t]\!](\sigma_2, \delta_2) : \Delta(tr_1) = \Delta(tr_2) \text{ (and vice versa)}\end{aligned}$$

we say that (σ_1, δ_1) and (σ_2, δ_2) have the same *termination behaviour* with respect to q_s and q_t. Note that it is not necessarily the case that a pair (σ, δ) has the same termination behaviour as itself.

4.2 Y-Bisimulation Security

In this section, we define our security notion based on a bisimulation relation on the configurations of the timed automaton.

In particular, as in Section 3.2 we model the victim as an automaton that runs on some secret input. We assume the same adversary model as in Section 3.2, but with one core difference. In contrast to the adversary model from Section 3.2, the adversary now is able to observe information at intermediate nodes of the automaton, and not only at final nodes. Moreover, computations of the victim leading to some of those nodes are allowed to leak secret information, permitting in that way locality-based data declassification. Our security via bisimulation then says that computations of the automaton under two configurations that are indistinguishable to the adversary (i.e they are equal on the their public part) lead to distinguishable configurations only at nodes where declassification is allowed.

To this end, as in Section 3.2, for a timed automaton $\mathsf{TA} = (\mathsf{Q}, \mathsf{E}, \mathsf{I}, q_\circ)$ we assume a security mapping $\mathcal{L}$ associating clocks and variables[1] to a high $\mathtt{H}$ or low $\mathtt{L}$ security level.

EXAMPLE 4.2 *Returning to Example 4.1 of our smart grid system, we have that $\mathcal{L}$ maps the program variable* $\mathtt{ed}$ *of the electricity data, the variables* $\mathtt{e}_{\mathtt{am}}, \mathtt{e}_{\mathtt{pm}}$ *that store this data, the collectors* $\mathtt{c}_{\mathtt{am}}, \mathtt{c}_{\mathtt{pm}}$ *and the bill* $\mathtt{b}$ *to the security level* $\mathtt{H}$*, while the rest of the program variables and clocks are mapped to* $\mathtt{L}$.

We also assume that there exists a set of *observable nodes* $Y \subseteq \mathsf{Q}$, which are the nodes where the values of program variables and clocks with low security are observable by the adversary. The observable nodes will be described by the union of two *disjoint* sets Y_s and Y_w, where a node q in Y_s (Y_w resp.) will be called *strongly observable* (*weakly observable* resp.). Computations leading to weakly observable nodes, are allowed to bypass the security policy $\mathcal{L}$, whereas this is not the case for the ones leading to strongly observable nodes.

EXAMPLE 4.3 *For the smart grid automaton* SG *from Example 4.1 , we have the set of observable nodes $Y = \{2, 3, 4\}$, with the strongly observable ones being the nodes 2 and 4 ($Y_s = \{2, 4\}$), and the weakly one is the node 3 ($Y_w = \{3\}$). Node 3 is weakly observable because that's the place in the automaton where* TTP *is allowed to release the secret information of* C*. The only non-observable node here is the initial node 1, which describes the data aggregation phase that takes place at the customer's side* C.

Observable steps Since the values of low program variables and clocks are only observable at the nodes in Y, we collapse the transitions of the automaton that lead to non-observable nodes into one. Thus we have an *observable successful step*

$$\langle q_s, \sigma, \delta \rangle \overset{D}{\Longrightarrow}_Y \langle q_t, \sigma', \delta' \rangle$$

whenever there exists a successful trace tr

$$\langle q_s, \sigma, \delta \rangle = \langle q_0, \sigma_0, \delta_0 \rangle \xrightarrow{t_1} \cdots \xrightarrow{t_n} \langle q_n, \sigma_n, \delta_n \rangle = \langle q_t, \sigma', \delta' \rangle \qquad (n > 0)$$

from $\langle q_s, \sigma, \delta \rangle$ to q_t in TA and $q_t \in Y$, $D = \Delta(tr)$ and $\forall i \in \{1, ..., n-1\} : q_i \notin Y$.

And we have an *observable unsuccessful trace*

$$\langle q_s, \sigma, \delta \rangle \overset{\infty}{\Longrightarrow}_Y \perp$$

[1]Recall that, in Section 3.2, $\mathcal{L}$ was mapping also nodes to a security level, however this is not needed in this setting.

whenever there exists an unsuccessful finite trace

$$\langle q_s, \sigma, \delta\rangle = \langle q_0, \sigma_0, \delta_0\rangle \xrightarrow{t_1} \cdots \xrightarrow{t_n} \langle q_n, \sigma_n, \delta_n\rangle \qquad (n \geq 0)$$

or an unsuccessful infinite trace

$$\langle q_s, \sigma, \delta\rangle = \langle q_0, \sigma_0, \delta_0\rangle \xrightarrow{t_1} \cdots \xrightarrow{t_n} \langle q_n, \sigma_n, \delta_n\rangle \xrightarrow{t_{n+1}} \cdots$$

from $\langle q_s, \sigma, \delta\rangle$ to any of the nodes in Y and $\forall i > 0 : q_i \notin Y$. From now on a configuration γ will range over **Config** $\cup \{\perp\}$.

Next, as in Section 3.2 we write $(\sigma, \delta) \equiv (\sigma', \delta')$ to indicate that the two pairs are equal on low variables and low clocks, and our bisimulation relation is defined as follows

Definition 4.2 (Y**-Bisimulation**) For a timed automaton $\mathsf{TA} = (\mathsf{Q}, \mathsf{E}, \mathsf{I}, q_\circ)$, and a set of observable nodes $Y = Y_s \cup Y_w$, a relation $\simeq_Y \subseteq (\mathbf{Config} \cup \{\perp\}) \times (\mathbf{Config} \cup \{\perp\})$ will be called a $Y-$bisimulation relation if $\simeq_Y$ is symmetric, and we have that if $\gamma_1 = \langle q_1, \sigma_1, \delta_1\rangle \simeq_Y \langle q_2, \sigma_2, \delta_2\rangle = \gamma_2$ then

$$\begin{aligned}(\sigma_1, \delta_1) \equiv (\sigma_2, \delta_2) \Rightarrow \text{if } \gamma_1 \overset{D_1}{\Longrightarrow}_Y \gamma_1' \text{ then } \exists \gamma_2', D_2 :&\\ \gamma_2 \overset{D_2}{\Longrightarrow}_Y \gamma_2' \wedge \gamma_1' \simeq_Y \gamma_2' \wedge&\\ (\gamma_1' \neq \perp \wedge \gamma_2' \neq \perp) \Rightarrow ((\text{node}(\gamma_1') \in Y_w \wedge \text{node}(\gamma_2') \in Y_w) \vee&\\ \text{pair}(\gamma_1') \equiv \text{pair}(\gamma_2'))&\end{aligned}$$

where $\text{node}(\langle q, \sigma, \delta\rangle) = q$, $\text{pair}(\langle q, \sigma, \delta\rangle) = (\sigma, \delta)$. In addition to that, if $\gamma_1 \simeq_Y \gamma_2$ then

$$(\gamma_1 = \perp \Leftrightarrow \gamma_2 = \perp)$$

We write $\sim_Y$ for the union of all the Y-bisimulations and it is immediate that this definition of $\sim_Y$ is both a Y-bisimulation and an equivalence relation. Intuitively, when two configurations are related in $\sim_Y$, and they are low equivalent then they produce distinguishable pairs of states only at the weakly observable nodes. Otherwise, observations made at strongly observable nodes should be still indistinguishable. In both cases, the resulting configurations of two Y-bisimilar configurations should also be Y-bisimilar, meaning that even if we declassify data, later computations leading to strongly observable nodes should respect the security policy $\mathcal{L}$. Making use of $\sim_Y$ our security condition is then defined as

Definition 4.3 ($Y-$**Bisimulation Security**) For a timed automaton $\mathsf{TA} = (\mathsf{Q}, \mathsf{E}, \mathsf{I}, q_\circ)$, a set $Y = Y_s \cup Y_w$ of observable nodes, and a security policy $\mathcal{L}$, we say that TA satisfies Y-bisimulation security whenever:

$$\begin{aligned}&\forall q \in \{q_\circ\} \cup Y : \forall (\sigma, \delta), (\sigma', \delta') :\\ &\quad ([\![\mathsf{I}(q)]\!](\sigma, \delta) \ \wedge \ [\![\mathsf{I}(q)]\!](\sigma', \delta')) \Rightarrow \langle q, \sigma, \delta\rangle \sim_Y \langle q, \sigma', \delta'\rangle\end{aligned}$$

This condition ensures that the initial configurations and the ones with nodes in Y, where the adversary can make observations are related in $\sim_Y$. In addition, whenever the set of observable nodes Y_w is empty our notion of security coincides with standard definitions of non-interference where information does not flow from secret to public, and in particular, whenever there is only one strongly observable node, and there are no edges leaving from it, then we have the non-interference notion given in Definition 3.1.

4.3 Post-Dominators

The lack of having a structured language as the one of timed commands from Section 3.3 (but instead we have an arbitrary timed automaton graph) poses a challenge on determining the information flows between low and high entities. In particular, for the implicit flows arising from testing guards, it is challenging to find their end points (nodes), that are the points where the control flow is not dependent on the variables appearing in the guard anymore. To this end, we define a generalized version of the post-dominator relation, and the immediate post-dominator relation [LT79], which will help us to solve this problem.

A *path* π in a timed automaton $\mathsf{TA} = (\mathsf{Q}, \mathsf{E}, \mathsf{I}, q_\circ)$ is a finite $\pi = q_0 act_1 q_1 ... q_{n-1} act_n q_n$ ($n \geq 0$), or infinite $\pi = q_0 act_1 q_1 ... q_{n-1} act_n q_n ...$ sequence of alternating nodes and actions, such that $\forall i > 0 : (q_{i-1}, act_i, q_i) \in \mathsf{E}$. We say that a path is *trivial* if $\pi = q_0$ and we say that a node q belongs to the path π, or π contains q, and we will write $q \in \pi$, if there exists some i such that $q_i = q$. For a finite path $\pi = q_0 act_1 q_1 ... q_{n-1} act_n q_n$ we write $\pi(i) = q_i act_{i+1} q_{i+1} ... q_{n-1} act_n q_n$ ($i \leq n$) for the suffix of π that starts at the i-th position and we usually refer to it as the i-th suffix of π. Finally, for a node q and a set of nodes $Y \subseteq \mathsf{Q}$ we write

$$\Pi_{(q,Y)} = \{\pi \mid \pi = q_0 act_1 q_1 ... q_{n-1} act_n q_n : \begin{array}{l} n > 0 \wedge q_0 = q \wedge q_n \in Y \wedge \\ \forall i \in \{1, ..., n-1\} : q_i \notin Y\} \end{array}$$

for the set of all the non-trivial finite paths that start at q, end at a node y in Y and all the intermediate nodes of the path do not belong to Y. Our post-dominator definition is then the following

DEFINITION 4.4 (Y **Post-Dominators**) For a node q and a set of nodes $Y \subseteq \mathsf{Q}$ we define the set

$$\mathsf{pdom}_Y(q) = \{q' \mid \forall \pi \in \Pi_{(q,Y)} : q' \in \pi(1)\ \}$$

and whenever $q' \in \mathsf{pdom}_Y(q)$, we will say that q' is a Y *post-dominator* of q.

Intuitively, whenever a node q' is a Y post-dominator of a node q, it means that every non-trivial path that starts at q has to visit q' before it visits one of the nodes in Y. We write $\mathsf{pdom}_y(q)$ whenever $Y = \{y\}$ is a singleton and we have the following facts

FACT 3 *For a set of nodes $Y \subseteq \mathsf{Q}$ and for a node q we have that*

$$\mathsf{pdom}_Y(q) = \bigcap_{y \in Y} \mathsf{pdom}_y(q)$$

FACT 4 *The post-dominator set for a singleton set $\{y\}$ can be computed by finding the greatest solution of the following data-flow equations:*

$$\begin{array}{ll}
\mathsf{pdom}_y(q) = \mathsf{Q} & \textit{if } \Pi_{(q,\{y\})} = \emptyset \\
\mathsf{pdom}_y(q) = \{y\} & \textit{if } y \in \mathsf{succ}(q) \\
\mathsf{pdom}_y(q) = \bigcap_{q' \in \mathsf{succ}(q)} \left(\{q'\} \cup \mathsf{pdom}_y(q')\right) & \textit{otherwise}
\end{array}$$

where $\mathsf{succ}(q)$ is the set of the immediate successors of q.

For a node q, we are interested in finding the Y post-dominator "closest" to it. For this, we give the definition of an *immediate post-dominator* as follows

DEFINITION 4.5 (**Immediate Y Post-Dominator**) For a node q and a set of nodes Y we define the set

$$\begin{aligned}
\mathsf{ipdom}_Y(q) = \{q' \in \mathsf{pdom}_Y(q) \mid\ & \mathsf{pdom}_Y(q) = \{q'\} \vee \\
& q' \notin Y \wedge (\forall q'' \in \mathsf{pdom}_Y(q) : q'' \neq q' \Rightarrow \\
& \qquad q'' \in \mathsf{pdom}_Y(q'))\}
\end{aligned}$$

and a node $q' \in \mathsf{ipdom}_Y(q)$ will be called an *immediate Y post-dominator* of q.

The following fact gives us a unique immediate Y post-dominator for the nodes that can reach Y ($\Pi_{(q,Y)} \neq \emptyset$). Intuitively this unique immediate Y post-dominator of a node q is the node that is the "closest" Y post-dominator of q, meaning that in any non-trivial path starting from q and ending in Y, the Y immediate post-dominator of q will always be visited first before any other Y post-dominator of q.

FACT 5 *For a set of nodes Y and a node q, whenever $\Pi_{(q,Y)} \neq \emptyset$, and $\mathsf{pdom}_Y(q) \neq \emptyset$, then there exists node q' such that $\mathsf{ipdom}_Y(q) = \{q'\}$.*

For simplicity, whenever a node q' is the unique immediate Y post-dominator of a node q and $\Pi_{(q,Y)} \neq \emptyset$ we shall write $\mathsf{ipd}_Y(q)$ for q' and we will say that *the unique* *immediate Y post-dominator of q is defined.* For any other case where q can either not reach Y ($\Pi_{(q,Y)} = \emptyset$) or $\mathsf{pdom}_Y(q) = \emptyset$ we will say that *the unique immediate post-dominator of q is not defined.*

EXAMPLE 4.4 *For the timed automaton SG from Example 4.1 and for the set of observable nodes $Y = \{2, 3, 4\}$ from Example 4.3, we have that $\mathsf{pdom}_Y(q) = \mathsf{ipd}_Y(q) = \{2\}$ for q being 1, 3 and 4, while $\mathsf{pdom}_Y(2) = \emptyset$. Therefore for the nodes 1,3 and 4 their unique immediate Y post-dominator is defined, and it is the node 2, while the unique immediate Y post-dominator of the node 2 is not defined.*

4.4 Algorithm for Secure Declassification

We develop an algorithm (given in Table 4.1) that traverses the graph of a timed automaton TA, and imposes information flow constraints (of the form $\chi \rightsquigarrow \chi'$ as in Section 3.2) on the program variables and clocks of the automaton, with respect to a security policy $\mathcal{L}$, and a Y post-dominator relation, where $Y = Y_s \cup Y_w$ is the set of observable nodes. Before we explain the algorithm we start by defining some auxiliary operators.

Auxiliary Operators For an edge $(q_s, g \rightarrow \boldsymbol{x} := \boldsymbol{a} \colon \boldsymbol{r}, q_t) \in \mathsf{E}$ we define the auxiliary operator $\mathsf{ass}(.)$, $\mathsf{expr}(.)$ and $\mathsf{con}(.)$ as

$$\begin{array}{rcl}
\mathsf{ass}((q_s, g \rightarrow \boldsymbol{x} := \boldsymbol{a} \colon \boldsymbol{r}, q_t)) & = & \{\boldsymbol{x}, \boldsymbol{r}\} \\
\mathsf{expr}((q_s, g \rightarrow \boldsymbol{x} := \boldsymbol{a} \colon \boldsymbol{r}, q_t)) & = & \{\boldsymbol{a}\} \\
\mathsf{con}((q_s, g \rightarrow \boldsymbol{x} := \boldsymbol{a} \colon \boldsymbol{r}, q_t)) & = & \mathsf{I}(q_s) \wedge g \wedge \mathsf{I}(q_t)[\boldsymbol{a}/\boldsymbol{x}][\boldsymbol{0}/\boldsymbol{r}]
\end{array}$$

where $\mathsf{ass}(.)$ gives the modified variables and clocks of the assignment performed by TA using that edge, $\mathsf{expr}(.)$ gives the expressions used for the assignment, and the operator $\mathsf{con}(.)$ returns the condition that has to hold in order for the assignment to be performed. We lift the $\mathsf{ass}(.)$ operator to operate on finite paths, and thus for a path $\pi = q_0 act_1 q_1 ... q_{n-1} act_n q_n$ we define the auxiliary operator $\mathsf{Ass}(.)$ as

$$\mathsf{Ass}(q_0 act_1 q_1 ... q_{n-1} act_n q_n) \quad = \quad \bigcup_{i=1}^{n} \mathsf{ass}((q_{i-1}, act_i, q_i))$$

Next, we write

$$\mathsf{Q}_{\rightsquigarrow \mathsf{w}} = \{q \mid \forall \pi = q..q' \in \Pi_{(q,Y)} : q' \in Y_w\}$$

for the set of nodes, where whenever the automaton performs a successful observable step starting from a node $q \in \mathsf{Q}_{\rightsquigarrow \mathsf{w}}$, and ending in an observable node $q' \in Y$, it is always the case that q' is weakly observable.

C1. For all $q \in \mathsf{Q}_{\rightsquigarrow w}$:
(a) $\quad \bigwedge_{e\in \mathsf{E}_q} \forall y \in \mathsf{fv}(\mathsf{con}(e)) : \mathcal{L}(y) = \mathsf{L} \wedge (\Psi_e \vee A_e)$
C2. For all $q \in \mathsf{Q}^c_{\rightsquigarrow w}$ such that their unique immediate Y post-dominator is defined :
(a) $\quad \bigwedge_{e\in \mathsf{E}_q} \bigcup_{\pi\in\Pi_{(e,\{\mathsf{ipd}_Y(q)\})}} \mathsf{fv}(\mathsf{con}(e)) \rightsquigarrow \mathsf{Ass}(\pi) \wedge A_e$
(b) $\quad \bigwedge_{e\neq e' : e\in \mathsf{E}_q,\, e'\in \mathsf{E}_q,\, \underline{\mathrm{sat}}(\mathsf{con}(e)\wedge \mathsf{con}(e'))} \mathsf{fv}(\mathsf{con}(e)) \rightsquigarrow \bigcup_{\pi\in\Pi_{(e',\{\mathsf{ipd}_Y(q)\})}} \mathsf{Ass}(\pi)$
(c) $\quad \Phi_q \Rightarrow \bigwedge_{e\in \mathsf{E}_q} \forall y \in \mathsf{fv}(\mathsf{con}(e)) : \mathcal{L}(y) = \mathsf{L}$
C3. For all $q \in \mathsf{Q}^c_{\rightsquigarrow w}$ such that their unique immediate Y post-dominator is not defined :
(a) $\quad \bigwedge_{e\in \mathsf{E}_q} \forall y \in \mathsf{fv}(\mathsf{con}(e)) : \mathcal{L}(y) = \mathsf{L} \wedge$
(b) $\quad ((e \rightsquigarrow w \wedge \Psi_e) \vee A_e)$

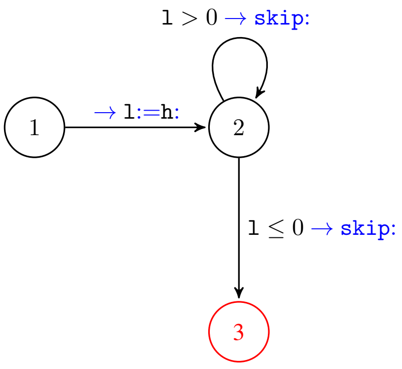
Table 4.1: Security of $\mathsf{TA} = (\mathsf{Q}, \mathsf{E}, \mathsf{I}, q_\circ)$ with respect to $\mathcal{L}$ and the Y post-dominator relation.

Condition C1 We start by looking at the nodes in $\mathsf{Q}_{\rightsquigarrow \mathsf{w}}$. According to our security notion (given in Definition 4.3), for two low equivalent configurations at a node q, whenever the first one performs a successful (or unsuccessful) observable step that ends at a weakly observable node, then also the second should be able to perform an observable step that ends at a weakly observable node (or an unsuccessful one resp.). For that, the condition **C1** (a) first requires that the guards of the outgoing edges in E_q, where

$$\mathsf{E}_q = \{(q, act, q') \mid (q, act, q') \in \mathsf{E}\}$$

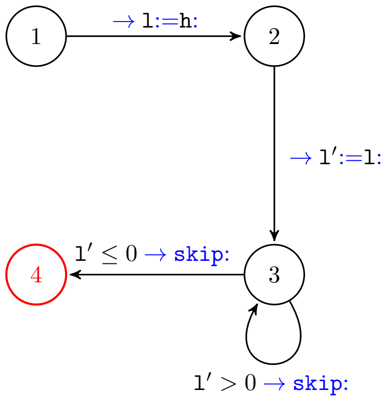
contain only low variables. However, this is not enough.

To explain the rest of the constraints imposed by the condition **C1** (a), consider the following automaton



where the node 3 is weakly observable, h and l are a high and a low variable respectively, and all the invariants of the nodes are set to tt. This automaton is not secure with respect to Definition 4.3. To see this, we have $([\mathsf{l} \mapsto 0, \mathtt{h} \mapsto 1], \delta) \equiv ([\mathsf{l} \mapsto 0, \mathtt{h} \mapsto 0], \delta)$ (for some clock state δ) but the pair $([\mathtt{l} \mapsto 0, \mathtt{h} \mapsto 1], \delta)$ always produces $\perp$ since we will have an infinite loop at the node 2, whereas $([\mathtt{l} \mapsto 0, \mathtt{h} \mapsto 0], \delta)$ always terminates at the node 3. This is because even if both edges of the node 2 contain only the low variable l in their condition, the assignment l:=h bypasses the policy $\mathcal{L}$ and thus, right after it, the two pairs stop being low equivalent.

As another example, consider the automaton



1
$\rightarrow$ l:=h:
2
$\rightarrow$ l′:=l:
4
$\mathtt{l}' \le 0 \rightarrow$ skip:
3
$\mathtt{l}' > 0 \rightarrow$ skip:

Here the node 4 is weakly observable, h is a high variable, l, l′ are two low variables, and all the invariants of nodes are set to tt again. We have $([\mathtt{l} \mapsto 0, \mathtt{l}' \mapsto 0, \mathtt{h} \mapsto 1], \delta) \equiv ([\mathtt{l} \mapsto 0, \mathtt{l}' \mapsto 0, \mathtt{h} \mapsto 0], \delta)$ (for some clock state δ) and again the first pair produces $\perp$ by looping at the node 3, whereas the second pair always terminates. Here even if the variable l is not used in any condition after the assignment l:=h, it influences the value of l′ and consequently, since l′ appears on the condition of the edges of the node 3 we get this behavior.

To cater for such cases, for an edge $e = (q_s, g \rightarrow \boldsymbol{x} := \boldsymbol{a}\colon \boldsymbol{r}, q_t)$ we first define the predicate

$$A_e = \bigwedge_i \mathsf{fv}(a_i) \rightsquigarrow \{x_i\}$$

that takes care of the explicit flows arising from the assignments. We then define

$$\Pi_{(e,Y)} = \{\pi \mid e = (q_0, act_1, q_1) : \pi = q_0 act_1 q_1 ... q_{n-1} act_n q_n \in \Pi_{(q_0,Y)}\}$$

to be the set of paths (the ones defined in Section 4.3) that start with e and end in Y, and all the intermediate nodes do not belong to Y. Finally, whenever an assignment bypasses the security policy $\mathcal{L}$ due to an explicit flow and consequently A_e is false, we then impose the predicate

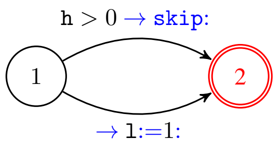
$$\begin{aligned}\Psi_e = \forall \pi \in \Pi_{(e,Y)} : \forall q' \in \pi(1) : \\ q' \notin Y \Rightarrow (\forall e' \in E_{q'} : (\mathsf{ass}(e) \setminus \textbf{Clocks}) \cap (\mathsf{fv}(\mathsf{con}(e')) \cup \mathsf{fv}(\mathsf{expr}(e'))) = \emptyset)\end{aligned}$$

The predicate Ψ_e demands that the assigned program variables of $e = (q_s, act, q_t)$ cannot be used in any expression or condition that appears in a path that starts with q_t and goes to an observable node. Note here that even if Ψ_e quantifies over a possibly infinite set of paths ($\Pi_{(e,Y)}$), it can be computed in *finite time* by only looking at the paths where each cycle occurs at most once.

We now look at the nodes where the automaton may perform a successful observable step that ends in a strongly observable node. Those nodes are described by the set $\mathsf{Q}^c_{\rightsquigarrow w} = \mathsf{Q} \setminus \mathsf{Q}_{\rightsquigarrow \mathsf{w}}$, that is the complement of $\mathsf{Q}_{\rightsquigarrow \mathsf{w}}$.

Condition C2 For a node q in $\mathsf{Q}^c_{\rightsquigarrow w}$, whose immediate Y post-dominator is defined, condition **C2** (a) takes care of the explicit and the implicit flows generated by the assignment and the control dependencies respectively, arising from the edges of q. Note here that we do not propagate the implicit flows any further after $\mathsf{ipd}_Y(q)$. This is because $\mathsf{ipd}_Y(q)$ is the point where all the branches of q are joining and any further computation is not control-dependent on them anymore.

Next, condition **C2** (b) is essential for dealing with flows due to the non-deterministic semantics of timed automata (see Section 3.4). We recall such a flow by the following example automaton



where `h` and `l` is a high and a low variable respectively, the node 2 is strongly observable, and both nodes 1 and 2 have their invariant set to `tt`. Next take $([\mathsf{l} \mapsto 0, \mathtt{h} \mapsto 1], \delta) \equiv ([\mathsf{l} \mapsto 0, \mathtt{h} \mapsto 0], \delta)$ (for some clock state δ) and note that the first pair can result in a configuration in 2 with $([\mathtt{l} \mapsto 0, \mathtt{h} \mapsto 1], \delta)$ (taking the top branch) while the second pair always ends in 2 with $[\mathtt{l} \mapsto 1, \mathtt{h} \mapsto 0]$. Therefore this automaton is not secure with respect to our Definition 4.3.

Recall that $\underline{\mathrm{sat}}(\cdots)$ expresses the satisfiability of the $\cdots$ formula. Whenever there are two branches (induced by the edges e and e' both leaving q) that are not mutually exclusive, that is that $\underline{\text{sat}}(\text{con}(e) \wedge \text{con}(e'))$, we make sure to record the information flow arising from *bypassing* the branch that would otherwise perform an assignment.

Next, the following fact shows that our Ass(.) operator used in condition **C2**, correctly approximates the modified clocks and variables along a path.

FACT 6 *For a timed automaton* $\mathsf{TA} = (\mathsf{Q}, \mathsf{E}, \mathsf{I}, q_\circ)$*, we have that if*

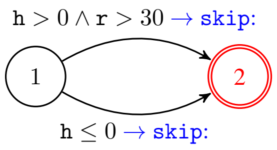
$$\langle q, \sigma, \delta \rangle \xRightarrow{D}_{\{q'\}} \langle q', \sigma', \delta' \rangle$$

then

$$\{x \mid \sigma(x) \neq \sigma'(x)\} \cup \{r \mid \delta'(r) \neq \delta(r) + D\} \subseteq \bigcup_{\pi \in \Pi_{(e,\{q'\})}} \mathsf{Ass}(\pi)$$

where e *corresponds to the initial edge of this observable step.*

Finally, condition **C2** (c) takes care of cases where a *timing* covert channel (see Section 3.4) could have occurred. We recall such a case with the following automaton



where $\mathtt{h}$ and $\mathtt{r}$ is a high program variable and a low clock respectively, node 2 is strongly observable and both 1 and 2 have their invariant set to $\mathtt{tt}$. Next, for $([\mathtt{h} \mapsto 1], [\mathtt{r} \mapsto 0]) \equiv ([\mathtt{h} \mapsto 0], [\mathtt{r} \mapsto 0])$ we have that the first pair always delays at least 30 units and ends in 2 with a clock state that has $\mathtt{r} > 30$, whereas the second pair can go to 2 taking the lower branch immediately without any delay, and thus the resulting pairs will not be low equivalent. To take care of such behaviours, we stipulate a predicate Φ_q such that

$$\begin{array}{l} \exists tr_1, tr_2 \in \bigcup_{(\sigma,\delta):[\![\mathsf{I}(q)]\!](\sigma,\delta)} \mathsf{Traces}[\![\mathsf{TA} : q \mapsto \mathsf{ipd}_Y(q)]\!](\sigma,\delta) : \Delta(tr_1) \neq \Delta(tr_2) \\ \Downarrow \\ \Phi_q \end{array}$$

Using this predicate we demand that whenever the TA does not have a "constant" *termination behavior* from the node q to the node $\mathsf{ipd}_Y(q)$, then variables that influence the termination behavior should not be of high security level.[2]

[2]In Section 3.4, the predicate Φ was used only for detection of timing channels, which were created due to non-termination. Here we refine the predicate Φ to Φ_q, so it caters for any timing channel.

Condition C3 We are now left with the nodes in $\mathsf{Q}^c_{\rightsquigarrow w}$, whose immediate Y post-dominator is not defined. Since for such a node q, we cannot find a point (the unique immediate Y post-dominator) where the control dependencies from the branches of q end, condition **C3** (a) requires that the conditions of the edges of q should not be dependent on high security variables.

Condition **C3** (b) caters for the explicit flows, of an edge e using the predicate A_e. However we are allowed to dispense A_e, whenever further computations after taking the edge e may lead only to weakly observable nodes and Ψ_e holds. To express this, for an edge $e = (q_s, g \rightarrow \boldsymbol{x} := \boldsymbol{a} \colon \boldsymbol{r}, q_t)$ we write

$$e \rightsquigarrow w$$

whenever $q_t \in Y_w$ or $q_t \in \mathsf{Q}_{\rightsquigarrow \mathsf{w}}$.

Example 4.5 *Consider now the automaton* SG *of Example 4.1, and the Y post-dominator relation of Example 4.4.*

*We have that the nodes 1, 3 and 4 are in $\mathsf{Q}^c_{\rightsquigarrow w}$ and also that their immediate unique Y post-dominator is defined. Condition **C2** (a) and **C2** (b) impose the following constraints*

$$\{\mathtt{r_m}, \mathtt{r_d}, \mathtt{r_f}\} \rightsquigarrow \{\mathtt{ed}, \mathtt{e_{am}}, \mathtt{e_{pm}}, \mathtt{c_{am}}, \mathtt{c_{pm}}, \mathtt{d}, \mathtt{r_d}, \mathtt{r_f}\}, \{\mathtt{ed}, \mathtt{e_{am}}\} \rightsquigarrow \{\mathtt{e_{am}}\},$$
$$\{\mathtt{ed}, \mathtt{e_{pm}}\} \rightsquigarrow \{\mathtt{e_{pm}}\}, \{\mathtt{e_{am}}\} \rightsquigarrow \{\mathtt{c_{am}}\}, \{\mathtt{e_{pm}}\} \rightsquigarrow \{\mathtt{c_{pm}}\}, \{\mathtt{v_{am}}\} \rightsquigarrow \{\mathtt{p_{am}}\}, \{\mathtt{v_{pm}}\} \rightsquigarrow \{\mathtt{p_{pm}}\},$$
$$\{\mathtt{r_m}\} \rightsquigarrow \{\mathtt{p_{am}}, \mathtt{p_{pm}}, \mathtt{a}, \mathtt{f}\}, \{\mathtt{z}\} \rightsquigarrow \{\mathtt{a}\}, \{\} \rightsquigarrow \{\mathtt{d}, \mathtt{f}, \mathtt{a}\}$$

Finally, for the node 1*, because* Φ_1 *(condition **C2** (c)) all the clocks need to be of low security level.*

*Next, the node 2 is in $\mathsf{Q}^c_{\rightsquigarrow w}$ and since its unique immediate Y post-dominator is not defined, condition **C3** (b) impose the constraints*

$$\{\mathtt{p_{am}}, \mathtt{p_{pm}}, \mathtt{c_{am}}, \mathtt{c_{pm}}\} \rightsquigarrow \{\mathtt{b}\}, \{\mathtt{a}\} \rightsquigarrow \{\mathtt{x}\}, \{\} \rightsquigarrow \{\mathtt{e_{am}}, \mathtt{e_{pm}}, \mathtt{f}\}$$

*and condition **C3** (a) imposes that $\mathtt{r_m}$, $\mathtt{d}$ and $\mathtt{f}$ should be of low security level. Notice here that since for the edge $e = (2, \mathtt{d} = 1 \rightarrow (\mathtt{y_{am}}, \mathtt{y_{pm}}) := (\mathtt{c_{am}}, \mathtt{c_{pm}}) \colon, 3)$ that releases the sensitive information of C we have that $e \rightsquigarrow w$, we are not imposing the constraints $\{\mathtt{c_{am}}\} \rightsquigarrow \{\mathtt{y_{am}}\}$ and $\{\mathtt{c_{pm}}\} \rightsquigarrow \{\mathtt{y_{pm}}\}$. All those constraints are easy to verify for the security assignment of Example 4.2.*

Now if we were to change the node 3 from being a weakly observable to a strongly observable node, the automaton SG *will not be secure with respect to Definition 4.3. In that case our algorithm will reject it, since for the edge e we would have that $e \not\rightsquigarrow w$ and the predicate A_e would have resulted in false.*

Finally, we write $\text{sec}_{Y,\mathcal{L}}(\mathsf{TA})$ whenever the constraints arising from our algorithm (given in Table 4.1) are satisfied, and we then have the following theorems, which are the building stones for establishing the soundness of our algorithm.

THEOREM 4.6 *For a timed automaton* $\mathsf{TA} = (\mathsf{Q}, \mathsf{E}, \mathsf{I}, q_\circ)$, *if* $sec_{Y,\mathcal{L}}(\mathsf{TA})$ *then for* (σ_1, δ_1), (σ_2, δ_2) *such that* $[\![\mathsf{I}(q)]\!](\sigma_1, \delta_1)$ *and* $[\![\mathsf{I}(q)]\!](\sigma_2, \delta_2)$ *and* $(\sigma_1, \delta_1) \equiv (\sigma_2, \delta_2)$ *we have that*

$$\textit{if } \langle q, \sigma_1, \delta_1\rangle \overset{D_1}{\Longrightarrow}_Y \langle q', \sigma_1', \delta_1'\rangle \textit{ then } \exists(\sigma_2', \delta_2'), D_2 : \quad \begin{array}{l} \langle q, \sigma_2, \delta_2\rangle \overset{D_2}{\Longrightarrow}_Y \langle q', \sigma_2', \delta_2'\rangle \wedge \\ (q' \in Y_w \vee (\sigma_1', \delta_1') \equiv (\sigma_2', \delta_2')) \end{array}$$

THEOREM 4.7 *For a timed automaton* $\mathsf{TA} = (\mathsf{Q}, \mathsf{E}, \mathsf{I}, q_\circ)$, *if* $sec_{Y,\mathcal{L}}(\mathsf{TA})$ *then for* (σ_1, δ_1), (σ_2, δ_2) *such that* $[\![\mathsf{I}(q)]\!](\sigma_1, \delta_1)$ *and* $[\![\mathsf{I}(q)]\!](\sigma_2, \delta_2)$ *and* $(\sigma_1, \delta_1) \equiv (\sigma_2, \delta_2)$ *we have that*

$$\textit{if } \langle q, \sigma_1, \delta_1\rangle \overset{\infty}{\Longrightarrow}_Y \perp \textit{ then also } \langle q, \sigma_2, \delta_2\rangle \overset{\infty}{\Longrightarrow}_Y \perp$$

The following corollary concludes the two theorems from above to establish the soundness of our algorithm with respect to the notion of security of Definition 4.3.

COROLLARY 4.8 *For a timed automaton* $\mathsf{TA} = (\mathsf{Q}, \mathsf{E}, \mathsf{I}, q_\circ)$, *if* $sec_{Y,\mathcal{L}}(\mathsf{TA})$ *then* TA *satisfies* Y*-bisimulation security.*

4.5 Related Work

The work of this chapter can be seen as an extension of the work presented in Chapter 3. We are now capable of dealing with any timed automaton, and with adversaries that can make multiple observations at any node in the timed automaton. Whereas, in Chapter 3, we were restricted by the timed automata expressed from the constructs of the timed commands, while we were only concerned with adversaries that could make observations only at the final node of the timed automaton. Finally, the work in Chapter 3 does not support declassification of data.

Having said that, the related work presented in Section 3.6 also applies here with the same comparison, while there is one substantial difference, that is that, none of the works discussed in Section 3.6 considers policies that permit data declassification. To our knowledge, we are the first who considered the problem of locality-based declassification in a setting with systems of real-time semantics, such as timed automata.

Our notion of security is based on intransitive non-interference [LMP12, MS04, Pin95, RG99]. Whenever certain conditions are met, intransitive non-interference permits some flows which would have been otherwise disallowed from standard non-interference notions. In particular, our notion shares some ideas from [MS04, LMP12], where a bisimulation-based security is defined for a programming language with threads. In their approach, the bypassing of the security policy is localized on the actions of the program. An attacker is able to observe the low variables of a program at any of its computation steps (e.g in a timed automaton all of the nodes would have been observable). Moreover, discrete time is considered in [MS04, LMP12], and it is defined as the number of transitions a program takes. It is also considered that time is observed by the adversary. In contrast to [MS04, LMP12], we localize bypassing of policies at the level of the nodes, while we also define a more flexible notion of security with respect to the adversary's observability, since we allow for choosing on which nodes of the automaton the adversary makes observations. Finally, our semantics for time differs, since we are concerned with dense time.

Papers that deal with other dimensions of declassification e.g *what* data can be declassified, and *who* can declassify data are also relevant, and have been discussed in Section 2.3.

4.6 Conclusions

We have shown how to successfully enforce information flow policies, that permit controlled locality-based declassification on timed automata. For this, we developed an algorithm that prevents unnecessary *covert channels* and deals with *non-determinism*, *non-termination*, *continuous real-time*, and with adversaries that are allowed to observe computations on several locations in the timed automaton. Our algorithm makes use of a novel definition of post-dominators, which allows us to deal with the implicit flows arising from the unstructured control flow in timed automata. Finally, our algorithm has been proved sound by means of a bisimulation result.

Automation Our work lacks full automation mainly because of the predicate Φ_q and the $\underline{\text{sat}}(\cdots)$ operation.

In particular, recall that the predicate Φ_q is used for the detection of timing channels, by requiring that the time needed getting from the node q to its immediate post-dominator $\mathsf{ipd}_Y(q)$ should be constant. There has been a lot of research [ARF17, ARF16] done for determining the maximum (max_t), or minimum (min_t) execution time that a timed automaton needs to move from a location q_s to a location q_t. One possibility for over-approximating Φ_q is to make use of this work [ARF17, ARF16] and thus the predicate Φ_q would amount to checking if the execution time between the two nodes of interest (q and $\mathsf{ipd}_Y(q)$) is constant (i.e $max_t = min_t$). Next, to reduce the complexity of our algorithm, we could also relax our requirements on when to check the predicate Φ_q.

In particular, checking Φ_q only on nodes whose edges contain high variables (at least one) is sufficient, since if all the variables are low we get no contribution from Φ_q.

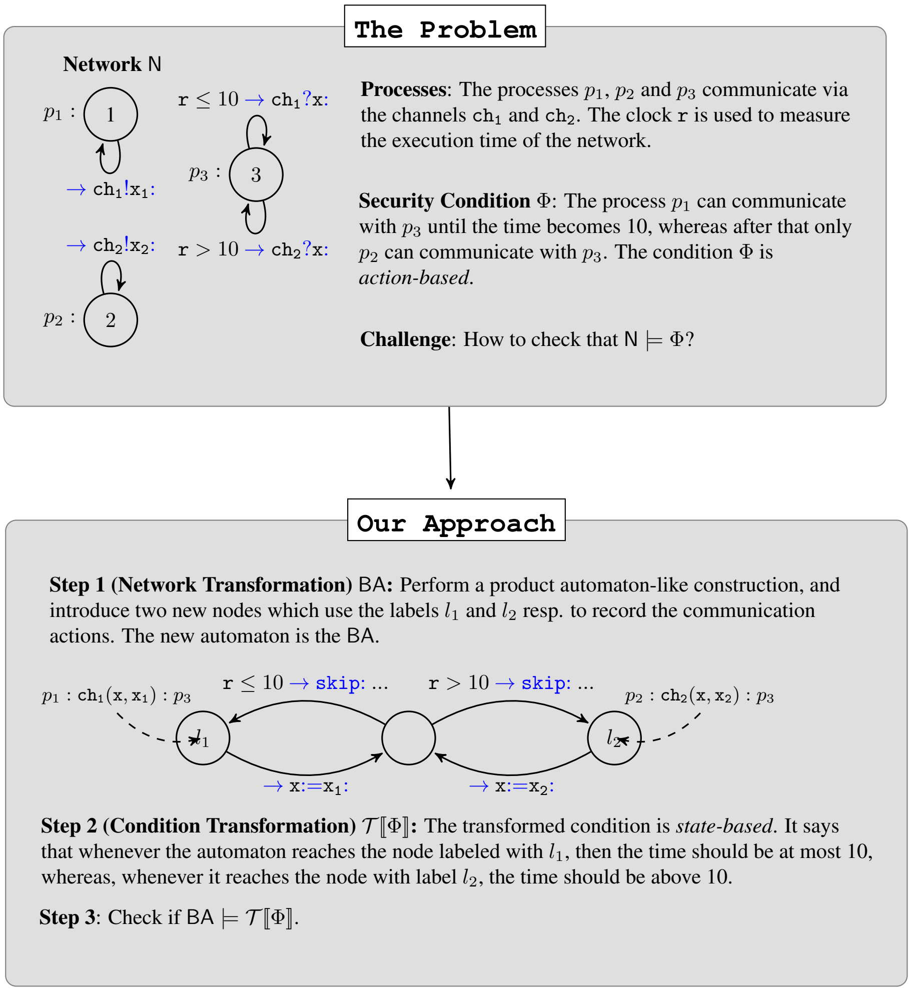
Finally, recall that the $\underline{\mathtt{sat}}(\cdots)$ operation expresses the satisfiability of the $\cdots$ formula, and it is used to detect implicit flows arising from the non-deterministic semantics of timed automata. Already the work of [NN19] considers an algorithm that over-approximates $\underline{\mathtt{sat}}(...)$, and has been implemented at `http://www.formalmethods.dk/if4fun/` for detecting flows in a guarded-command style language. Another direction could be rephrase the problem as an SMT problem (Satisfaction Modulo Theories) and use one of the available solvers such as Z3 [dMB08].

CHAPTER 5

Data- and Time- Dependent Policy-Based Access Control

In this chapter, we develop a logic which can be used to enforce data- and time- dependent policy based access control on systems of distributed real-time processes. In particular, we model systems of distributed real-time processes as *networks of timed automata*, and we define an *information flow* instrumented operational semantics for them. The semantics make use of labels called *behaviours*, in order to record the different explicit flows, arising whenever a process is trying to read, or write to a variable, or a channel of the network. Our logic is called **BTCTL** (behaviour timed computational tree logic), and it is based on those behaviours. Moreover, we illustrate how we can successfully express and enforce security policies for our gateway example presented in Example 2.1.

We then show that the model-checking problem $\mathsf{N} \models \Phi$ of a network N and a **BTCTL** formula Φ, can be reduced to the model-checking problem $\mathsf{BA} \models \mathcal{T}[\![\Phi]\!]$. Here, BA is a timed automaton, raising from a product automaton-like construction of the network N, while, $\mathcal{T}[\![\Phi]\!]$ is a formula produced by a transformation on Φ. The transformed formula $\mathcal{T}[\![\Phi]\!]$ results to a formula in a logic called **TCTL**$^{+}$, that is a variation of the TCTL [ACD93] (timed computational tree logic). Finally, we implement this reduction, and we show how the enforcement of policies can be fully automated using the model-checker UPPAAL. Our approach is depicted in Figure 5.1.


The Problem
Network N
p_1 : 1
$\rightarrow \mathtt{ch_1!x_1}$:
$\rightarrow \mathtt{ch_2!x_2}$:
p_2 : 2
$\mathtt{r} \leq 10 \rightarrow \mathtt{ch_1?x}$:
p_3 : 3
$\mathtt{r} > 10 \rightarrow \mathtt{ch_2?x}$:
Our Approach
$p_1 : \mathtt{ch_1(x, x_1)} : p_3$
$\mathtt{r} \leq 10 \rightarrow \mathtt{skip}$: ...
$\mathtt{r} > 10 \rightarrow \mathtt{skip}$: ...
$p_2 : \mathtt{ch_2(x, x_2)} : p_3$
l_1
l_2
$\rightarrow \mathtt{x{:=}x_1}$:
$\rightarrow \mathtt{x{:=}x_2}$:

Processes: The processes p_1, p_2 and p_3 communicate via the channels $\mathtt{ch_1}$ and $\mathtt{ch_2}$. The clock $\mathtt{r}$ is used to measure the execution time of the network.

Security Condition Φ: The process p_1 can communicate with p_3 until the time becomes 10, whereas after that only p_2 can communicate with p_3. The condition Φ is *action-based.*

Challenge: How to check that $\mathsf{N} \models \Phi$?

Step 1 (Network Transformation) BA**:** Perform a product automaton-like construction, and introduce two new nodes which use the labels l_1 and l_2 resp. to record the communication actions. The new automaton is the BA.

Step 2 (Condition Transformation) $\mathcal{T}[\![\Phi]\!]$**:** The transformed condition is *state-based.* It says that whenever the automaton reaches the node labeled with l_1, then the time should be at most 10, whereas, whenever it reaches the node with label l_2, the time should be above 10.

Step 3: Check if $\mathsf{BA} \models \mathcal{T}[\![\Phi]\!]$.

Figure 5.1: The idea of our approach in Chapter 5.

Chapter Organisation In Section 5.1, we give the model of networks of timed automata, and in Section 5.2 we give the semantics for them. Next, in Section 5.3, we present the syntax and the semantics of **BTCTL** , and in Section 5.4 we present the reduction of the model-checking problem from **BTCTL** to **TCTL**$^+$. In Section 5.5, we discuss an implementation of this reduction, and we show how we can use the model-checker UPPAAL to fully automate our enforcement of security policies. Finally, in Section 5.6 and Section 5.7, we give our related work and conclusions respectively.

5.1 Networks of Timed Automata

We start by giving the model of a *network of timed automata*, which can be seen as an abstraction of systems of distributed real-time processes. In particular, the processes in the network have isolated memories and the only way to communicate is through polyadic channels. In addition, communications may be constrained from the current time of the network and from the content of some of the variables.

Networks Formally, a *network of timed automata* N

$$p_1 : \mathsf{TA}_1 \mid\mid ... \mid\mid p_n : \mathsf{TA}_n \ (n \geq 1)$$

which sometimes, we simply call it network, is the parallel composition of n timed automata called *processes*, written as the family N=$(\mathsf{TA}_i)_{i \leq n}$.

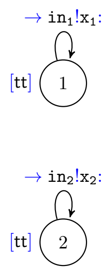
The processes are able to exchange information via synchronous message passing, using polyadic channels from the finite set **Chan**. Each of the processes in the network, is labelled with a unique identifier $p \in$ **P**=$\{p_1, ..., p_n\}$ and we write $\mathbf{Var}_p$ and $\mathbf{Clocks}_p$ for the data variables and clocks appearing in the process p. We also require that the sets of data variables and clocks for the processes are mutually disjoint ($\forall i \neq j : \mathbf{Var}_{p_i} \cap \mathbf{Var}_{p_j} = \emptyset \wedge \mathbf{Clocks}_{p_i} \cap \mathbf{Clocks}_{p_j} = \emptyset$) and we write **Var**=$\bigcup_i \mathbf{Var}_{p_i}$ and **Clocks**=$\bigcup_i \mathbf{Clocks}_{p_i}$ for the overall data variables and clocks appearing in the network. Finally, we also require that the sets of nodes of the processes are mutually disjoint (i.e $\forall i \neq j : \mathsf{Q}_i \cap \mathsf{Q}_j = \emptyset$).

Processes Each *process* is modelled as a *timed automaton* [AD94, AILS07] $\mathsf{TA} = (\mathsf{Q}, \mathsf{E}, \mathsf{I}, q_\circ)$ which is defined as in Section 3.1. However, the edges are now labelled with actions $g \rightarrow act$: $\boldsymbol{r}$, where the syntax of the action act is given by

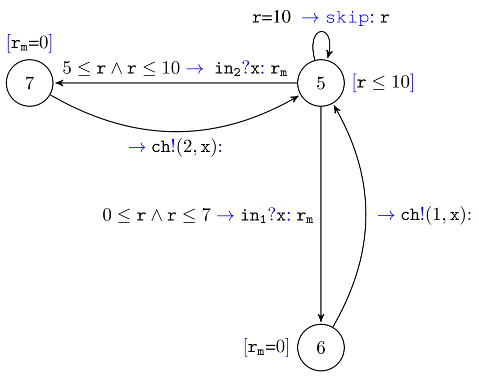
$$act ::= \boldsymbol{x} := \boldsymbol{a} \mid ch!\boldsymbol{a} \mid ch?\boldsymbol{x}$$

The assignment action $g \rightarrow \boldsymbol{x} := \boldsymbol{a}$: $\boldsymbol{r}$ is as in the previous chapters, while the action $g \rightarrow ch!\boldsymbol{a}$: $\boldsymbol{r}$ is used to communicate the data of the expressions in $\boldsymbol{a}$ using the channel $ch \in$ **Chan**, and the action $g \rightarrow ch?\boldsymbol{x}$: $\boldsymbol{r}$ is used to receive data and store it in the variables of the sequence $\boldsymbol{x}$.

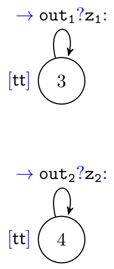
Recall that we assume that the sequences $\boldsymbol{x}$ and $\boldsymbol{a}$ of data variables and expressions, respectively, have the same length and that in addition $\boldsymbol{x}$ does not contain any repetitions.

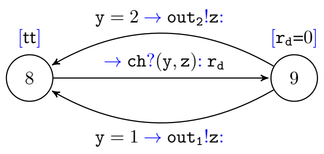


(a) The processes of the producers p_1 (top) and p_2 (bottom).


r=10 $\rightarrow$ skip: r
[r$_m$=0]
7
$5 \leq$ r $\wedge$ r $\leq 10 \rightarrow$ in$_2$?x: r$_m$
5
[r ≤ 10]
$\rightarrow$ ch!(2, x):
$0 \leq$ r $\wedge$ r $\leq 7 \rightarrow$ in$_1$?x: r$_m$
$\rightarrow$ ch!(1, x):
[r$_m$=0]
6

(b) The process of the multiplexer m.




y = 2 $\rightarrow$ out$_2$!z:
[tt]
[r$_d$=0]
$\rightarrow$ ch?(y, z): r$_d$
8
9
y = 1 $\rightarrow$ out$_1$!z:

(c) The process of the demultiplexer d.

(d) The processes of the consumers c_1 (top) and c_2 (bottom).

Figure 5.2: The network of the gateway Example 2.1.

Finally, we write $\boldsymbol{x}(i)$ (and also $\boldsymbol{a}(i)$) for the i-th element of the sequence $\boldsymbol{x}$ (and $\boldsymbol{a}$ respectively). Recall also that for special cases of the assignment action, we shall write $g \rightarrow$ `skip`: $\boldsymbol{r}$ when $\boldsymbol{x}$ (and hence $\boldsymbol{a}$) is empty; also for any kind of action we shall omit the guard g when it is equal to tt, and omit the clock resets when $\boldsymbol{r}$ is empty. If it is the case that all of the above take place together we omit the whole action.

EXAMPLE 5.1 *The network of our gateway from Example 2.1 is given in Figure 5.2.*

Processes and Channels *The network consists of six processes given from the set* $\boldsymbol{P} = \{p_1, p_2, m, d, c_1, c_2\}$. *Two producers p_1 and p_2 send their data via the channels* in$_1$ *and* in$_2$ *respectively. The multiplexer m collects the data from the producers using the channel* ch *and then forwards it to the demultiplexer d, who then distributes it to the consumers* c$_1$ *and* c$_2$ *via the channels* out$_1$ *and* out$_2$ *respectively.*

Clocks *We use two clocks* $\mathtt{r_m}$ *(in the multiplexer) and* $\mathtt{r_d}$ *(in the demultiplexer) to model instantaneous transitions, i.e in this model actions (assignments and channel communication) take no time. The multiplexer makes use of an additional clock* $\mathtt{r}$ *(in the multiplexer) in order to regulate the communication requests from the two producers* p_1 *and* p_2*. The clock* $\mathtt{r}$ *is reset every 10 time units, splitting in this way the overall execution of the automaton in intervals of 10 time units.*

Variables *The variables* $\mathtt{x_1}$ *and* $\mathtt{x_2}$ *carry the data the producers* p_1 *and* p_2 *wish to communicate respectively. Upon communication between one of the producers and the multiplexer, the multiplexer stores the data of the producers in the variable* $\mathtt{x}$*. Next, at the side of the demultiplexer the value of* $\mathtt{x}$ *is stored in the variable* $\mathtt{z}$ *and the variable* $\mathtt{y}$ *is used from the multiplexer to identify the origin of the data. Finally, the consumers* c_1 *and* c_2 *use the variables* $\mathtt{z_1}$ *and* $\mathtt{z_2}$ *respectively to store the value of* $\mathtt{z}$*.*

Transitions *In each period the multiplexer* m *reads data from the channel* $\mathtt{in_1}$ *only whenever the time* $t \in [0, 7]$*, expressed by the guard* $0 \leq \mathtt{r} \wedge \mathtt{r} \leq 7$*, while it reads data from the channel* $\mathtt{in_2}$ *only whenever* $t \in [5, 10]$*, expressed by the guard* $5 \leq \mathtt{r} \wedge \mathtt{r} \leq 10$*. Whenever* $t \in [5, 7]$ *the multiplexer chooses non-deterministically to read either from* $\mathtt{in_1}$ *or* $\mathtt{in_2}$*. Next, the multiplexer transfers the data together with a constant, using the dyadic channel* $\mathtt{ch}$ *to the demultiplexer* d*; the constant is used as a mark to indicate the source of the data. In case of* p_1 *being the source, then this constant is 1, otherwise it is 2. Finally, the demultiplexer delivers the data to the right consumer according to the constant, expressed by the guards* $\mathtt{y} = 1$ *and* $\mathtt{y} = 2$*. Finally, whenever a period of 10 time units has been completed (*$\mathtt{r} = 10$*) the multiplexer resets the clock* $\mathtt{r}$*.*

5.2 Information Flow Instrumented Semantics

In this section, we develop an information flow instrumented semantics for networks. The instrumentation makes use of labels which we call behaviours. The behaviours allow us to monitor the accesses on variables and channels performed by the processes, and they will be one of the main ingredients for the enforcement of our security policies in later sections.

5.2.1 Behaviours

The transitions of networks are labeled with behaviours. In particular, a *behaviour* records information relevant to the action that has occurred and also information about the processes which were involved in the action. Formally a behavior takes the form

$$\mathfrak{b} \in \mathcal{B}_{local} \cup \mathcal{B}_{com} \cup \{\epsilon\}$$

where

$$\mathcal{B}_{local} = \mathbf{P} \times \overrightarrow{\mathbf{Var}} \times \overrightarrow{\mathbf{Exp}}$$

are the behaviours that occur due to assignments and

$$\mathcal{B}_{com} = \mathbf{P} \times \mathbf{Chan} \times \overrightarrow{\mathbf{Var}} \times \overrightarrow{\mathbf{Exp}} \times \mathbf{P}$$

are the behaviours that occur due to the communication between two processes. We write $\overrightarrow{\mathbf{Var}}$ and $\overrightarrow{\mathbf{Exp}}$ for the sets of sequences with elements over the data variables and arithmetic expressions respectively.

For instance, the local behaviour $p : (\boldsymbol{x}, \boldsymbol{a})$ records that the process p has performed an assignment in which the sequence $\boldsymbol{a}$ is used to modify the variables of the sequence $\boldsymbol{x}$. As another example, consider the behaviour $p : ch(\boldsymbol{x}, \boldsymbol{a}) : p'$, which records that a communication between the processes p (the sender) and p' (the receiver) has occurred, using the channel ch, and the sequence $\boldsymbol{a}$ is the sequence of expressions whose values have been communicated and have been bound to the variables of the sequence $\boldsymbol{x}$.

Finally, in all of the behaviours, the sequences that are being used must have the same length, while for the delay action we will write the empty behaviour ϵ.

5.2.2 Operational Semantics

To specify the semantics of networks, let σ be a state mapping data variables to integers, let δ be a clock assignment mapping clocks to non-negative reals and let κ be a mapping from data variables to a set of processes which we call *writers*. The mapping κ is used to monitor the explicit flows that occur from the assignments and the communication between two processes in the network. We will explain the use of κ in more detail in a while.

We then have total semantic functions $[\![.]\!]$ for evaluating the expressions, boolean tests and guards. We evaluate expressions either with a state σ, or the mapping κ, where for the first case the evaluation returns a value, and in the second it returns the writers of the expression.

The configurations of a network $\mathsf{N} = (\mathsf{TA}_i)_{i \leq n}$ are of the form $\langle \boldsymbol{q}, \sigma, \delta, \kappa \rangle$, where $\boldsymbol{q} \in \mathsf{Q}_1 \times ... \times \mathsf{Q}_n$ is an n-tuple of nodes, we write $\boldsymbol{q}(i)$ for the i-th element of $\boldsymbol{q}$, and we have that $\bigwedge_{i=1}^{n} [\![\mathsf{I}_i(\boldsymbol{q}(i))]\!](\sigma, \delta)$. Finally, we write $\boldsymbol{q}[q'/q]$ to substitute the node q with the node q' in $\boldsymbol{q}$.

$$\langle \boldsymbol{q}_s, \sigma, \delta, \kappa \rangle \overset{p_j:(\boldsymbol{x},\boldsymbol{a})}{\Longrightarrow} \langle \boldsymbol{q}_t, \sigma', \delta', \kappa' \rangle \quad \text{if} \begin{cases} (q, g \rightarrow \boldsymbol{x} := \boldsymbol{a}\colon \boldsymbol{r}, q') \text{ is in } \mathsf{E}_j \\ [\![g]\!](\sigma,\delta) = \mathsf{tt} \\ \sigma' = \sigma[\boldsymbol{x} \mapsto [\![\boldsymbol{a}]\!]\sigma] \\ \delta' = \delta[\boldsymbol{r} \mapsto \boldsymbol{0}] \\ \kappa' = \kappa[\boldsymbol{x} \mapsto [\![\boldsymbol{a}]\!]\kappa] \\ \boldsymbol{q}_t = \boldsymbol{q}_s[q'/q] \\ \bigwedge_{i=1}^{n} [\![\mathsf{I}_i(\boldsymbol{q}_t(i))]\!](\sigma',\delta') = \mathsf{tt} \end{cases}$$

$$\langle \boldsymbol{q}_s, \sigma, \delta, \kappa \rangle \overset{p_h:ch(\boldsymbol{x},\boldsymbol{a}):p_l}{\Longrightarrow} \langle \boldsymbol{q}_t, \sigma', \delta', \kappa' \rangle \quad \text{if} \begin{cases} h \neq l \\ (q_1, g_1 \rightarrow ch!\boldsymbol{a}\colon \boldsymbol{r}_1, q_1') \text{ is in } \mathsf{E}_h \\ (q_2, g_2 \rightarrow ch?\boldsymbol{x}\colon \boldsymbol{r}_2, q_2') \text{ is in } \mathsf{E}_l \\ \sigma' = \sigma[\boldsymbol{x} \mapsto [\![\boldsymbol{a}]\!]\sigma] \\ \delta' = (\delta[\boldsymbol{r}_1 \mapsto \boldsymbol{0}])[\boldsymbol{r}_2 \mapsto \boldsymbol{0}] \\ \kappa' = \kappa[\boldsymbol{x} \mapsto [\![\boldsymbol{a}]\!]\kappa] \\ \boldsymbol{q}_t = \boldsymbol{q}_s[q_1'/q_1][q_2'/q_2] \\ \bigwedge_{i=1}^{n} [\![\mathsf{I}_i(\boldsymbol{q}_t(i))]\!](\sigma',\delta') = \mathsf{tt} \end{cases}$$

$$\langle \boldsymbol{q}, \sigma, \delta, \kappa \rangle \overset{\epsilon}{\Longrightarrow} \langle \boldsymbol{q}, \sigma, \delta', \kappa \rangle \quad \text{if} \begin{cases} \exists t > 0 : \delta' = \lambda r.\, \delta(r) + t, \\ \bigwedge_{i=1}^{n} [\![\mathsf{I}_i(\boldsymbol{q}(i))]\!](\sigma,\delta') = \mathsf{tt} \end{cases}$$

Table 5.1: Instrumented semantics of networks.

The transitions of a network take the form

$$\langle \boldsymbol{q}_s, \sigma, \delta, \kappa \rangle \overset{\mathfrak{b}}{\Longrightarrow} \langle \boldsymbol{q}_t, \sigma', \delta', \kappa' \rangle$$

and the initial configurations are of the form $\langle \boldsymbol{q}_\circ, \sigma, \lambda r.0, \kappa_\circ \rangle$ where $\boldsymbol{q}_\circ$ is the sequence, whose elements are the initial nodes of the timed automata in the network, and $\kappa_\circ$ maps each variable to the process that it belongs to (i.e $\kappa_\circ(x) = \{p\}$ iff $x \in \mathbf{Var}_p$). The transition relation is given in Table 5.1.

The rule for the assignment, ensures that the guard is satisfied in the starting configuration and updates the mappings σ, δ, κ and the location of the process p_j. Finally, it ensures that the invariant is satisfied in the resulting configuration. The behaviour $p_j : (\boldsymbol{x}, \boldsymbol{a})$ records that the process p_j is performing an assignment to the sequence $\boldsymbol{x}$ using the sequence $\boldsymbol{a}$, and the mapping κ' records the information flow that occurs due to this behaviour, by updating the writers of each variable $\boldsymbol{x}(i)$ with the writers of the expression $\boldsymbol{a}(i)$. Here, for a single expression a', we have that $[\![a']\!]\kappa = \bigcup_{y \in \mathsf{fv}(a')} \kappa(y)$, and recall that $\mathsf{fv}(a')$ is the set of free variables occurring in a'.

To understand the rule for the communication one could see it as an assignment of the form $\boldsymbol{x} := \boldsymbol{a}$, where $\boldsymbol{a}$ are the expressions which are used at the channel output action, and $\boldsymbol{x}$ are the variables that are used in the channel input action.

Finally, the delay rule only modifies the clock assignment with a delay t, ensuring that the invariant is satisfied in the resulting configuration. The mapping κ remains the same since the delay action produces the empty behaviour ϵ.

DEFINITION 5.1 (**Runs**) A run of a configuration γ is a (possibly infinite) sequence

$$\langle \boldsymbol{q}_0, \sigma_0, \delta_0, \kappa_0 \rangle \overset{\mathfrak{b}_1}{\Longrightarrow} \ldots \overset{\mathfrak{b}_n}{\Longrightarrow} \langle \boldsymbol{q}_n, \sigma_n, \delta_n, \kappa_n \rangle \overset{\mathfrak{b}_{n+1}}{\Longrightarrow} \ldots$$

where $\gamma = \langle \boldsymbol{q}_0, \sigma_0, \delta_0, \kappa_0 \rangle$ and we write $\mathsf{Runs}(\gamma)$ for the set of runs of γ.

EXAMPLE 5.2 *To understand better how the semantics work, return to Example 5.1 and consider the configuration $\langle \boldsymbol{q}, \sigma, \delta, \kappa \rangle$ where $\boldsymbol{q} = (1, 2, 5, 8, 3, 4)$, and let $\sigma = \lambda x.0$, $\delta = \lambda r.0$ and*

$$\kappa = [\mathtt{x}_1 \mapsto \{p_1\}, \mathtt{x}_2 \mapsto \{p_2\}, \mathtt{x} \mapsto \{m\}, \mathtt{y} \mapsto \{d\}, \mathtt{z} \mapsto \{d\}, \mathtt{z}_1 \mapsto \{c_1\}, \mathtt{z}_2 \mapsto \{c_2\}]$$

A possible transition of the network is the following

$$\langle \boldsymbol{q}, \sigma, \delta, \kappa \rangle \overset{p_1:\mathtt{in}_1(\mathtt{x},\mathtt{x}_1):m}{\Longrightarrow} \langle \boldsymbol{q}[6/5], \sigma[\mathtt{x} \mapsto \sigma(\mathtt{x}_1)], \delta[\mathtt{r}_\mathtt{m} \mapsto 0], \kappa[\mathtt{x} \mapsto \{p_1\}] \rangle$$

which corresponds to the communication between the producer p_1 and the multiplexer m. The resulting mapping $\kappa[\mathtt{x} \mapsto \{p_1\}]$ records that p_1 has written its value into the variable $\mathtt{x}$, since there is an explicit flow from the variable $\mathtt{x}_1$ to the variable $\mathtt{x}$ and $\mathtt{x}_1$ has previously been written by p_1.

5.3 Access Control in BTCTL

In this section, we present our behaviour based logic **BTCTL** which serves to specify data- and time-dependent security policies for access control, based on the behaviours of the network. In particular, each of the variables of the network will be associated with a security policy that expresses under which content of the network's variables and clocks one may influence the value of this particular variable. The goal of the policies here is to ensure data integrity. We will then specify a security condition as a formula, in order to specify when a network satisfies the security policies. However, recall that access control protects only from explicit flows of information and not from implicit ones.

5.3.1 The Syntax

The syntax of the **BTCTL** formulas ϕ is given by

$$\phi ::= g \mid set_1 \;\mathsf{rel}\; set_2 \mid \forall\Box_{\mathfrak{b}}(\phi_1, \phi_2) \mid \phi_1 \wedge \phi_2 \mid \neg\phi$$

where

$$set ::= \underline{a} \mid W$$

We have basic formulas which can be either a guard g, or relations between two sets of writers, set_1 rel set_2, where rel $= \{\subseteq, \supseteq\}$. The underlined set expression $\underline{a}$ denotes the set of writers of the expression a and $W \in \mathbb{P}(\mathbf{P})$ is a set of writers. For example, let x be a variable and p_1, p_2 two processes. One example policy could be $\underline{x} \subseteq \{p_1\}$, which specifies that only p_1 can influence the value of x and not p_2.

We use the box operator $\forall\Box_{\mathfrak{b}}(\phi_1, \phi_2)$ to speak about pre- and post-conditions whenever the non-empty behaviour $\mathfrak{b} \neq \epsilon$ happens. Informally speaking, a configuration γ will satisfy the $\forall\Box_{\mathfrak{b}}(\phi_1, \phi_2)$ formula whenever for all of the network runs starting at γ, if a transition labelled with the behaviour $\mathfrak{b}$ occurs, then ϕ_1 should hold at the configuration before the transition and ϕ_2 at the configuration after it. As we will see shortly, the box operator will be the key formula to enforce access control policies. The $\neg\phi$ and $\phi_1 \wedge \phi_2$ cases are the usual ones. Finally, we sometimes write $\phi_1 \Rightarrow \phi_2$ for $\neg(\phi_1 \wedge \neg\phi_2)$.

EXAMPLE 5.3 *Let's now go back to our gateway network given in Example 5.1.*

Security Policies *Each of the variables has a policy which specifies the maximum set of permitted writers of the variable. We are interested in the policies of the variables of the multiplexer, the demultiplexer and the two consumers. Therefore we have the following policies*

$$\begin{array}{ll}
P_{\mathtt{x}} = & (0 \leq \mathtt{r} \wedge \mathtt{r} < 5 \Rightarrow \underline{\mathtt{x}} \subseteq \{p_1\})\wedge \\
& (5 \leq \mathtt{r} \wedge \mathtt{r} \leq 7 \Rightarrow \underline{\mathtt{x}} \subseteq \{p_1, p_2\})\wedge \\
& (7 < \mathtt{r} \wedge \mathtt{r} \leq 10 \Rightarrow \underline{\mathtt{x}} \subseteq \{p_2\}) \\
P_{\mathtt{y}} = & \underline{\mathtt{y}} \subseteq \{m\} \\
P_{\mathtt{z}} = & (0 \leq \mathtt{r} \wedge \mathtt{r} \leq 7 \wedge \mathtt{y} = 1 \Rightarrow \underline{\mathtt{z}} \subseteq \{p_1\})\wedge \\
& (5 \leq \mathtt{r} \wedge \mathtt{r} \leq 10 \wedge \mathtt{y} = 2 \Rightarrow \underline{\mathtt{z}} \subseteq \{p_2\}) \\
P_{\mathtt{z}_1} = & \underline{\mathtt{z}_1} \subseteq \{p_1\} \\
P_{\mathtt{z}_2} = & \underline{\mathtt{z}_2} \subseteq \{p_2\}
\end{array}$$

The first line of the policy for the variable $\mathtt{x}$*, expresses that whenever the clock* $\mathtt{r} \in [0, 5)$*, only the process* p_1 *is allowed to write data to* $\mathtt{x}$*, while both* p_1 *and* p_2 *may write to* $\mathtt{x}$ *if* $\mathtt{r} \in [5, 7]$*. Finally, the last line expresses that whenever* $\mathtt{r} \in (7, 10]$*, then only*

p_2 can write to $\texttt{x}$. Next, looking at the policy for the variable $\texttt{y}$, a write action to it is allowed only by the multiplexer.

The policy of the variable $\texttt{z}$ is both a time- and data-dependent policy. In particular, it specifies that whenever $\texttt{r} \in [0,7]$, and $\texttt{y}$ contains the value 1, then the value of $\texttt{z}$ can be influenced by p_1, whereas whenever $\texttt{r} \in [5,10]$ and $\texttt{y}$ is 2, p_2 may influence the value of $\texttt{z}$.

Finally, the policies for $\texttt{z}_1$ and $\texttt{z}_2$, express that only p_1 can influence $\texttt{z}_1$, and only p_2 can influence $\texttt{z}_2$ respectively.

Security Condition *Next, we need to specify our semantic security condition that defines when the network satisfies the given security policies. In particular, we need to ensure that the policies are enforced, whenever an action that writes data to the variables of interest occurs. To this end, we define the following formulas*

$$\begin{aligned}
&\Phi_{\texttt{x}} = \forall\Box_{p_1:\texttt{in}_1(\texttt{x},\texttt{x}_1):m}(\mathsf{tt}, P_{\texttt{x}}) \wedge \forall\Box_{p_2:\texttt{in}_2(\texttt{x},\texttt{x}_2):m}(\mathsf{tt}, P_{\texttt{x}})\\
&\Phi_{\texttt{y},\texttt{z}} = \forall\Box_{m:\texttt{ch}((\texttt{y},\texttt{z}),(1,\texttt{x})):d}(\mathsf{tt}, P_{\texttt{y}} \wedge P_{\texttt{z}}) \wedge \forall\Box_{m:\texttt{ch}((\texttt{y},\texttt{z}),(2,\texttt{x})):d}(\mathsf{tt}, P_{\texttt{y}} \wedge P_{\texttt{z}})\\
&\Phi_{\texttt{z}_1} = \forall\Box_{d:\texttt{out}_1(\texttt{z}_1,\texttt{z}):c_1}(\mathsf{tt}, P_{\texttt{z}_1})\\
&\Phi_{\texttt{z}_2} = \forall\Box_{d:\texttt{out}_2(\texttt{z}_2,\texttt{z}):c_2}(\mathsf{tt}, P_{\texttt{z}_2})
\end{aligned}$$

Each of the formulas expresses that whenever someone is writing to the variable (or variables) appearing as a subscript, then the policy of the variable (or variables) is imposed as a post condition. The variable $\texttt{x}$ is accessed (someone is writing data to $\texttt{x}$) whenever p_1 and p_2 communicates with the multiplexer, and thus we have to impose the policy of the variable $\texttt{x}$ for both of those actions. The variables $\texttt{y}$ and $\texttt{z}$ are being accessed whenever the multiplexer communicates with the demultiplexer, and that happens with two communication actions. Similarly, we define the formulas for the variables $\texttt{z}_1$ and $\texttt{z}_2$.

5.3.2 Semantics of the BTCTL Formulas

The formal semantics for when a configuration γ satisfies a **BTCTL** formula ϕ is given below

$$\begin{array}{lll}
\gamma \models g & \text{iff} & \gamma = \langle \boldsymbol{q}, \sigma, \delta, \kappa \rangle \Rightarrow [\![g]\!](\sigma, \delta) \\
\gamma \models set_1 \,\mathsf{rel}\, set_2 & \text{iff} & \gamma = \langle \boldsymbol{q}, \sigma, \delta, \kappa \rangle \Rightarrow [\![set_1]\!]\kappa \,\mathsf{rel}\, [\![set_2]\!]\kappa \\
\gamma \models \forall\Box_{\mathfrak{b}}(\phi_1, \phi_2) & \text{iff} & \forall \gamma_0 \overset{\mathfrak{b}_1}{\Longrightarrow} \gamma_1 \overset{\mathfrak{b}_2}{\Longrightarrow} .. \in \mathsf{Runs}(\gamma) : \\
 & & \forall i \geq 1 : \mathfrak{b}_i = \mathfrak{b} \Rightarrow \gamma_{i-1} \models \phi_1 \text{ and } \gamma_i \models \phi_2 \\
\gamma \models \phi_1 \wedge \phi_2 & \text{iff} & \gamma \models \phi_1 \text{ and } \gamma \models \phi_2 \\
\gamma \models \neg\phi & \text{iff} & \gamma \not\models \phi
\end{array}$$

A guard g is then satisfied by a configuration γ whenever g holds in γ. For the case of the set relation rel, γ satisfies it whenever $set_1 \,\mathsf{rel}\, set_2$ evaluates to true, and we do that check by lifting the definition of $[\![.]\!]\kappa$ to set of expressions, by defining $[\![\underline{a}]\!]\kappa = [\![a]\!]\kappa$, and $[\![W]\!]\kappa = W$. A configuration γ satisfies the box formula $\forall\Box_{\mathfrak{b}}(\phi_1, \phi_2)$ whenever for all the execution paths that start from γ, if a behaviour $\mathfrak{b}'$ occurs and $\mathfrak{b}'$ is syntactically equal to $\mathfrak{b}$, then the pre-condition ϕ_1 has to hold at the configuration before the behaviour and the post-condition ϕ_2 at the configuration after it. The rest of the cases are the usual ones.

Example 5.4 *We now give an example to illustrate how the box operator semantics work. For this, consider the prefix*

$$pr = \gamma_0 \overset{p_1:\mathtt{in_1(x,x_1)}:m}{\Longrightarrow} \gamma_1 \overset{m:\mathtt{ch((y,z),(1,x))}:d}{\Longrightarrow} \gamma_2 \overset{d:\mathtt{out_1(z_1,z)}:c_1}{\Longrightarrow} \gamma_3$$

of an execution trace of the network given in Example 5.1, where for the initial configuration $\gamma_0 = \langle \boldsymbol{q}_0, \sigma_0, \delta_0, \kappa_0 \rangle$, we have that $\boldsymbol{q}_0 = (1, 2, 5, 8, 3, 4)$, σ_0 is arbitrary, $\delta_0 = \lambda r.0$ and

$$\kappa_0 = [\mathtt{x_1} \mapsto \{p_1\}, \mathtt{x_2} \mapsto \{p_2\}, \mathtt{x} \mapsto \{m\}, \mathtt{y} \mapsto \{d\}, \mathtt{z} \mapsto \{d\}, \mathtt{z_1} \mapsto \{c_1\}, \mathtt{z_2} \mapsto \{c_2\}]$$

In addition, for the rest of the configurations we have that $\gamma_1 = \langle \boldsymbol{q}_1, \sigma_1, \delta_1, \kappa_1 \rangle$, $\gamma_2 = \langle \boldsymbol{q}_2, \sigma_2, \delta_2, \kappa_2 \rangle$ and $\gamma_3 = \langle \boldsymbol{q}_3, \sigma_3, \delta_3, \kappa_3 \rangle$ where

$\boldsymbol{q}_1 = \boldsymbol{q}_0[6/5]$, $\sigma_1 = \sigma_0[\mathtt{x} \mapsto \sigma_0(\mathtt{x_1})]$, $\delta_1 = \delta_0[\mathtt{r_m} \mapsto 0]$, $\kappa_1 = \kappa_0[\mathtt{x} \mapsto \{p_1\}]$

$\boldsymbol{q}_2 = \boldsymbol{q}_1[5/6][9/8]$, $\sigma_2 = \sigma_1[\mathtt{y} \mapsto 1, \mathtt{z} \mapsto \sigma_1(\mathtt{x})]$, $\delta_2 = \delta_1[\mathtt{r_d} \mapsto 0]$,
$\kappa_2 = \kappa_1[\mathtt{y} \mapsto \emptyset, \mathtt{z} \mapsto \{p_1\}]$

$\boldsymbol{q}_3 = \boldsymbol{q}_1[8/9]$, $\sigma_3 = \sigma_2[\mathtt{z_1} \mapsto \sigma_2(\mathtt{z})]$, $\delta_3 = \delta_2$, $\kappa_3 = \kappa_2[\mathtt{z_1} \mapsto \{p_1\}]$.

Next, consider the formulas $\Phi_{\mathtt{x}}$, $\Phi_{\mathtt{y,z}}$, $\Phi_{\mathtt{z_1}}$ given in Example 5.3, and we will now do the appropriate checks for those formulas on the prefix pr.

The formula $\Phi_{\mathtt{x}}$ is the conjunction of two box operators, where for the first one because of the behaviour $p_1 : \mathtt{in_1(x,x_1)} : m$ of the transition $\gamma_0 \overset{p_1:\mathtt{in_1(x,x_1)}:m}{\Longrightarrow} \gamma_1$, we have to check that $\gamma_1 \models P_{\mathtt{x}}$. Recall that

$$\begin{aligned} P_{\mathtt{x}} = \quad & (0 \leq \mathtt{r} \land \mathtt{r} < 5 \Rightarrow \underline{\mathtt{x}} \subseteq \{p_1\}) \land \\ & (5 \leq \mathtt{r} \land \mathtt{r} \leq 7 \Rightarrow \underline{\mathtt{x}} \subseteq \{p_1, p_2\}) \land \\ & (7 < \mathtt{r} \land \mathtt{r} \leq 10 \Rightarrow \underline{\mathtt{x}} \subseteq \{p_2\}) \end{aligned}$$

This check evaluates to true, since γ_1 satisfies only the guard of the first line of the policy and $\kappa_1(\mathtt{x}) = \{p_1\}$. For the transition $\gamma_1 \overset{m:\mathtt{ch}((\mathtt{y},\mathtt{z}),(1,\mathtt{x})):d}{\Longrightarrow} \gamma_2$, because of the formula $\Phi_{\mathtt{y},\mathtt{z}}$ we have to check that $\gamma_2 \models P_{\mathtt{y}} \land P_{\mathtt{z}}$, and recall that

$$\begin{aligned} P_{\mathtt{y}} = \quad & \underline{\mathtt{y}} \subseteq \{m\} \\ P_{\mathtt{z}} = \quad & (0 \leq \mathtt{r} \land \mathtt{r} \leq 7 \land \mathtt{y} = 1 \Rightarrow \underline{\mathtt{z}} \subseteq \{p_1\}) \land \\ & (5 \leq \mathtt{r} \land \mathtt{r} \leq 10 \land \mathtt{y} = 2 \Rightarrow \underline{\mathtt{z}} \subseteq \{p_2\}) \end{aligned}$$

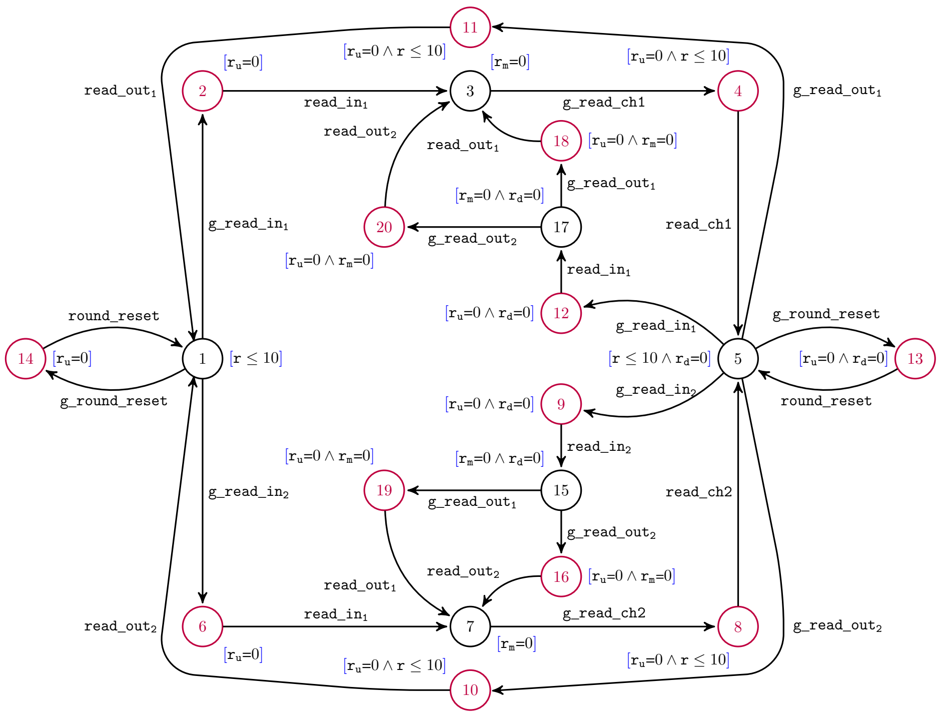
This check evaluates to true, since $\kappa_2(\mathtt{y}) = \emptyset$, and thus $\gamma_2 \models P_{\mathtt{y}}$, while for the policy $P_{\mathtt{z}}$ we have that γ_2 satisfies only the condition at the first line of the policy, and $\kappa_2(\mathtt{z}) = \{p_1\}$, which give us that $\gamma_2 \models P_{\mathtt{z}}$ as required.

Finally, for the last transition $\gamma_2 \overset{d:\mathtt{out}_1(\mathtt{z}_1,\mathtt{z}):c_1}{\Longrightarrow} \gamma_3$, because of the $\Phi_{\mathtt{z}_1}$ formula, we have to check that $\gamma_3 \models P_{\mathtt{z}}$. This check evaluates to true, since $\kappa_3(\mathtt{z}_1) = \{p_1\}$ and $P_{\mathtt{z}_1} = \underline{\mathtt{z}_1} \subseteq \{p_1\}$.

5.4 Reduction of BTCTL to TCTL$^+$

In this section, we perform a transformation of the original network, and the security conditions which are expressed as **BTCTL** formulas. The transformation results in a timed automaton, whose nodes carry extra information with regards to the actions performed in the original network. We call this automaton a behaviour automaton. The transformed security conditions are formulas in a new logic which we call **TCTL$^+$**. **TCTL$^+$** is a logic based on TCTL (timed computational tree logic) [ACD93].

In the next section, we will show how a fragment of **TCTL$^+$** can be handled by the model checker UPPAAL [UPP]. This will allows to fully automate our static enforcement of access control policies. Both the transformation of the network, and the security conditions are inspired by the work done in [JW02], where the action-based logic ATCTL (action-TCTL) is being reduced to TCTL [ACD93].



Genuine Vertices

v	$\mathsf{L}(v)$
1	(1,2,5,8,3,4)
3	(1,2,6,8,3,4)
5	(1,2,5,9,3,4)
7	(1,2,7,8,3,4)
15	(1,2,7,9,3,4)
17	(1,2,6,9,3,4)

Auxiliary Vertices

v	$\mathsf{L}(v)$	v	$\mathsf{L}(v)$
2	$p_1 : \mathtt{in}_1(\mathtt{x}, \mathtt{x}_1) : m$	10	$d : \mathtt{out}_2(\mathtt{z}_2, \mathtt{z}) : c_2$
4	$m : \mathtt{ch}((\mathtt{y}, \mathtt{z}), (1, \mathtt{x})) : d$	11	$d : \mathtt{out}_1(\mathtt{z}_1, \mathtt{z}) : c_1$
6	$p_2 : \mathtt{in}_2(\mathtt{x}, \mathtt{x}_2) : m$	12	$p_1 : \mathtt{in}_1(\mathtt{x}, \mathtt{x}_1) : m$
8	$m : \mathtt{ch}((\mathtt{y}, \mathtt{z}), (2, \mathtt{x})) : d$	13	$m : (\epsilon, \epsilon)$
9	$p_2 : \mathtt{in}_2(\mathtt{x}, \mathtt{x}_2) : m$	14	$m : (\epsilon, \epsilon)$
16	$d : \mathtt{out}_2(\mathtt{z}_2, \mathtt{z}) : c_2$	18	$d : \mathtt{out}_1(\mathtt{z}_1, \mathtt{z}) : c_1$
19	$d : \mathtt{out}_1(\mathtt{z}_1, \mathtt{z}) : c_1$	20	$d : \mathtt{out}_2(\mathtt{z}_2, \mathtt{z}) : c_2$

Actions

$\mathtt{read\_in}_1$	$\rightarrow \mathtt{x}{:=}\mathtt{x}_1{:}\ \mathtt{r}_\mathtt{m}$	$\mathtt{g\_read\_in}_1$	$0 \leq \mathtt{r} \leq 7 \rightarrow \mathtt{skip}{:}\ \mathtt{r}_\mathtt{u}$
$\mathtt{read\_in}_2$	$\rightarrow \mathtt{x}{:=}\mathtt{x}_2{:}\ \mathtt{r}_\mathtt{m}$	$\mathtt{g\_read\_in}_2$	$5 \leq \mathtt{r} \leq 10 \rightarrow \mathtt{skip}{:}\ \mathtt{r}_\mathtt{u}$
$\mathtt{read\_ch1}$	$\rightarrow (\mathtt{y}, \mathtt{z}){:=}(1, \mathtt{x}){:}\ \mathtt{r}_\mathtt{d}$	$\mathtt{g\_read\_ch1}$	$\rightarrow \mathtt{skip}{:}\ \mathtt{r}_\mathtt{u}$
$\mathtt{read\_ch2}$	$\rightarrow (\mathtt{y}, \mathtt{z}){:=}(2, \mathtt{x}){:}\ \mathtt{r}_\mathtt{d}$	$\mathtt{g\_read\_ch2}$	$\rightarrow \mathtt{skip}{:}\ \mathtt{r}_\mathtt{u}$
$\mathtt{read\_out}_1$	$\rightarrow \mathtt{z}_1{:=}\mathtt{z}{:}$	$\mathtt{g\_read\_out}_1$	$\mathtt{y} = 1 \rightarrow \mathtt{skip}{:}\ \mathtt{r}_\mathtt{u}$
$\mathtt{read\_out}_2$	$\rightarrow \mathtt{z}_2{:=}\mathtt{z}{:}$	$\mathtt{g\_read\_out}_2$	$\mathtt{y} = 2 \rightarrow \mathtt{skip}{:}\ \mathtt{r}_\mathtt{u}$
$\mathtt{round\_reset}$	$\rightarrow \mathtt{skip}{:}\ \mathtt{r}$	$\mathtt{g\_round\_reset}$	$\mathtt{r} = 10 \rightarrow \mathtt{skip}{:}\ \mathtt{r}_\mathtt{u}$

Figure 5.3: The behaviour automaton of the gateway's network, its vertices together with their labels, and the abbreviations for the actions used. The auxiliary vertices are colored with purple.

```
Step 1.  let $Q_{gen} = \emptyset$; let $Q_{aux} = \emptyset$;
         for all $\boldsymbol{q}$: create fresh $v$; let $\mathsf{L}(v) = \boldsymbol{q}$; let $\mathsf{I}(v) = \bigwedge_{i=1}^{n} \mathsf{I}_i(\boldsymbol{q}(i))$; insert $v$ in $Q_{gen}$
Step 2.  for all $(q_i, g \to \boldsymbol{x} := \boldsymbol{a}\colon \boldsymbol{r}, q_i') \in E_i$:

           for all $v_s \in Q_{gen}, v_t \in Q_{gen}$ such that $\begin{cases} \mathsf{L}(v_s)(i) = q_i \\ \mathsf{L}(v_t)(i) = q_i' \\ \forall j : j \neq i : \mathsf{L}(v_s)(j) = \mathsf{L}(v_t)(j) \end{cases}$ :

             create fresh $v$;
             insert $(v_s, g \to \mathtt{skip}\colon r_u, v)$ in $\mathsf{E}$; insert $(v, \boldsymbol{x} := \boldsymbol{a}\colon \boldsymbol{r}, v_t)$ in $\mathsf{E}$ ;
             let $\mathsf{L}(v) = p_i : (\boldsymbol{x}, \boldsymbol{a})$; let $\mathsf{I}(v) = (r_u = 0) \wedge \mathsf{I}(v_t)[\boldsymbol{a}/\boldsymbol{x}][\boldsymbol{0}/\boldsymbol{r}]$; insert $v$ in $Q_{aux}$
Step 3.  for all $(q_i, g_1 \to ch!\boldsymbol{a}\colon \boldsymbol{r}_1, q_i') \in E_i$ and $(q_j, g_2 \to ch?\boldsymbol{x}\colon \boldsymbol{r}_2, q_j') \in E_j$ such that $i \neq j$:

           for all $v_s \in Q_{gen}, v_t \in Q_{gen}$ such that $\begin{cases} \mathsf{L}(v_s)(i) = q_i \wedge \mathsf{L}(v_s)(j) = q_j \\ \mathsf{L}(v_t)(i) = q_i' \wedge \mathsf{L}(v_t)(j) = q_j' \\ \forall l : l \neq i \wedge l \neq j : \mathsf{L}(v_s)(l) = \mathsf{L}(v_t)(l) \end{cases}$ :

             create fresh $v$; let $g = g_1 \wedge g_2$; let $\boldsymbol{r} = \boldsymbol{r}_1\boldsymbol{r}_2$;
             insert $(v_s, g \to \mathtt{skip}\colon r_u, v)$ in $\mathsf{E}$; insert $(v, \boldsymbol{x} := \boldsymbol{a}\colon \boldsymbol{r}, v_t)$ in $\mathsf{E}$;
             let $\mathsf{L}(v) = p_i : ch(\boldsymbol{x}, \boldsymbol{a}) : p_j$; let $\mathsf{I}(v) = (r_u = 0) \wedge \mathsf{I}(v_t)[\boldsymbol{a}/\boldsymbol{x}][\boldsymbol{0}/\boldsymbol{r}]$; insert $v$ in $Q_{aux}$
Step 4.  let $\mathsf{Q} = Q_{gen} \cup Q_{aux}$
```

Table 5.2: The algorithm for constructing the edges E, the invariant function I, the vertices Q and the labelling function L of the behaviour automaton BA.

5.4.1 Behaviour Automata

A network $\mathsf{N} = (\mathsf{TA}_i)_{i \leq n}$ yields a *behaviour automaton* $\mathsf{BA} = (\mathsf{Q}, \mathsf{E}, \mathsf{I}, \mathsf{L}, v_\circ)$, which is a kind of timed automaton in that it is the product automaton of the network, extended to contain *auxiliary vertices* that represent the actions of the network, and a labelling function L that assigns to each vertex a *property*.

A *property* is either a behaviour or a location vector of the original network N. Auxiliary vertices will be labeled with the behaviour that corresponds to the particular action of the vertex, while *genuine vertices* which represent locations of the network N are labeled with a location vector. The initial vertex $v_\circ$ will be labeled with the initial location vector of the network $\boldsymbol{q}_\circ$.

The behaviour automaton BA has the same set of variables as the network N, while for the clock variables it has an extra clock r_u, which is used to model instantaneous transitions. Similarly to the timed automata, E is a finite set of edges, the mapping I imposes an invariant on each vertex, and Q is the finite set of vertices.

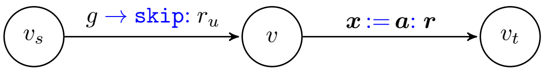


Figure 5.4: Edge construction of the behaviour automaton BA.

The algorithm for constructing the edges E, the labelling functions I and L, and the set of vertices $\mathsf{Q} = Q_{gen} \cup Q_{aux}$ (and notice that $Q_{gen} \cap Q_{aux} = \emptyset$) where Q_{gen} and Q_{aux} contain the genuine and auxiliary vertices respectively, is given in Table 5.2.

In the first step, we create the genuine vertices and we label them with the invariant of the location vector that they represent. Each of those vertices is inserted in Q_{gen}, which will be used in the next steps to create the auxiliary vertices.

In step 2, we create the auxiliary vertices, and the edges that correspond to the assignment actions of the network. For each process p_i, we start looking at all of its assignment edges $(q_i, g \to \boldsymbol{x} := \boldsymbol{a}\colon \boldsymbol{r}, q_i') \in E_i$. For each one of those edges and for all the vertices $v_s \in Q_{gen}$ and $v_t \in Q_{gen}$, where the label of v_s, $\mathsf{L}(v_s)$ corresponds to a vector location where this assignment could have been performed, and would have moved the network to the location $\mathsf{L}(v_t)$, we create the edges $(v_s, g \to \texttt{skip}\colon r_u, v)$ and $(v, \boldsymbol{x} := \boldsymbol{a}\colon \boldsymbol{r}, v_t)$. Notice here that v is a fresh auxiliary vertex, whereas in the construction of the product automaton one would have constructed only the edge $(v_s, g \to \boldsymbol{x} := \boldsymbol{a}\colon \boldsymbol{r}, v_t)$.

The auxiliary vertex v is labelled with the assignment behaviour $p_i : (\boldsymbol{x}, \boldsymbol{a})$ and its invariant is being set to $(r_u = 0) \wedge \mathsf{I}(v_t)[\boldsymbol{a}/\boldsymbol{x}][\boldsymbol{0}/\boldsymbol{r}]$, to first ensure that the action of the edge leaving v will be performed instantaneous, and secondly that we can not get stuck at an auxiliary vertex. Figure 5.4 illustrates the construction. Finally, note that each auxiliary vertex v has *exactly one predecessor* and *exactly one successor*.

Similarly to step 2, in step 3 we construct the auxiliary vertices for the communication actions of the network, and finally, in step 4, we define the set Q.

EXAMPLE 5.5 *The behaviour automaton of the our gateway network from Example 5.1 is given in Figure 5.3. It has 20 vertices, with vertex 1 being its initial vertex and 27 edges.*

5.4.2 Trace Equivalence

We will now show that the runs of the original network N can be simulated by the behaviour automaton BA and vice versa.

Particularly, each transition in the network N is equivalent to a single step transition (in

$$\langle v_s, \sigma, \delta, \kappa\rangle \longrightarrow \langle v_t, \sigma', \delta', \kappa'\rangle \quad \text{if} \begin{cases} (v_s, g \to \boldsymbol{x} := \boldsymbol{a} \colon \boldsymbol{r}, v_t) \text{ is in } \mathsf{E} \\ [\![g]\!](\sigma,\delta) = \mathsf{tt} \\ \sigma' = \sigma[\boldsymbol{x} \mapsto [\![\boldsymbol{a}]\!]\sigma] \\ \delta' = \delta[\boldsymbol{r} \mapsto \mathbf{0}] \\ \kappa' = \kappa[\boldsymbol{x} \mapsto [\![\boldsymbol{a}]\!]\kappa] \\ [\![\mathsf{I}(v_t)]\!](\sigma',\delta') = \mathsf{tt} \end{cases}$$

$$\langle v, \sigma, \delta, \kappa\rangle \longrightarrow \langle v, \sigma, \delta', \kappa\rangle \quad \text{if} \begin{cases} \exists t > 0 : \delta' = \lambda r.\, \delta(r) + t, \\ [\![\mathsf{I}(v)]\!](\sigma,\delta') = \mathsf{tt} \end{cases}$$

Table 5.3: Semantics for Behaviour Automata.

the case of a delay) or a two-step transition (in the case of an action) in its behaviour automaton BA. To overcome the vagueness of this explanation we will define a relation between runs of the network N, and the runs of the behaviour automaton BA.

To this end, we start by giving the operational semantics of behaviour automata in Table 5.3. The semantics is similar to the semantics of the timed automata networks, however now, the transitions are not labelled with behaviours, and there is no communication. Also for a configuration γ' of the behaviour automaton BA we write $\mathsf{Runs}_{\mathsf{BA}}(\gamma')$ for the set of runs it produces. Moreover, we now write $\mathsf{Runs}_{\mathsf{N}}(\gamma)$ for the runs of a configuration γ in the network N.

Now let γ, and γ' be two configurations of a network N, and its behaviour automaton BA respectively. We define the relation $\cong$: $\mathbf{Config}_{\mathsf{N}} \times \mathbf{Config}_{\mathsf{BA}} \to \{\mathsf{tt}, \mathsf{ff}\}$ to be

$$\langle \boldsymbol{q}, \sigma, \delta, \kappa\rangle \cong \langle v, \sigma', \delta', \kappa'\rangle \quad \text{iff} \quad \begin{array}{l} \boldsymbol{q} = \mathsf{L}(v) \\ \sigma = \sigma' \\ \forall r \in \mathbf{Clocks} : \delta(r) = \delta'(r) \\ \kappa = \kappa' \end{array}$$

where recall that **Clocks** is the set of the clocks appearing in the network N, and thus the clock r_u of the behaviour automaton BA is *not* included in it. It is straightforward (from the first condition) by the definition of $\cong$ that configurations of the network N can only be related with configurations that correspond to genuine vertices in the behaviour automaton BA.

Next, for the behaviour automata BA, we define a *macro transition* t' to be a single step delay transition $\gamma'_s \longrightarrow \gamma'_t$ or a two-step transition $\gamma'_s \longrightarrow \gamma_{aux} \longrightarrow \gamma'_t$, where γ_{aux} is an *auxiliary configuration* (a configuration that corresponds to an auxiliary vertex) and γ'_s and γ'_t are *genuine configurations* (configurations that correspond to genuine vertices). We then lift the definition of $\cong$ to single step transitions of the network N and macro transitions of the behaviour automaton BA as

$$(\gamma_s \xRightarrow{\epsilon} \gamma_t) \cong (\gamma'_s \longrightarrow \gamma'_t) \text{ iff } \begin{cases} \gamma_s \cong \gamma'_s \\ \gamma_t \cong \gamma'_t \end{cases}$$

$$(\gamma_s \xRightarrow{\mathfrak{b}} \gamma_t) \cong (\gamma'_s \longrightarrow \gamma_{aux} \longrightarrow \gamma'_t) \text{ iff } \begin{cases} \gamma_s \cong \gamma'_s \\ \gamma_t \cong \gamma'_t \\ \gamma_{aux} = \langle v, \sigma, \delta, \kappa \rangle \Rightarrow \mathfrak{b} = \mathsf{L}(v) \end{cases}$$

Next for each genuine configuration γ' of BA, we have that every run $\rho' = \gamma'_0 \longrightarrow \gamma'_1 .... \in \mathsf{Runs}_{\mathsf{BA}}(\gamma')$ of γ', with length greater than 0, can be parsed as a *macro transition run* $\mathsf{MTR}(\rho') = \mathsf{t}'_1\mathsf{t}'_2\mathsf{t}'_3.....$ where each t'_j is a macro transition. For example the finite run

$$\gamma'_0 \longrightarrow \gamma'_1 \longrightarrow \gamma_{aux_1} \longrightarrow \gamma'_2 \longrightarrow \gamma'_3 \longrightarrow \gamma_{aux_2} \longrightarrow \gamma'_4$$

which is a sequence of a delay, action (two-step), delay, action (two-step) will produce the macro transition trace $\mathsf{t}'_1\mathsf{t}'_2\mathsf{t}'_3\mathsf{t}'_4$ where $\mathsf{t}'_1 = \gamma'_0 \longrightarrow \gamma'_1$, $\mathsf{t}'_2 = \gamma'_1 \longrightarrow \gamma_{aux_1} \longrightarrow \gamma'_2$, $\mathsf{t}'_3 = \gamma'_2 \longrightarrow \gamma'_3$ and $\mathsf{t}'_4 = \gamma'_3 \longrightarrow \gamma_{aux_2} \longrightarrow \gamma'_4$.

Similarly to the macro transition runs, for each configuration γ of the network N, we can write each nonzero-length run, $\rho = \gamma_0 \xRightarrow{\mathfrak{b}_1} \gamma_1 \xRightarrow{\mathfrak{b}_2} \gamma_2... \in \mathsf{Runs}_{\mathsf{N}}(\gamma)$ of γ, as a *transition run* $\mathsf{TR}(\rho) = \mathsf{t}_1\mathsf{t}_2...$ where $\mathsf{t}_i = \gamma_{i-1} \xRightarrow{\mathfrak{b}_i} \gamma_i$ (for all $i \geq 1$).

Finally we lift the definition of $\cong$ to runs (of length $n > 0$). For a run ρ of the network, and a run ρ' which starts in a genuine configuration in the behaviour automaton we have that

$$\rho \cong \rho' \text{ iff } \begin{cases} \mathsf{TR}(\rho) \text{ and } \mathsf{MTR}(\rho') \text{ have the same length} \\ \forall i \geq 1 : \mathsf{TR}(\rho)(i) \cong \mathsf{MTR}(\rho')(i) \end{cases}$$

Therefore, $\rho \cong \rho'$, if and only if, the transition run $\mathsf{TR}(\rho)$ of ρ and the macro transition run $\mathsf{MTR}(\rho')$ of ρ' have the same length and they are equivalent stepwise.

The following fact follows from the method of constructing a behaviour automaton and states that equivalent configurations in the network N and its behaviour automaton BA produce equivalent execution traces.

FACT 7 *For a network* N*, its behaviour automaton* BA *and two configurations* $\gamma \in$ ***Config***$_{\mathsf{N}}$ *and* $\gamma' \in$ ***Config***$_{\mathsf{BA}}$ *such that* $\gamma \cong \gamma'$ *we have that :*

- $\forall \rho \in \mathsf{Runs}_{\mathsf{N}}(\gamma) : \exists \rho' \in \mathsf{Runs}_{\mathsf{BA}}(\gamma') : \rho \cong \rho'$
- $\forall \rho' \in \mathsf{Runs}_{\mathsf{BA}}(\gamma') : \exists \rho \in \mathsf{Runs}_{\mathsf{N}}(\gamma) : \rho \cong \rho'$

EXAMPLE 5.6 *Now consider the prefix*

$$pr = \gamma_0 \overset{p_1:\mathtt{in_1(x,x_1)}:m}{\Longrightarrow} \gamma_1 \overset{m:\mathtt{ch((y,z),(1,x))}:d}{\Longrightarrow} \gamma_2 \overset{d:\mathtt{out_1(z_1,z)}:c_1}{\Longrightarrow} \gamma_3$$

from our gateway's network from Example 5.4. An equivalent prefix in the behaviour automaton from Example 5.5 is

$$pr' = \gamma_0' \longrightarrow \gamma_{aux_1} \longrightarrow \gamma_1' \longrightarrow \gamma_{aux_2} \longrightarrow \gamma_2' \longrightarrow \gamma_{aux_3} \longrightarrow \gamma_3'$$

where $\gamma_0 = \langle \boldsymbol{q}_0, \sigma_0, \delta_0, \kappa_0 \rangle$*, and* $\gamma_0' = \langle 1, \sigma_0, \delta_0[\mathtt{r_u} \mapsto 0], \kappa_0 \rangle$*, and notice here that* $\mathsf{L}(1) = \boldsymbol{q}_0$*. Next,* $\gamma_{aux_1} = \langle 2, \sigma_0, \delta_0[\mathtt{r_u} \mapsto 0], \kappa_0 \rangle$ *with* $\mathsf{L}(2) = p_1 : \mathtt{in_1(x, x_1)} : m$*. We then have that,* $\gamma_1 = \langle \boldsymbol{q}_1, \sigma_1, \delta_1, \kappa_1 \rangle$*, and* $\gamma_1' = \langle 3, \sigma_1, \delta_1[\mathtt{r_u} \mapsto 0], \kappa_1 \rangle$ *with* $\mathsf{L}(3) = \boldsymbol{q}_1$*. Next,* $\gamma_{aux_2} = \langle 4, \sigma_1, \delta_1[\mathtt{r_u} \mapsto 0], \kappa_1 \rangle$ *with* $\mathsf{L}(4) = m : \mathtt{ch((y, z), (1, x))} : d$*. We continue and we have that,*$\gamma_2 = \langle \boldsymbol{q}_2, \sigma_2, \delta_2, \kappa_2 \rangle$*, and* $\gamma_2' = \langle 5, \sigma_2, \delta_2[\mathtt{r_u} \mapsto 0], \kappa_2 \rangle$ *with* $\mathsf{L}(5) = \boldsymbol{q}_2$*. Next,* $\gamma_{aux_3} = \langle 11, \sigma_2, \delta_2[\mathtt{r_u} \mapsto 0], \kappa_2 \rangle$ *with* $\mathsf{L}(11) = d : \mathtt{out_1(z_1, z)} : c_1$*. Finally,* $\gamma_3 = \langle \boldsymbol{q}_3, \sigma_3, \delta_3, \kappa_3 \rangle$*, and* $\gamma_3' = \langle 1, \sigma_3, \delta_3[\mathtt{r_u} \mapsto 0], \kappa_3 \rangle$ *with* $\mathsf{L}(1) = \boldsymbol{q}_3$*.*

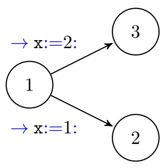
5.4.3 TCTL$^+$

For the behaviour automata, we define a new logic called **TCTL**$^+$ patterned after TCTL [ACD93]. The syntax of a **TCTL**$^+$ formula ψ is given by

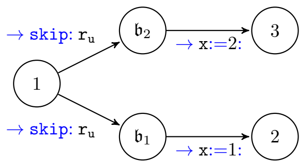
$$\psi ::= prop \mid g \mid set_1 \,\mathsf{rel}\, set_2 \mid \forall\Box\psi \mid \exists(\psi_1 U \psi_2) \mid \neg\psi \mid \psi_1 \wedge \psi_2$$

The basic formula *prop* is a proposition, which is either a behaviour, or a location vector, and it holds in a configuration if its vertex is labelled with *prop*. The set formulas are the same as in **BTCTL** . The $\forall\Box\psi$ formula holds in a configuration if for all of its execution traces, ψ holds in all the configurations of the trace, while for the $\exists(\psi_1 U \psi_2)$ to hold, it is sufficient that there exists an execution trace where ψ_1 holds for a prefix of the trace until ψ_2 holds at some configuration. The rest of the operators are the same as in **BTCTL** . The formal semantics of the **TCTL**$^+$ formulas is given by

$$\begin{array}{lll l}
\gamma' \models prop & \text{iff} & \gamma' = \langle v, \sigma, \delta, \kappa \rangle \Rightarrow \mathsf{L}(v) = prop & \\
\gamma' \models g & \text{iff} & \gamma' = \langle v, \sigma, \delta, \kappa \rangle \Rightarrow [\![g]\!](\sigma, \delta) & \\
\gamma' \models set_1 \,\mathsf{rel}\, set_2 & \text{iff} & \gamma' = \langle v, \sigma, \delta, \kappa \rangle \Rightarrow [\![set_1]\!]\kappa \,\mathsf{rel}\, [\![set_2]\!]\kappa & \\
\gamma' \models \forall\Box\psi & \text{iff} & \forall\gamma_0' \longrightarrow \gamma_1' \longrightarrow \gamma_2' .... \in \mathsf{Runs_{BA}}(\gamma') : & \forall i \geq 0 : \gamma_i' \models \psi \\
\gamma' \models \exists(\psi_1 U \psi_2) & \text{iff} & \exists\gamma_0' \longrightarrow \gamma_1' \longrightarrow \gamma_2' .... \in \mathsf{Runs_{BA}}(\gamma') : & \exists i : \gamma_i' \models \psi_2 \text{ and} \\
 & & & \forall j < i : \gamma_j' \models \psi_1 \\
\gamma' \models \psi_1 \wedge \psi_2 & \text{iff} & \gamma' \models \psi_1 \text{ and } \gamma' \models \psi_2 & \\
\gamma' \models \neg\psi & \text{iff} & \gamma' \not\models \psi &
\end{array}$$



(a) The timed automaton of the process p.



(b) The behaviour automaton of p.

Figure 5.5: An example of a timed automaton and its behaviour automaton.

Our goal is to transform a **BTCTL** formula ϕ into a **TCTL**$^+$ formula ψ, and then show that: for two equivalent configurations γ, and γ' of a network N, and its behaviour automaton BA respectively, the problem of checking the formula ϕ in γ can be reduced to the problem of checking the transformed formula ψ in γ', and vice versa.

We perform the transformation of the formulas using a function $\mathcal{T}[\![.]\!]$ as follows

$$
\begin{array}{lcl}
\mathcal{T}[\![g]\!] & = & g \\
\mathcal{T}[\![set_1 \ \mathsf{rel}\ set_2]\!] & = & set_1 \ \mathsf{rel}\ set_2 \\
\mathcal{T}[\![\forall\Box_{\mathfrak{b}}(\phi_1,\phi_2)]\!] & = & \forall\Box(\mathfrak{b} \Rightarrow (\mathcal{T}[\![\phi_1]\!] \ \wedge \exists(\mathfrak{b}\ U(\neg\mathfrak{b} \wedge \mathcal{T}[\![\phi_2]\!])))) \\
\mathcal{T}[\![\phi_1 \wedge \phi_2]\!] & = & \mathcal{T}[\![\phi_1]\!] \wedge \mathcal{T}[\![\phi_2]\!] \\
\mathcal{T}[\![\neg\phi]\!] & = & \neg\mathcal{T}[\![\phi]\!]
\end{array}
$$

The only non-trivial case here is the one of the formula $\forall\Box_{\mathfrak{b}}(\phi_1,\phi_2)$. To understand the translation recall that the satisfaction of the $\forall\Box_{\mathfrak{b}}(\phi_1,\phi_2)$ requires that whenever the behaviour $\mathfrak{b}$ appears in a networks's run, then ϕ_1 should hold in the pre- configuration and ϕ_2 in the post-configuration.

In the behaviour automaton BA, auxiliary configurations that are labelled with the behaviour $\mathfrak{b}$ are equal with the pre-configurations on the variable state, clock assignment and the writers state, and thus it is sufficient to check the precondition on them (i.e to check that $\mathcal{T}[\![\phi_1]\!]$). Next, we make use of the $\mathfrak{b}\ U(\neg\mathfrak{b} \wedge \mathcal{T}[\![\phi_2]\!])$ formula to check the formula $\mathcal{T}[\![\phi_2]\!]$ exactly at the successor of the auxiliary configuration (since the successor will not satisfy the behaviour $\mathfrak{b}$), which is again equal to the post-configuration.

Next, for the special cases $\forall\Box_{\mathfrak{b}}(\mathsf{tt},\phi_2)$ (ϕ_2 is not tt) and $\forall\Box_{\mathfrak{b}}(\phi_1,\mathsf{tt})$ (ϕ_1 is not tt) we shall omit the transformed formula that corresponds to the trivial formula tt, by writing $\mathcal{T}[\![\forall\Box_{\mathfrak{b}}(\mathsf{tt},\phi_2)]\!] = \forall\Box(\mathfrak{b} \Rightarrow \mathfrak{b}\ U(\neg\mathfrak{b}\wedge\mathcal{T}[\![\phi_2]\!]))$ for the first case and $\mathcal{T}[\![\forall\Box_{\mathfrak{b}}(\phi_1,\mathsf{tt})]\!] = \forall\Box(\mathfrak{b} \Rightarrow \mathcal{T}[\![\phi_1]\!])$ for the second case.

Example 5.7 *Take the security condition*

$$\Phi_{\mathtt{x}} = \forall\Box_{\mathfrak{b}_1}(\mathsf{tt}, P_{\mathtt{x}}) \wedge \forall\Box_{\mathfrak{b}_2}(\mathsf{tt}, P_{\mathtt{x}})$$

for the $\mathtt{x}$ *variable from Example 5.3, where* $\mathfrak{b}_1 = p_1 : \mathtt{in}_1(\mathtt{x}, \mathtt{x}_1) : m$ *and* $\mathfrak{b}_2 = p_2 : \mathtt{in}_2(\mathtt{x}, \mathtt{x}_2) : m$. *We apply our transformation and we get*

$$
\begin{aligned}
\mathcal{T}[\![\Phi_{\mathtt{x}}]\!] &= \mathcal{T}[\![\forall\Box_{\mathfrak{b}_1}(\mathtt{tt}, P_{\mathtt{x}}) \wedge \forall\Box_{\mathfrak{b}_2}(\mathtt{tt}, P_{\mathtt{x}})]\!] \\
&= \mathcal{T}[\![\forall\Box_{\mathfrak{b}_1}(\mathtt{tt}, P_{\mathtt{x}})]\!] \wedge \mathcal{T}[\![\forall\Box_{\mathfrak{b}_2}(\mathtt{tt}, P_{\mathtt{x}})]\!] \\
&= \forall\Box(\mathfrak{b}_1 \Rightarrow \mathfrak{b}_1\ U(\neg\mathfrak{b}_1 \wedge \mathcal{T}[\![P_{\mathtt{x}}]\!])) \wedge \forall\Box(\mathfrak{b}_2 \Rightarrow \mathfrak{b}_2\ U(\neg\mathfrak{b}_2 \wedge \mathcal{T}[\![P_{\mathtt{x}}]\!])) \\
&= \forall\Box(\mathfrak{b}_1 \Rightarrow \mathfrak{b}_1\ U(\neg\mathfrak{b}_1 \wedge P_{\mathtt{x}})) \wedge \forall\Box(\mathfrak{b}_2 \Rightarrow \mathfrak{b}_2\ U(\neg\mathfrak{b}_2 \wedge P_{\mathtt{x}}))
\end{aligned}
$$

Finally, *we shall assume that formulas in the pre-condition of the* $\forall\Box_{\mathfrak{b}}(\phi_1, \phi_2)$ *are not nested*. This assumption is essential for proving the correctness of our translation. The following example illustrates the problem if we allow nested formulas in the pre-condition.

EXAMPLE 5.8 *Consider the timed automaton of a process* p *(given in Figure 5.5a) with a variable* $\mathtt{x}$ *and a clock* $\mathtt{r}$*, and its behaviour automaton* BA *(given in Figure 5.5b), where* $\mathfrak{b}_1 = p : (\mathtt{x}, 1)$ *and* $\mathfrak{b}_2 = p : (\mathtt{x}, 2)$ *are the behaviours of the actions* $\mathtt{x}{:=}1$ *and* $\mathtt{x}{:=}2$ *respectively, and all the location invariants in the timed automaton of* p *are* tt.

Now let $\phi = \forall\Box_{\mathfrak{b}_1}(\forall\Box_{\mathfrak{b}_2}(\mathtt{tt}, \mathtt{x} = 1), \mathtt{tt})$ *and observe that every initial configuration of the process* p *does not satisfy* ϕ*, whereas every initial configuration of the behaviour automaton does satisfy the transformed formula* $\mathcal{T}[\![\phi]\!] = \forall\Box(\mathfrak{b}_1 \Rightarrow (\forall\Box(\mathfrak{b}_2 \Rightarrow \exists(\mathfrak{b}_2\ U(\neg\mathfrak{b}_2 \wedge \mathtt{x} = 1)))))$.

Although, we do not allow nested formulas in the pre-condition of the box operator, note that the proposed formula transformation is sufficient to express the access control policies of our interest.

Finally, we state the correctness of the function $\mathcal{T}[\![.]\!]$ with the following theorem

THEOREM 5.2 *For a network* N*, its behaviour automaton* BA*, a* ***BTCTL*** *formula* ϕ *and for every configuration* γ *and* γ' *of* N *and* BA *respectively, we have that if* $\gamma \cong \gamma'$ *then*

$$\gamma \models \phi \textit{ iff } \gamma' \models \mathcal{T}[\![\phi]\!]$$

5.4.4 Reduction Complexity

We give a computation bound for the algorithm of Table 5.2, that given a network $\mathsf{N} = (\mathsf{TA}_i)_{i\leq n}$ constructs the behaviour automaton $\mathsf{BA} = (\mathsf{Q}, \mathsf{E}, \mathsf{I}, \mathsf{L}, v_\circ)$. Assuming that the computation time of all the simple operations (creation of fresh vertices, setting of invariants e.t.c) is constant, we have that : let $K = |\mathsf{Q}_1| + ... + |\mathsf{Q}_n|$ and $E =$

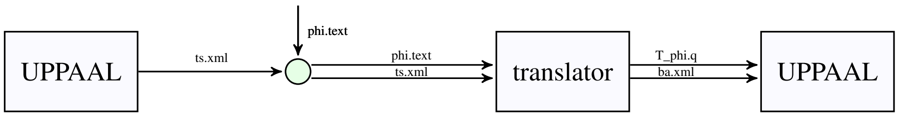


Figure 5.6: The architecture of the translator.

$|\mathsf{E}_1| + ... + |\mathsf{E}_n|$ then the first part of the algorithm is bounded by K^n. The second part iterates over the assignment edges and all the pairs of the auxiliary vertices and that is bounded by $E \times K^{2n} \times n$, where n corresponds to the computation bound of checking the third condition of the branch of the for-loop. Similarly to the second part of the algorithm the third part is bounded by $E^2 \times K^{2n} \times n$ and therefore for the total sum of those bounds we obtain a complexity of $\mathcal{O}(E^2 \times K^{2n} \times n)$. Finally, for a **BTCTL** formula ϕ the complexity of the transformation $\mathcal{T}[\![\phi]\!]$ is linear to the size of ϕ.

5.5 The Translator

We have implemented a translator in Java that works together with the model checker UPPAAL version 4.0 [UPP]. Figure 5.6 depicts the architecture of the translator.

UPPAAL is using a graphical interface in which one can model (draw) a network of timed automata. We first do this, and next UPPAAL saves it as a file in the eXtensible Markup Language (XML) [XML]; the xml file together with a text file that contains the desired property ϕ that we want to check are then being passed to the translator.

The translator parses the two files and produces an xml file which contains the behaviour automaton of the network together with a UPPAAL query file that includes the property $\mathcal{T}[\![\phi]\!]$. The two files are imported to UPPAAL and then one can check if the desired property holds.

Since UPPALL does not allow nested formulas nor supports the operator $\exists \phi_1 U \phi_2$, we had to find a workaround for some of the transformed formulas. The guards g are translated directly; for the set_1 rel set_2, we model a set as a bit array since UPPAAL supports multidimensional integer arrays and then we check the bit version of the relation rel . In case of the $\mathcal{T}[\![\forall\Box_{\mathfrak{b}}(\phi_1, \phi_2)]\!] = \forall\Box(\mathfrak{b} \Rightarrow (\mathcal{T}[\![\phi_1]\!] \wedge \exists(\mathfrak{b}\ U(\neg\mathfrak{b} \wedge \mathcal{T}[\![\phi_2]\!]))))$, UPPAAL allows labelling a vertex with a string (the name of the vertex) and thus auxiliary vertices with label $\mathfrak{b}$ have as a name a string that corresponds to the behaviour $\mathfrak{b}$. For the part $\mathfrak{b}\ U(\neg\mathfrak{b} \wedge \mathcal{T}[\![\phi_2]\!])$ we annotate the outgoing edges of the auxiliary vertices with an assignment to a fresh variable a that works as a switch. We switch on by $a := 1$, only when we leave the auxiliary vertex, and we switch off by $a := 0$, whenever we leave the successor of the auxiliary vertex. Thus the formula $\mathfrak{b}\ U(\neg\mathfrak{b} \wedge \mathcal{T}[\![\phi_2]\!])$

is transformed into the formula $a = 1 \Rightarrow \mathcal{T}[\![\phi_2]\!]$.

Finally, since the mapping κ is not part of the timed automata of UPPAAL, we had to find a way to encode it with our translator. To achieve this, we first associate each variable and each process of the network with a number, and next we model κ as a two-dimensional array, whose first index corresponds to a variable and its second to a process. For instance, assume that we want to model the state κ with $\kappa(x) = \{p\}$. If now x is associated with the number 1 and p is associated with the number 2, then $\kappa[1][2] = 1$, while for any other index $j \neq 2$, $\kappa[1][j] = 0$, modelling in that way that only p has written data in x. The edges of the automaton are also annotated with assignments to κ to capture the updates to it whenever the network performs an action.

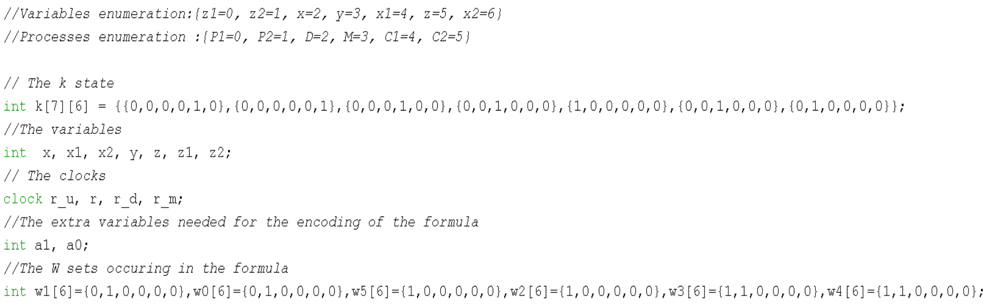
EXAMPLE 5.9 *We used our translator and verified the security conditions from Example 5.3, for our gateway Example 5.1.*

In Figure 5.7 (a), you can see the declarations of the various components of the behaviour automaton generated by our translator. In particular, the first two lines describe the way the enumeration has been done for creating the κ states, and on the third line the initial state $\kappa_\circ$ is being declared. Next, we have the declarations of the variables and clocks used by the automaton. Finally, we finish with the two variables $a1$ and $a0$, that will be used as switches in order to encode our formula, and the writers set appearing in our formula (to be explained shortly).

In Figure 5.7 (b), you can see the behaviour automaton generated from our gateway. Here, we have hided the actions, the labels of the vertices, and their invariants for readability issues.

Next, in Figure 5.7 (c), we see the formula $\Phi_{\mathtt{x}}$. The $< x >$ notation is used to describe the $\underline{x}$ expression.

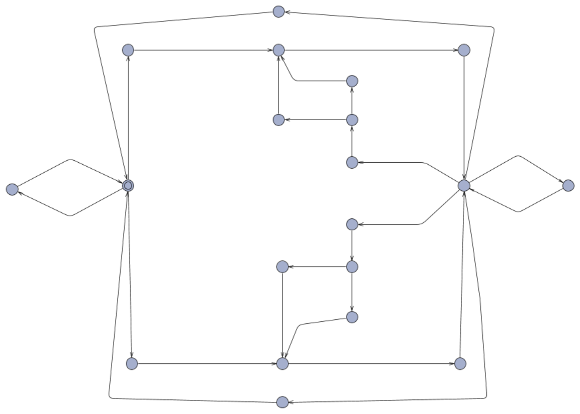
In Figure 5.7 (d), we see how the translation of $\Phi_{\mathtt{x}}$ looks. UPPAAL requires the nodes to have different names. Because of this, on the first line, you see that instead of having the formula ba.P1in1xx1M imply true which encodes the formula $p_1 : \mathtt{in}_1(\mathtt{x}, \mathtt{x}_1) : m \Rightarrow$ $\mathtt{tt}$, we need to enumerate all the vertices that come with the property $p_1 : \mathtt{in}_1(\mathtt{x}, \mathtt{x}_1)$, and use them before the implication. In our case, there are only two such vertices with labels P1in1xx1M0 and P1in1xx1M1. Next, the $a0$ variable is used as a switch for the first box formula. To understand the rest of the formula, take for example the part forall (i: int[0,5]) ba.k[2][i] <=ba.w5[i] which encodes the formula $\underline{\mathtt{x}} \subseteq \{p_1\}$. The writers set $\underline{\mathtt{x}}$ is described from the set $k[2]$, and the set $\{p_1\}$ is described from the set w5 (see the declarations in Figure 5.7 (a)). The rest of the translation can be described similarly.



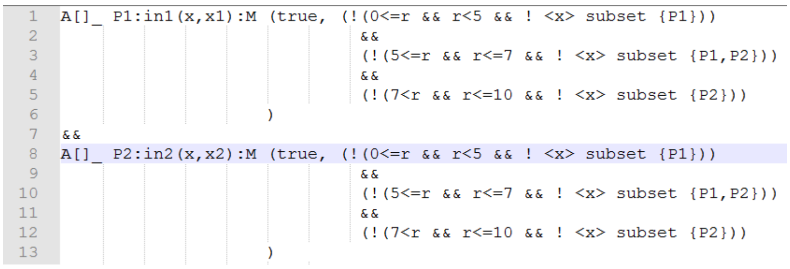
```
//Variables enumeration:{z1=0, z2=1, x=2, y=3, x1=4, z=5, x2=6}
//Processes enumeration :{P1=0, P2=1, D=2, M=3, C1=4, C2=5}

// The k state
int k[7][6] = {{0,0,0,0,1,0},{0,0,0,0,0,1},{0,0,0,1,0,0},{0,0,1,0,0,0},{1,0,0,0,0,0},{0,0,1,0,0,0},{0,1,0,0,0,0}};
//The variables
int  x, x1, x2, y, z, z1, z2;
// The clocks
clock r_u, r, r_d, r_m;
//The extra variables needed for the encoding of the formula
int a1, a0;
//The W sets occuring in the formula
int w1[6]={0,1,0,0,0,0},w0[6]={0,1,0,0,0,0},w5[6]={1,0,0,0,0,0},w2[6]={1,0,0,0,0,0},w3[6]={1,1,0,0,0,0},w4[6]={1,1,0,0,0,0};
```

(a) The declarations of the behaviour automaton.

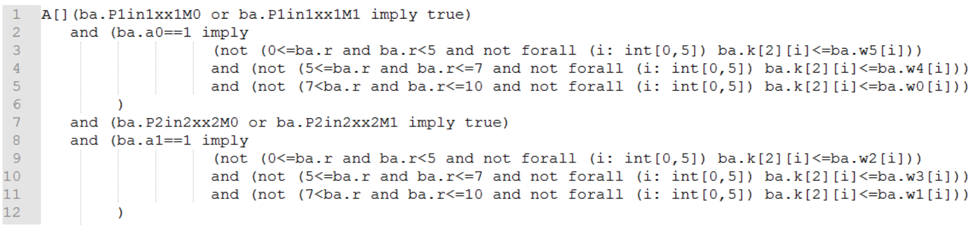


(b) The behaviour automaton.



```
A[]_ P1:in1(x,x1):M (true, (!(0<=r && r<5 && ! <x> subset {P1}))
                            &&
                            (!(5<=r && r<=7 && ! <x> subset {P1,P2}))
                            &&
                            (!(7<r && r<=10 && ! <x> subset {P2}))
                    )
&&
A[]_ P2:in2(x,x2):M (true, (!(0<=r && r<5 && ! <x> subset {P1}))
                            &&
                            (!(5<=r && r<=7 && ! <x> subset {P1,P2}))
                            &&
                            (!(7<r && r<=10 && ! <x> subset {P2}))
                    )
```

(c) The formula Φ_x



```
A[](ba.P1in1xx1M0 or ba.P1in1xx1M1 imply true)
   and (ba.a0==1 imply
                  (not (0<=ba.r and ba.r<5 and not forall (i: int[0,5]) ba.k[2][i]<=ba.w5[i]))
                  and (not (5<=ba.r and ba.r<=7 and not forall (i: int[0,5]) ba.k[2][i]<=ba.w4[i]))
                  and (not (7<ba.r and ba.r<=10 and not forall (i: int[0,5]) ba.k[2][i]<=ba.w0[i]))
       )
   and (ba.P2in2xx2M0 or ba.P2in2xx2M1 imply true)
   and (ba.a1==1 imply
                  (not (0<=ba.r and ba.r<5 and not forall (i: int[0,5]) ba.k[2][i]<=ba.w2[i]))
                  and (not (5<=ba.r and ba.r<=7 and not forall (i: int[0,5]) ba.k[2][i]<=ba.w3[i]))
                  and (not (7<ba.r and ba.r<=10 and not forall (i: int[0,5]) ba.k[2][i]<=ba.w1[i]))
       )
```

(d) The translated formula $\mathcal{T}[\![\Phi_x]\!]$.

Figure 5.7: An illustration of the translator on the Example 5.1.

5.6 Related Work

There are many other papers dealing with access control [RZFG01, KS16a, HLH14, HHD08, ZYL14, XAC], however without considering time-dependent security policies. A survey of access control models is available at [dVSS14].

A rich logic that allows reasoning about time-dependent policies, together with a proof checker for the logic is considered in [DGP08], while SecPAL [BFG10] is another logic that supports time-dependent policies, as well as the encoding of many well-known policy idioms such as DAC, MAC, RBAC, and ABAC. A somewhat different approach has been taken in [BHKZ11], where a (offline or online) monitor is used to enforce time-dependent access control, by checking a system's logs that records the different actions of the users in a database system. Our contribution focuses on the challenges of time- and data-dependent access control for embedded systems modeled as timed automata.

The work of [MS08] presents a formal specification and verification of the temporal role-based access control (TRBAC) model [BBF01], a flexible model in which the roles of the users of the system are enabled or disabled depending on the time of the system. They then use UPPAAL [UPP] to model a TRBAC system and verify the desired security policies. The same authors of this paper, present in [MSA11], an extension of this model which is based on the generalized-TRBAC (GTRBAC) model [JBLG05]. The work of [GBO12] considers spatial-TRBAC (STRBAC) models [CWW$^+$10] in which the rights of the user may depend on the time as well as on the location of the user; again the different roles of the system are modeled as timed automata and verified in UPPAAL. Although all of those models deal with an important number of access control policies at the subject-object level considering time dependencies, there are no information flow considerations that occur at the application level of the system.

The work of [PSL$^+$15] formalizes the timed decentralised label model (TDLM) an extension of the traditional and well-established decentralised label model (DLM) [ML97], which deals with both information flow and time-dependent security policies. However, their work does not consider an enforcement mechanism for the policies. Our key contribution is to develop a logic that allows the specification of time, data's content and information flow dependent policies for access control, and to make use of current model checkers such as UPPAAL [UPP] for the enforcement of the policies.

5.7 Conclusions

We have successfully shown how to enforce access control policies on networks of timed automata using a behaviour-based logic. The logic allows specification of data- and time-dependent security policies, an essential need in the modern world of cyberphysical systems. We have developed a sound reduction of a substantial fragment of our logic to a logic based on TCTL [ACD93], so that the model checking of the formulas can be performed by existing model checkers such as UPPAAL [UPP]. We implemented a translator which performs the reduction and together with UPPAAL it enforces access control policies. Finally, we illustrated our development using an example from the aerospace industry, where ensuring data's integrity is a life critical goal.

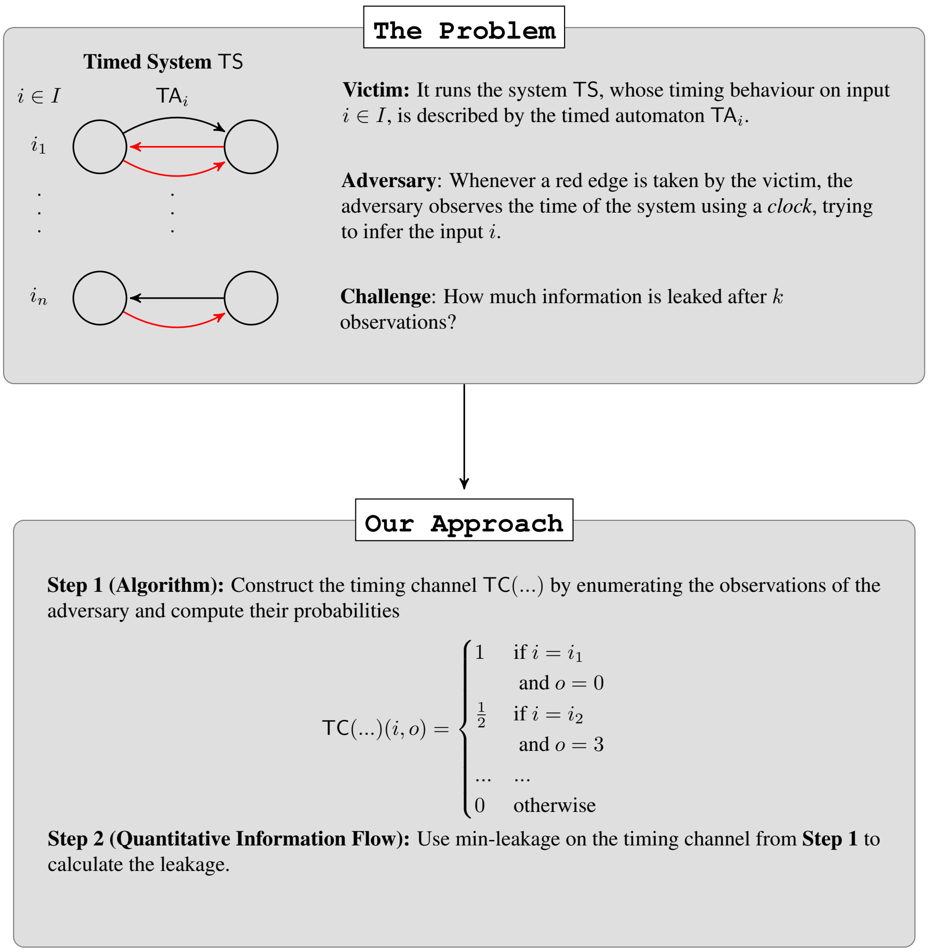
CHAPTER 6

Timing Leaks and Coarse-Grained Clocks

A widely deployed countermeasure against timing channel attacks is to reduce the accuracy of the clocks that are available to adversaries. While a number of high-profile attacks show that this mitigation can be side-stepped, there has not been a principled analysis of the degree of security it provides until now.

In this chapter, we perform the first information-flow analysis w.r.t. adversaries with low-resolution clocks.

We define clock resolution based on the period of the clock, which we call *grain*. Our analysis relies on a notion of adversaries with clocks that is parametric in the clock's grain and on the number of timing observations they can make. We connect this adversary to a victim modeled as a timed automata [AILS07, AD94] system with either deterministic or stochastic time semantics. We show that the resulting model is expressive enough to capture the essential aspects of state-of-the-art attacks [SWT01, KS16b, SMGM17, Wra91], such as the clock-edge, and the one-pad, and we present a novel timing technique, which we call the co-prime technique. In order to facilitate an information-flow analysis of the model, we present a novel algorithm that, given a system of timed automata and an adversary with a clock, constructs a timing channel that represents the information that the adversary can extract from the system. The main challenge of the construction is computing the probabilities of the adversary's timing observations, since (stochastic) timed automata semantics are defined based on



The Problem

Victim: It runs the system TS, whose timing behaviour on input $i \in I$, is described by the timed automaton TA_i.

Adversary: Whenever a red edge is taken by the victim, the adversary observes the time of the system using a *clock*, trying to infer the input i.

Challenge: How much information is leaked after k observations?

Our Approach

Step 1 (Algorithm): Construct the timing channel TC(...) by enumerating the observations of the adversary and compute their probabilities

$$\mathsf{TC}(...)(i, o) = \begin{cases} 1 & \text{if } i = i_1 \\ & \text{ and } o = 0 \\ \frac{1}{2} & \text{if } i = i_2 \\ & \text{ and } o = 3 \\ ... & ... \\ 0 & \text{otherwise} \end{cases}$$

Step 2 (Quantitative Information Flow): Use min-leakage on the timing channel from **Step 1** to calculate the leakage.

Figure 6.1: The idea of our development in Chapter 6.

general probability measures.

The construction enables us to leverage state-of-the-art techniques for quantitative information-flow analysis for formalizing and computing leakage to adversaries with fine-grained clocks.

Using this approach, we derive the following insights:

- While it is well-known that coarse-grained clocks can be bypassed using attack techniques such as the clock-edge and the one-pad, we show here that a coarse-grained clock might leak even more than a fine-grained clock without using any attack technique.

• We provide sufficient conditions for when a coarse-grained clock leaks less than a fine-grained one. In particular, we show that when the system is deterministic and we increase the grain of the clock by a multiple, the corresponding timing channel leaks less information.

• We show that the different techniques to construct fine-grained clocks, namely the one-pad technique, the clock-edge technique, and the co-prime technique, form a strict hierarchy in terms of the amount of information the adversary can extract.

Finally, we illustrate our approach on the distributed system of Example 2.11. The idea of our approach is depicted in Figure 6.1.

Chapter Organisation In Section 6.1, we describe the countermeasure of reducing clock resolution. In Section 6.2, we give semantics of (stochastic) timed automata, and in Section 6.3 we give the models of timed automata systems, and adversaries with clocks. In Section 6.4, we recall the basics of quantitative information flow, and we present our algorithm for constructing timing channels of systems. In Section 6.5, we present our theoretical insights for the case of timing channels of deterministic systems, and the timing techniques one-pad, clock-edge and co-prime. In Section 6.6, we apply our techniques on the Example 2.11. Section 6.7, and Section 6.8 discusses related work and conclusions respectively.

6.1 Coarse-Grained Clocks

Recall that a timing channel is a mechanism which reveals information through the timing behaviour of a system. Information conveyed by timing channels has been used by adversaries to recover cryptographic keys, where the timing channel is built by measuring cryptographic or cache-dependent operations, and by malicious websites which correlate this information with the internal state of a victim who visits the website [Koc96, SWT01, BT11, VK17, OKSK15, FS00, AKM$^+$15]. There are two main approaches to defeating timing channels. The first approach relies on closing or mitigating the channel. Examples of this approach include:

• Constant-time software [ABB$^+$16], which is a coding discipline that forbids that branching decisions, memory access patterns, and variable-latency instructions depend

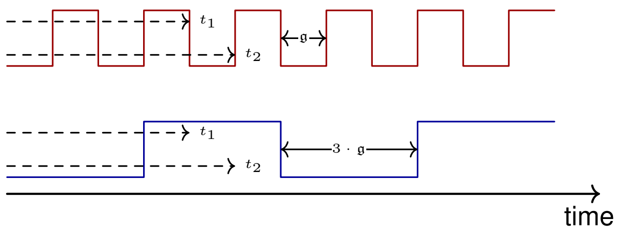


Figure 6.2: The countermeasure of decreasing clock resolution.

on secrets. Constant-time defeats completely timing channels. However, it is very difficult to achieve due to the complexity of modern software, and hardware architecture, and the efficiency constraints which are essential in certain cases.

• Input blinding, which decorrelates the payload of cryptographic algorithms from the execution time, making it provably difficult for the adversary to recover the cryptographic key [CRS83, KD09]. The guarantees of this countermeasure have been studied formally [CRS83, KD09]. However, it is limited to cryptographic applications only.

• Predictive mitigation techniques, which group timing observations into epochs of increasing duration, thereby provably reducing the amount of leaked information [ZAM11]. This countermeasure has been shown [BDK13] to be effective against timing channel attacks created due to packet delays in web-applications. However, it can still create performance overhead.

The other approach relies on reducing the adversary's ability to make precise timing observations, e.g., by decreasing an adversary's clock resolution or removing the clock entirely. A clock is a counter that is being increased every after a fixed period. Increasing this period results in a clock with lower resolution, changing the perception of the adversary regarding the timing behaviour of the system, and hence making it more difficult for it to build a timing channel.

This countermeasure is attractive because it does not imply performance overhead, while it works for adversaries that run on the same platform (i.e. not for remote attackers where one cannot control the clock). In particular, it has been proposed in the literature for mitigation of interrupt-related timing channels [MS07] and deployed in all major browser implementations after the first browser-based side-channel attacks [OKSK15].

Figure 6.2 shows an example of the countermeasure. The clock on the top is being increased with a period of g, and it is able to distinguish between two events that arrive at times t_1 and t_2 resp. since at time t_1 the clock has been increased 4 times, while at t_2, 5 times. By increasing this period by a factor of $2 \cdot g$ (Figure 6.2 bottom) the times of the two events become indistinguishable since in both cases the clock will be increased only once.

Unfortunately, low-resolution clocks are not an effective defense against many kinds of attacks. Several authors have successfully side-stepped this defense by building their own high-resolution clocks from primitives such as low-resolution clocks and simple counter processes [SWT01, KS16b, SMGM17, Wra91]. Therefore a principled analysis of this countermeasure and its security guarantees is needed.

6.2 (Stochastic) Timed Automata

In this section we give our models of (stochastic) timed automata. *We make sure to say dense clocks when we talk about timed automata, whereas we say clocks when we talk about the adversary.*

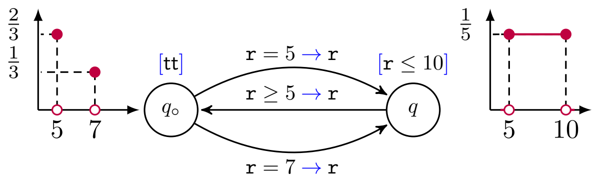
We use the same model of a timed automaton $\mathsf{TA} = (\mathsf{Q}, \mathsf{E}, \mathsf{I}, q_\circ)$, as described in Section 3.1, however, we now drop the data variables, and our automaton uses only dense clocks described by the set **Clocks**. The edges in E take the form $(q_s, g \rightarrow \boldsymbol{r}, q_t)$ where $q_s \in \mathsf{Q}$ is the source location and $q_t \in \mathsf{Q}$ is the target location, the guard g now includes constraints over dense clock variables only, and $\boldsymbol{r}$ is the sequence of dense clocks to reset. Finally, $q_\circ$ is the initial node and I imposes an invariant (i.e a guard) on each node.

The semantics of a timed automaton is given by a transition system whose configurations have the form $\gamma = \langle q, \delta \rangle \in \textbf{Config}$ with $\gamma \models \mathsf{I}(q)$ and the transitions are described by an initial delay that increases the values of all the dense clocks followed by an action. Therefore, whenever $e = (q_s, g \rightarrow \boldsymbol{r}, q_t)$ is in E we have the transition :

$$\langle q_s, \delta \rangle \xrightarrow{t,e} \langle q_t, \delta' \rangle \text{ if } \begin{cases} t \geq 0, \\ \delta + t \models \mathsf{I}(q_s) \wedge g, \\ \delta' = (\delta + t)[\boldsymbol{r} \mapsto 0], \\ \delta' \models \mathsf{I}(q_t) \end{cases}$$

where t corresponds to the initial delay. The rule ensures that after the delay t the invariant and the guard are satisfied ($\models$) with the valuation $\delta + t$, and then updates the valuation $\delta + t$ to δ' by resetting the dense clocks in $\boldsymbol{r}$. Finally, it ensures that the invariant is satisfied in the resulting configuration that uses the valuation δ'. The initial configuration of the automaton have all the dense clocks initialised to 0, and has the form $\langle q_\circ, \lambda r.0 \rangle$ where $\lambda r.0 \models \mathsf{I}(q_\circ)$, and we write $\gamma_{q_\circ}$ for $\langle q_\circ, \lambda r.0 \rangle$.

Definition 6.1 (**Run**). A *run* of a configuration $\gamma = \langle q, \delta \rangle \in \textbf{Config}$ is a (possibly



2/3
1/3
5
7
[tt]
$q_\circ$
$\mathtt{r} = 5 \to \mathtt{r}$
$\mathtt{r} \geq 5 \to \mathtt{r}$
$\mathtt{r} = 7 \to \mathtt{r}$
[r ≤ 10]
q
1/5
5
10

Figure 6.3: An example of a (stochastic) timed automaton.

infinite) sequence

$$\langle q_0, \delta_0 \rangle \xrightarrow{t_1, e_1} \dots \xrightarrow{t_n, e_n} \langle q_n, \delta_n \rangle \xrightarrow{t_{n+1}, e_{n+1}} \dots$$

where $\langle q_0, \delta_0 \rangle = \langle q, \delta \rangle$ and we write $\mathsf{Runs}(\gamma)$ for the set of runs of γ.

Finally, a *run* of a timed automaton is described by a run $\rho \in \mathsf{Runs}(\gamma_{q_\circ})$ of the initial configuration $\gamma_{q_\circ}$ and we write $\mathsf{Runs}(\mathsf{TA})$ for $\mathsf{Runs}(\gamma_{q_\circ})$.

We now give our notion of determinism for timed automata

Definition 6.2 (**Deterministic timed automaton**). A timed automaton TA is deterministic whenever $\mathsf{Runs}(\mathsf{TA}) = \{\rho\}$ (for some run ρ).

Example 6.1 *Figure 6.3 depicts a timed automaton with two locations $q_\circ$ (the initial), q and a dense clock r. The automaton moves from $q_\circ$ to q after delaying 5 or 7 time units, and from q to $q_\circ$ after delaying for a time between* 5 *and* 10*. The dense clock r is reset after each move. Formally, we have that* $\mathsf{TA} = (\mathsf{Q}, \mathsf{E}, \mathsf{I}, q_\circ)$*, where* $\mathsf{Q} = \{q_\circ, q\}$*,* $\mathsf{E} = \{e_1, e_2, e_3\}$ *where* $e_1 = (q_\circ, \mathtt{r} = 5 \to \mathtt{r}, q)$*,* $e_2 = (q_\circ, \mathtt{r} = 7 \to \mathtt{r}, q)$*,* $e_3 = (q, \mathtt{r} \geq 5 \to \mathtt{r}, q_\circ)$ *and the invariant mapping is* $\mathsf{I} = [q_\circ \mapsto \mathtt{tt}, q \mapsto \mathtt{r} \leq 10]$*. A prefix of an example run of the automaton is*

$$\langle q_\circ, [\mathtt{r} \mapsto 0] \rangle \xrightarrow{5, e_1} \langle q, [\mathtt{r} \mapsto 0] \rangle \xrightarrow{5.5, e_3}$$
$$\langle q_\circ, [\mathtt{r} \mapsto 0] \rangle \xrightarrow{7, e_2} \langle q, [\mathtt{r} \mapsto 0] \rangle \xrightarrow{7.97, e_3} \langle q_\circ, [\mathtt{r} \mapsto 0] \rangle \dots$$

We define stochastic semantics for timed automata based on [Car17, BBB+14]. Stochastic timed automata are stochastic processes, where at each transition the automaton first chooses randomly a delay and then an edge. We start by defining some auxiliary operators. Let $\mathsf{TA} = (\mathsf{Q}, \mathsf{E}, \mathsf{I}, q_\circ)$ be a timed automaton, then for a configuration $\gamma \in \mathbf{Config}$ and an edge $e \in \mathsf{E}$ we define

$$\mathsf{Int}(\gamma, e) = \left\{ t \in \mathbb{R}_{\geq 0} \mid \exists \gamma' \in \mathbf{Config} : \gamma \xrightarrow{t, e} \gamma' \right\}$$

to be the set of delays such that the edge e can be taken from γ after such a delay, and we write

$$\mathsf{Int}(\gamma) = \bigcup_{e \in \mathsf{E}} \mathsf{Int}(\gamma, e)$$

for the set of all possible delays that γ can perform.[1] Finally, for a configuration $\gamma \in$ **Config** we write

$$\mathsf{Enab}(\gamma) = \{e \mid e \in \mathsf{E} : \mathsf{Int}(\gamma, e) \neq \emptyset\}$$

for the set of enabled edges of γ (i.e the ones that at least one transition is possible by taking them). We are now ready to give the definition of stochastic timed automaton.

DEFINITION **6.3** (**Stochastic Timed Automata**.) Given a timed automaton $\mathsf{TA} = (\mathsf{Q}, \mathsf{E}, \mathsf{I}, q_\circ)$, a *stochastic timed automaton* $\mathsf{STA} = (\mathsf{TA}, (\mu_\gamma)_{\gamma \in \textbf{Config}}, (\kappa_\gamma)_{\gamma \in \textbf{Config}})$ is a timed automaton equipped with the families $(\mu_\gamma)_{\gamma \in \textbf{Config}}$, $(\kappa_\gamma)_{\gamma \in \textbf{Config}}$ where

- for each configuration $\gamma \in$ **Config**, $\mu_\gamma : \mathcal{B}(\mathbb{R}_{\geq 0}) \mapsto [0, 1]$ is a probability measure over the Borel[2] σ-algebra $\mathcal{B}(\mathbb{R}_{\geq 0})$ such that $\mu_\gamma(\mathsf{Int}(\gamma)) = 1$;
- for each configuration $\gamma \in$ **Config**, $\kappa_\gamma : \mathsf{E} \mapsto [0, 1]$ is a probability distribution over the set of edges such that for all $e \in \mathsf{E} : \kappa_\gamma(e) > 0$ iff $e \in \mathsf{Enab}(\gamma)$, and we also have that $\sum_{e \in \mathsf{Enab}(\gamma)} \kappa(e) = 1$

For a run of the automaton now, at each transition (in the run) from a configuration $\langle q, \delta \rangle$, first, a delay t is chosen according to $\mu_{\langle q, \delta \rangle}$ and then an edge according to $\kappa_{\langle q, \delta + t \rangle}$.

EXAMPLE **6.2** *Back to Example 6.1, a stochastic version of the automaton is depicted in Figure 6.3, where for configurations of the initial location $q_\circ$ the delay is chosen according to a (discrete) probability distribution that is expressed with a probability mass function (Figure 6.3 left) that assigns $\frac{2}{3}$ and $\frac{1}{3}$ to the delays 5 and 7 resp., while for the configurations of the location q the delay is chosen according to a (continuous) uniform probability distribution that is expressed using the density function (Figure 6.3 right) over the interval* $[5, 10]$. *In both cases after the delay has been chosen, there is exactly one enabled edge, that is chosen with probability 1.*

6.3 Timed Systems and Adversaries with Clocks

In this section, we give our models of systems of timed automata and adversaries with clocks. Systems of timed automata, or simply *timed systems*, will be used to model the

[1] As in [Car17, BBB$^+$14] we shall assume that for all $\gamma \in$ **Config**, $\mathsf{Int}(\gamma) \neq \emptyset$, that is that the automaton is *deadlock-free*.

[2] The Borel set $\mathcal{B}(\mathbb{R}_{\geq 0})$ is the smallest σ-algebra generated by the open sets of $\mathbb{R}_{\geq 0}$.

timing behaviour of a system that operates on some secret provided by a victim. An adversary then tries to infer information of the victim's secret by measuring the timing behaviour of the system using a clock.

6.3.1 Timed Systems

Let I be a finite set of *secret* inputs of the victim. A *timed system* S is either a family of deterministic timed automata $(\mathsf{Q}_i, \mathsf{E}_i, \mathsf{I}_i, q_\circ^i)_{i\in I}$, or a family of stochastic timed automata $((\mathsf{Q}_i, \mathsf{E}_i, \mathsf{I}_i, q_\circ^i), (\mu_\gamma)^i_{\gamma\in\mathbf{Config}}, (\kappa_\gamma)^i_{\gamma\in\mathbf{Config}})_{i\in I}$, where for each $i \in I$, the automaton $\mathsf{TA}_i = (\mathsf{Q}_i, \mathsf{E}_i, \mathsf{I}_i, q_\circ^i)$ describes the timing behaviour of the system on input $i \in I$. In the first case the system will be called *deterministic* and in the second *stochastic*. We will also write $\mathsf{Q} = \bigcup_{i\in I} \mathsf{Q}_i$, $\mathsf{E} = \bigcup_{i\in I} \mathsf{E}_i$ and $\mathsf{Runs}(\mathsf{S}) = \bigcup_{i\in I} \mathsf{Runs}(\mathsf{TA}_i)$.

Example 6.3 *Consider the input set $I = \{i_1, i_2\}$ and the stochastic system S, where for the input i_1 the behaviour of S is described by the stochastic timed automaton from Example 2. For input i_2 the behaviour of S is given by a variation of the stochastic timed automaton of Example 6.2, where now the probability measure over the delays for the configurations of the initial location is given by a discrete uniform distribution that is described by a probability mass function that assigns $\frac{1}{2}$ to both delays 5 and 7.*

6.3.2 Clocks

The main tool of the adversary for building a timing channel from a timed system is a *clock*. A clock is a counter that is being increased after a constant period $\mathfrak{g}$, discretizing in this way the time domain $\mathbb{R}_{\geq 0}$ in buckets of size $\mathfrak{g}$.

This fixed period $\mathfrak{g}$ between two consecutive increments of the clock is called its *grain* or *granularity*. The grain of the clock is the smallest time unit that it can measure. A point in time where a clock is being incremented is called a *clock-edge*, and this increment is equal to the clock's grain $\mathfrak{g}$.

Formally, a clock is given by the following definition

Definition 6.4 (**Clock.**) A *clock* $c : \mathbb{R}_{\geq 0} \mapsto \mathbb{N}$ with *granularity* or *grain* $\mathfrak{g} \in \mathbb{N}_{>0}$ is a step function over the time domain $\mathbb{R}_{\geq 0}$, where at time $t \in \mathbb{R}_{\geq 0}$ the value of c is

$$c(t) = \left\lfloor \frac{t}{\mathfrak{g}} \right\rfloor \cdot \mathfrak{g}$$

and $\lfloor . \rfloor$ is the floor function.

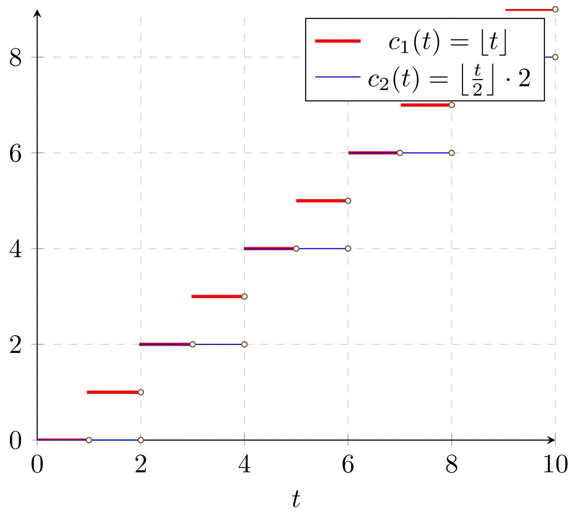


Figure 6.4: Two clocks c_1 and c_2 with grains 1 and 2 respectively.

A fine-grained clock gives more precise measurements, and intuitively one could think that an adversary with such a clock is a bigger threat in comparison to one with a coarse-grained clock; however, we will show later that this is not always the case.

EXAMPLE 6.4 *Consider the clocks c_1 and c_2 of grain 1 and 2 respectively, given in Figure 6.4. For the time points 0, 1.3, 2.5, 3.6... the value of c_1 is 0, 1, 2, 3... ,while for c_2 we have 0, 0, 2, 2,... For every clock-edge of c_2 we have two clock-edges of c_1. The clock-edges of c_1 happen at the time points 1, 2, 3,... while the ones of c_2 at the time points 2, 4, 6,...*

We now give two facts, which will be later used in the proofs of our two main theorems (Theorem 6.11 and Theorem 6.12 resp.).

FACT 8 *Let c_1 and c_2 be two clocks with grains $\mathfrak{g}_1$ and $\mathfrak{g}_2$ respectively. If $\mathfrak{g}_2$ is a multiple of $\mathfrak{g}_1$ then $\forall t_1, t_2 \in \mathbb{R}_{\geq 0} : (c_1(t_1) = c_1(t_2) \Rightarrow c_2(t_1) = c_2(t_2))$*

FACT 9 *Let c be a clock with grain $\mathfrak{g}$ and $n \in \mathbb{N}$, then for $t_1, t_2 \in \mathbb{R}_{\geq 0}$: $c(t_1 + n) = c(t_2 + n) \Leftrightarrow c(t_1 + (n \bmod \mathfrak{g})) = c(t_2 + (n \bmod \mathfrak{g}))$.*

6.3.3 Adversaries with Clocks

Let S be a (deterministic or stochastic) system. We model the *view* of an adversary on the runs of a system as a function $\mathsf{view}_c : \mathsf{Runs}(\mathsf{S}) \mapsto O$ that maps runs to a finite set of sequence of observations $O \subseteq \mathbb{N}^+$, obtained by making timing measurements using a clock c.

In particular, for the system we assume a finite set of *public edges* $\mathsf{E}_{\mathsf{pub}} \subseteq \mathsf{E}$ such that whenever the system performs a transition using this edge the adversary makes a timing observation using his clock (to be explained shortly).

We consider adversaries that make k (positive integer) number of timing observations and we also assume that each run of the system visits *at least* k times the public edges in $\mathsf{E}_{\mathsf{pub}}$ (not necessarily all of them).

For a run $\rho = \gamma_0 \xrightarrow{t_1,e_1} \gamma_1 .... \xrightarrow{t_n,e_n} \gamma_n ... \in \mathsf{Run}(\mathsf{S})$, let $j_1, ..., j_k$ to be the unique ordered sequence of indices of the first k public edges appearing in ρ, we then have that the view of the adversary on ρ is given by

$$\mathsf{view}_c(\rho) = (c(t'_1), c(t'_2), ..., c(t'_k))$$

where for $i \in \{1, ..., k\}$, $t'_i = t_1 + ... + t_{j_i}$ is the time moment, when the adversary performs his i-th observation. We sometimes refer to the sequence t'_1,...,t'_k as the *k-time sequence* of ρ.

To wrap up everything from above we give the definition of an *attack scenario*.

DEFINITION 6.5 (**Attack Scenario**). An attack scenario is a quadruple $\mathsf{AS} = (\mathsf{S}, \mathsf{E}_{\mathsf{pub}}, c, k)$ where S is a system, $\mathsf{E}_{\mathsf{pub}}$ are its public edges, c is the clock of the adversary, and k is the number of his timing observations.

EXAMPLE 6.5 *Consider the attack scenario $\mathsf{AS} = (\mathsf{S}, \mathsf{E}_{pub}, c, k)$ where S is the stochastic system given in Example 6.3 over the input set $I = \{i_1, i_2\}$, $\mathsf{E}_{pub} = \mathsf{E}$ is the set of observable edges of the system (i.e all the edges are observable), the adversary is using a clock c with grain $\mathfrak{g} = 5$, and performs k=2 timing observations.*

Consider the following prefix of a run of the system that corresponds to the input i_1

$$\langle q_\circ, [\mathtt{r} \mapsto 0]\rangle \xrightarrow{5,e_1} \langle q, [\mathtt{r} \mapsto 0]\rangle \xrightarrow{5.5,e_3}$$
$$\langle q_\circ, [\mathtt{r} \mapsto 0]\rangle \xrightarrow{7,e_2} \langle q, [\mathtt{r} \mapsto 0]\rangle \xrightarrow{7.97,e_3} \langle q_\circ, [\mathtt{r} \mapsto 0]\rangle ...$$

Since all of the edges are observable and the adversary makes $k = 2$ timing observations, the 2-time sequence of ρ is $t'_1 = 5$ and $t'_2 = t'_1 + 5.5 = 10.5$. For the values of *the clock* c *at those two time points we have* $c(t'_1) = 5$ *and* $c(t'_2) = 10$*, and therefore the view of the adversary on* ρ *is*

$$\mathsf{view}_c(\rho) = (c(t'_1), c(t'_2)) = (5, 10)$$

6.4 Quantifying Leakage in Timed Systems

In this section, we give an algorithm that given an attack scenario constructs the corresponding timing channel. Based on the timing channel, one can then quantify the leakage of the timed system using standard measures from quantitative information-flow. We start by recalling some basics of quantitative information-flow.

6.4.1 Timing Channels and Min-Leakage

For the rest of this subsection we assume a random variable $\mathcal{I}$ with range the *secret* input space I of a timed system, a probability distribution $p_{\mathcal{I}}$ on I, and a random variable $\mathcal{O}$ with range the *public* set of timing observations O of the adversary. Our definitions are based on [ACPS12, Smi09] and all the logarithms have base two.

The threat of the adversary guessing the secret input with one try, before making any timing observation, is given by the min-vulnerability of $\mathcal{I}$ defined as

$$\mathsf{V}(p_{\mathcal{I}}) = \ \text{max}_{i \in I} p_{\mathcal{I}}(i)$$

Min-vulnerability expresses that the adversary will choose for his guess the input that is more probable.

The relationship between the input space and the observations of the adversary is given by a timing channel $\mathsf{TC} : I \times O \mapsto [0, 1]$, that is a probability transition matrix, where for $i \in I$ and $o \in O$, $\mathsf{TC}(i, o)$ is the probability of $\mathcal{O} = o$ conditioned on $\mathcal{I} = i$ (i.e the conditional probability).

The expected probability of the adversary guessing the secret input, given his timing channel, is given by the conditional min-vulnerability of $\mathcal{I}$ and the timing channel TC, by

$$\mathsf{V}(p_{\mathcal{I}}, \mathsf{TC}) = \sum_{o \in O} \text{max}_{i \in I} p_{\mathcal{I}}(i) \cdot \mathsf{TC}(i, o)$$

The min-vulnerability and the conditional min-vulnerability, can be turned into entropies by taking their negative logarithm [Smi09].

For measuring the leakage of a timing channel we have the min-leakage [Smi09]

$$\mathsf{L}_{\min}(p_{\mathcal{I}}, \mathsf{TC}) = \log \frac{\mathsf{V}(p_{\mathcal{I},\mathsf{TC}})}{\mathsf{V}(p_{\mathcal{I}})}$$

and the min-capacity

$$\mathsf{C}_{\min}(\mathsf{TC}) = \sup_{p_{\mathcal{I}}} \mathsf{L}_{\min}(p_{\mathcal{I}}, \mathsf{TC})$$

The min-capacity is the worst-case leakage and it is realised over a uniform prior $p_{\mathcal{I}}$ [ACPS12]. Based on min-leakage we can order timing channels as

Definition 6.6 (**Ordering on Channels**) Given a random variable $\mathcal{I}$ with range I, two random variables $\mathcal{O}_1$, $\mathcal{O}_2$ with range O_1 and O_2 resp., and the timing channels $\mathsf{TC}_1 : I \times O_1 \mapsto [0,1]$ and $\mathsf{TC}_2 : I \times O_2 \mapsto [0,1]$, we write

$$\mathsf{TC}_1 \preceq \mathsf{TC}_2 \quad \text{if} \quad \forall p_{\mathcal{I}} : \mathsf{L}_{\min}(p_{\mathcal{I}}, \mathsf{TC}_1) \leq \mathsf{L}_{\min}(p_{\mathcal{I}}, \mathsf{TC}_2)$$

A special case of a timing channel is a *deterministic* timing channel. A timing channel $\mathsf{TC} : I \times O \mapsto [0,1]$ is deterministic whenever $\forall i \in I : \exists o \in O : \mathsf{TC}(i,o) = 1$ (i.e each row of the channel contains exactly one 1).

Recall [ACPS12, Smi09] that a deterministic channel $\mathsf{TC} : I \times O \mapsto [0,1]$ gives rise to an equivalence relation (or partition) on I, given by

$$i_1 \equiv_{\mathsf{TC}} i_2 \quad \text{iff} \quad \exists o \in O : \mathsf{TC}(i_1, o) = 1 = \mathsf{TC}(i_2, o)$$

Two secrets are indistinguishable to the the adversary if and only if they give the same observation through the timing channel TC. For example if $\equiv_{\mathsf{TC}}$ is equal to $\top = I \times I$ then $\equiv_{\mathsf{TC}}$ describes no leakage since all the secrets are related. On the other hand if $\equiv_{\mathsf{TC}}$ is equal to the identity relation $\bot = \{(i,i) \mid i \in I\}$ we have that everything is leaked since each secret produces a unique observable. In any other case where the equivalence relation $\equiv_{\mathsf{TC}}$ is such that $\bot \subset \equiv_{\mathsf{TC}} \subset \top$, we have partial information about the secret.

Deterministic channels can be ordered based on their equivalence relation by partition refinement.

Definition 6.7 (**Partition Refinement**). Given deterministic channels $\mathsf{TC}_1 : I \times O_1 \mapsto [0,1]$ and $\mathsf{TC}_2 : I \times O_2 \mapsto [0,1]$ we write $\mathsf{TC}_1 \sqsubseteq \mathsf{TC}_2$ if the partition of TC_1 is refined by the partition of TC_2.

The following theorem from [ACPS12, Smi09] shows that the leakage ordering corresponds to the partition refinement ordering.

```
Step 1. For each $i \in I$ let $O_i = \{\mathsf{view}_c(\rho) \mid \rho \in \mathsf{Runs}(\mathsf{TA}_i)\}$.
Step 2. Let $\mathcal{I}$ to be a random variable with range the input set $I$ and
$\mathcal{O}$ to be a random variable with range $O = \bigcup_i O_i$. For the timing
channel $\mathsf{TC}(\mathsf{AS}) : I \times O \mapsto [0,1]$, and for $i \in I$, and $o \in O$ :
  if S is deterministic and $o \in O_i$, then set $\mathsf{TC}(\mathsf{AS})(i,o) = 1$.
  if S is stochastic and $o \in O_i$, then set $\mathsf{TC}(\mathsf{AS})(i,o) = \mathsf{P}_{\gamma_{q_0^i}}(\mathsf{view}_c^{-1}(o))$.
  Otherwise, set $\mathsf{TC}(\mathsf{AS})(i,o) = 0$.
```

Table 6.1: The algorithm for constructing the timing channel $\mathsf{TC}(\mathsf{AS})$ of an attack scenario $\mathsf{AS} = (\mathsf{S}, \mathsf{E}_{\text{pub}}, c, k)$.

THEOREM 6.8 *Given deterministic channels* $\mathsf{TC}_1 : I \times O_1 \mapsto [0,1]$ *and* $\mathsf{TC}_2 : I \times O_2 \mapsto [0,1]$ *we have*

$$\mathsf{TC}_1 \sqsubseteq \mathsf{TC}_2 \textit{ iff } \mathsf{TC}_1 \preceq \mathsf{TC}_2$$

6.4.2 Timing Channels of Deterministic Systems

We now show how one can construct the timing channel of an attack scenario where the system is deterministic.

Let $\mathsf{AS} = (\mathsf{S}, \mathsf{E}_{\text{pub}}, c, k)$ be an attack scenario where $\mathsf{S} = (\mathsf{TA}_i)_{i \in I}$ is a deterministic system. We construct the timing channel $\mathsf{TC}(\mathsf{AS})$ of AS using the algorithm given in Table 6.1. The first step of the algorithm computes for each input $i \in I$, the set of its possible observations O_i. Since S is deterministic, $O_i = \{o\}$ is a singleton. All the observations of the system are described by the set $O = \bigcup_{i \in I} O_i$ and taking random variables $\mathcal{I}$ and $\mathcal{O}$ over the input set I and the observations O (resp.), we have the deterministic timing channel $\mathsf{TC}(\mathsf{AS}) : I \times O \mapsto [0,1]$, that for input i and its unique observation o it returns 1, otherwise it returns 0. Notice that our algorithm is independent of our choice of min-leakage to be used as the measure for quantifying leakage.

6.4.3 Probability Measure for Stochastic Timed Automata

To explain the construction for the case of S being stochastic we need to define a probability measure on the runs of stochastic timed automata. We will then use this measure to compute the probabilities of the timing observations of an adversary.

Let $\mathsf{STA} = (\mathsf{TA}, (\mu_\gamma)_{\gamma \in \mathbf{Config}}, (\kappa_\gamma)_{\gamma \in \mathbf{Config}})$ be a stochastic timed automaton, we define a probability measure over the set of $\mathsf{Runs}(\gamma)$ for each $\gamma \in \mathbf{Config}$ as in [Car17, BBB$^+$14]. We start by giving some helpful definitions.

For an edge $e \in \mathsf{E}$ we write $\mathsf{source}(e) = q_s$ for its source location, and $\mathsf{target}(e) = q_t$ for its target location. A *path* $e_1....e_n$ ($n \geq 1$) is a sequence of edges such that for all $i \in \{2, .., n\}$ we have that $\mathsf{source}(e_i) = \mathsf{target}(e_{i-1})$. For a path $\pi = e_1...e_n$, a configuration $\gamma \in \mathbf{Config}$ and a Borel set $\mathcal{C}$ of $\mathbb{R}^n_{\geq 0}$ ($n \geq 1$) (i.e $\mathcal{C} \in \mathcal{B}(\mathbb{R}^n_{\geq 0})$) we define the set of *$\mathcal{C}$-constrained cylinders* of π as

$$\mathsf{Cyl}_{\mathcal{C}}(\gamma, \pi) = \\ \left\{\gamma_0 \xrightarrow{t_1, e_1} \gamma_1 ... \gamma_{n-1} \xrightarrow{t_n, e_n} \gamma_n ... \in \mathsf{Runs}(\gamma) \mid (t_1, .., t_n) \in \mathcal{C}\right\}$$

that is the set of all runs of γ that go through the path $\pi = e_1...e_n$ and the time delays $t_1, ..., t_n$ satisfy the constrain $\mathcal{C}$.

For a path $\pi = e_1...e_n$, a configuration $\gamma \in \mathbf{Config}$ and a Borel set $\mathcal{C}$ of $\mathbb{R}^n_{\geq 0}$ ($n \geq 1$) we define inductively the probability measure P_γ. For the base case where $\pi = e$ we have that

$$\mathsf{P}_\gamma(\mathsf{Cyl}_{\mathcal{C}}(\gamma, e)) = \int_{t \in \mathsf{Int}(\gamma, e)} \kappa_{\gamma+t}(e) \cdot 1_{\mathcal{C}}(t) \mathrm{d}\mu_\gamma(t)$$

where $\gamma + t$ is γ with its valuation having its dense clocks increased by t and $1_{\mathcal{C}} : \mathbb{R}_{\geq 0} \mapsto \{0, 1\}$ is the indicator function defined as

$$1_{\mathcal{C}}(t) = \begin{cases} 1 & \text{if } t \in \mathcal{C} \\ 0 & \text{otherwise} \end{cases}$$

The domain of integration[3] is over all possible delays $t \in \mathsf{Int}(\gamma, e)$ that γ could make by choosing e and would result to a configuration $\gamma + t$. The function which is integrated then is the probability $\kappa_{\gamma+t}(e)$ of choosing e from $\gamma + t$, multiplied by $1_{\mathcal{C}}(t)$, ensuring that t satisfies $\mathcal{C}$.

For the inductive case, where $\pi = e_1...e_n$, we have

$$\mathsf{P}_\gamma(\mathsf{Cyl}_{\mathcal{C}}(\gamma, e_1...e_n)) = \\ \int_{t_1 \in \mathsf{Int}(\gamma, e_1)} \kappa_{\gamma+t_1}(e_1) \cdot \mathsf{P}_{\gamma'}(\mathsf{Cyl}_{\mathcal{C}^{t_1}}(\gamma', e_2...e_n)) \mathrm{d}\mu_\gamma(t_1)$$

where $\gamma \xrightarrow{t_1, e_1} \gamma'$, and

$$\mathcal{C}^{t_1} = \left\{(t_2, ..., t_n) \in \mathbb{R}^{n-1}_{\geq 0} \mid (t_1, ..., t_n) \in \mathcal{C}\right\}$$

[3]Whenever μ_γ is discrete then the integration becomes summation instead.

The explanation is similar to the base case, where now we also integrate over the probability of the constrained cylinder of the remaining path $e_2....e_n$, starting at the resulting configuration γ'.

The following theorem from [Car17, BBB+14] shows that P_γ is a well-defined probability measure.

THEOREM 6.9 *For a stochastic timed automaton* STA *and for each configuration* $\gamma \in$ ***Config****,* P_γ *is a probability measure over* $(\mathsf{Runs}(\gamma), \mathcal{F})$ *where* $\mathcal{F}$ *is the* σ*-algebra generated by the constrained cylinders of* γ*.*

EXAMPLE 6.6 *Consider the stochastic automaton of Example 6.2. For the constraint*

$$\mathcal{C} = \{(t_1, t_2) \in \mathbb{R}^2_{\geq 0} \mid (5 \leq t_1 < 10) \wedge (10 \leq t_1 + t_2 < 15)\}$$

and the path $\pi = e_1 e_3$ *we want to compute the probability* $P_{\gamma_{q_\circ}}(\mathsf{Cyl}_{\mathcal{C}}(\gamma_{q_\circ}, \pi))$ *that is equal to*

$$\int_{t_1 \in \mathsf{Int}(\gamma_{q_\circ}, e_1)} \kappa_{\gamma_{q_\circ}+t_1}(e_1) \cdot \mathsf{P}_{\gamma'}(\mathsf{Cyl}_{\mathcal{C}^{t_1}}(\gamma', e_3))\, d\mu_{\gamma_{q_\circ}}(t_1) \quad (1)$$

Next, for $\gamma_{q_\circ} = \langle q_\circ, [\mathtt{r} \mapsto 0]\rangle$ *we have the discrete probability distribution* $\mu_{\gamma_{q_\circ}}$ *over the all possible delays of* γ*,* $\mathsf{Int}(\gamma_{q_\circ}) = \{5, 7\}$*, defined by the probability mass function* p*, where* $p(5) = \frac{2}{3}$*, and* $p(7) = \frac{1}{3}$*. We also have that* $\mathsf{Int}(\gamma_{q_\circ}, e_1) = \{5\}$ *and for the resulting (after a delay) configurations* $\gamma_{q_\circ} + 5$ *the probability of taking the edge* e_1 *is* $\kappa_{\gamma_\circ+5}(e_1) = 1$*. Therefore (1) is equal to*

$$p(5) \cdot \mathsf{P}_{\gamma'}(\mathsf{Cyl}_{\mathcal{C}^{t_1}}(\gamma', e_3)) \quad (2)$$

and since $t_1 \in \mathsf{Int}(\gamma_{q_\circ}, e_1) = \{5\}$ *we have that*

$$\mathcal{C}^{t_1} = \mathcal{C}^5 = \{t_2 \in \mathbb{R}_{\geq 0} \mid 10 \leq t_2 + 5 < 15\} = [5, 10)$$

Next, for the resulting configuration $\gamma' = \langle q, \mathtt{r} \mapsto 0\rangle$ *we have the continuous uniform probability distribution* $\mu_{\gamma'}$ *over the all possible delays of* γ'*,* $\mathsf{Int}(\gamma') = [5, 10]$*, defined by the probability density function* $f(t) = \frac{1}{5} \cdot 1_{[5,10]}(t)$*. For the resulting (after a delay) configurations* $\gamma' + t$ *he probability of taking the edge* e_3 *is* $\kappa_{\gamma'+t}(e_3) = 1$*. Therefore for* $\mathsf{P}_{\gamma'}(\mathsf{Cyl}_{\mathcal{C}^5}(\gamma', e_3))$ *we have*

$$\begin{aligned}
\mathsf{P}_{\gamma'}(\mathsf{Cyl}_{\mathcal{C}^5}(\gamma', e_3)) &= \int_{t_2 \in \mathsf{Int}(\gamma', e_3)} \kappa_{\gamma'+t_2}(e_3) \cdot 1_{\mathcal{C}^5}(t_2) d\mu_{\gamma'}(t_2) \\
&= \int_{t_2 \in [5,10]} f(t) \cdot 1_{\mathcal{C}^5}(t_2) dt_2 \\
&= \int_{t_2 \in [5,10]} \frac{1}{5} \cdot 1_{[5,10]}(t_2) \cdot 1_{[5,10)}(t_2) dt_2 \\
&= \frac{1}{5} \cdot \int_{t_2 \in [5,10)} dt_2 = 1 \quad (3)
\end{aligned}$$

and thus using (1), (2) and (3) we have

$$\mathsf{P}_{\gamma_{q_\circ}}(\mathsf{Cyl}_{\mathcal{C}}(\gamma_{q_\circ}, e_1 e_3)) = \frac{2}{3} \cdot 1 = \frac{2}{3}$$

6.4.4 Timing Channels of Stochastic Systems

We now show how one can construct the timing channel of an attack scenario where the system is stochastic.

For an attack scenario $\mathsf{AS} = (\mathsf{S}, \mathsf{E}_{\text{pub}}, c, k)$ of a stochastic system

$$\mathsf{S} = (\mathsf{TA}_i, (\mu_\gamma)^i_{\gamma \in \textbf{Config}} (\kappa_\gamma)^i_{\gamma \in \textbf{Config}})_{i \in I}$$

we construct the corresponding timing channel using Table 6.1. The construction follows the same logic as the one for deterministic systems, where for each input $i \in I$ we need to enumerate all the possible observations (**Step 1** of Table 6.1) of the adversary and then compute its probabilities (**Step 2** of Table 6.1).

For deterministic systems, this process is straightforward, since each input is associated with exactly one observation, and consequently this observation has probability 1. However, this is not the case for stochastic systems.

Starting with **Step 1**, for an input $i \in I$, the set of possible observations $O_i = \{\mathsf{view}_c(\rho) \mid \rho \in \mathsf{Runs}(\mathsf{TA}_i)\}$ could turn to be infinite. To deal with this case (when needed) we assume that the clock c of the adversary has a limit (i.e this models that the clock has finite capacity). Now let $\mathfrak{g}$ be the grain of c and $l = m.\mathfrak{g}$ (m is a natural number) its capacity. The modified clock $c_l : \mathbb{R}_{\geq 0} \mapsto \mathbb{N}$ is given by

$$c_l(t) = \min \left\{ l, \left\lfloor \frac{t}{\mathfrak{g}} \right\rfloor \cdot \mathfrak{g} \right\}$$

The modified clock c_l now behaves as the clock c for time points $t < l$, whereas for values greater or equal to l its value becomes constant. Here notice that the algorithm from Table 6.1 remains unchanged, but only the definition of the clock changes, so we can bound the set O_i.

Next, for **Step 2** we compute the probability of an observation o. First, we have that the runs which can result in the observation o, are described by the preimage $\mathsf{view}_c^{-1}(o)$, and consequently the probability of the observation o is equal to $\mathsf{P}_{\gamma_{q^i_\circ}}(\mathsf{view}_c^{-1}(o))$. To show that $\mathsf{view}_c^{-1}(o)$ is measurable our goal is to express it as a union of disjoint constrained cylinders. This will also give us a more algorithmic approach for computing the probability of o. We start by providing some auxiliary sets and operators.

For the rest, we fix an input $i \in I$ and let TA be its corresponding timed automaton where we omit the subscript i. Let

$$\textbf{Paths} = \bigcup_{i=k}^{\infty} \{e_1...e_i \mid |\{j \mid e_j \in \mathsf{E}_{\text{pub}}\}| = k \wedge \\ \mathsf{source}(e_1) = q_\circ \wedge e_i \in \mathsf{E}_{\text{pub}}\}$$

be the set of paths that start at the initial location $q_\circ$, contain exactly k public edges and the last edge is public. Each path in this set represents one or more (prefixes of) runs that could result to a k-sequence of timing observations.

Example 6.7 *For the attack scenario of Example 6.5, we have that $k = 2$ and* ***Paths*** $= \{e_1e_3, e_2e_3\}$

Now for a sequence of observations $o = (z_1, ..., z_k) \in O_i$ and a path $\pi = e_1, ..., e_n \in$ **Paths** we want to specify a constraint that describes the set of possible delays that could result to this particular sequence of observations taking this path. We thus define

$$\mathcal{C}_{e_1...e_n}(z_1, ..., z_k) = \\ \bigcap_{i=1}^{k} \{(t_1, ..., t_n) \in \mathbb{R}^n_{\geq 0} \mid \{j \mid e_j \in \mathsf{E}_{\text{pub}}\} = \{j_1..., j_k\} \\ \Rightarrow\ t_1 + ... + t_{j_i} \in c^{-1}(z_i)\}$$

and notice here that $c^{-1}(z)$ (for $z \in \{z_1, ..., z_k\}$) is an interval.[4]

Example 6.8 *For the path $\pi = e_1e_3 \in$* ***Paths*** *from Example 6.7 and the observation $o = (5, 10) \in O$ we have the constraint*

$$\mathcal{C}_\pi(o) = \{(t_1, t_2) \in \mathbb{R}^2_{\geq 0} \mid t_1 \in c^{-1}(5)\} \cap \\ \{(t_1, t_2) \in \mathbb{R}^2_{\geq 0} \mid t_1 + t_2 \in c^{-1}(10)\}$$

and since $c^{-1}(5) = [5, 10)$ and $c^{-1}(10) = [10, 15)$ we have that

$$\mathcal{C}_\pi(o) = \{(t_1, t_2) \in \mathbb{R}^2_{\geq 0} \mid (5 \leq t_1 < 10) \wedge \\ (10 \leq t_1 + t_2 < 15)\}$$

Finally, this allows us to express $\mathsf{view}_c^{-1}(o)$ as a union of disjoint constrained cylinders as

$$\mathsf{view}_c^{-1}(o) = \bigcup_{\pi \in \textbf{Paths}} \mathsf{Cyl}_{\mathcal{C}_\pi(o)}(\gamma_{q_\circ}, \pi)$$

[4]The same holds whenever we have a clock c_l with limit l.

and thus the probability of the observation $o \in O_i$ is

$$\mathsf{P}_{\gamma_{q_\circ}}(\mathsf{view}_c^{-1}(o)) = \sum_{\pi \in \textbf{Paths}} \mathsf{P}_{\gamma_{q_\circ}}(\mathsf{Cyl}_{\mathcal{C}_\pi(o)}(\gamma_{q_\circ}, \pi))$$

The next example illustrates the construction of a timing channel for the case of the system being stochastic.

EXAMPLE 6.9 *We will now compute the timing channel of the attack scenario from Example 6.5.*

*The set of possible observations of the adversary is $O = \{(5,10),(5,15)\} = O_{i_1} = O_{i_2}$ and from Example 6.7, we have that **Paths**$=\{e_1e_3, e_2e_3\}$.*

Next, let $\pi_1 = e_1e_3$, and $\pi_2 = e_2e_3$. For the input i_1 and the observation $o = (5,10)$ we have that

$$\begin{aligned}\mathsf{view}_c^{-1}((5,10)) = {} & \mathsf{Cyl}_{\mathcal{C}_{\pi_1}(5,10)}(\gamma_{q_\circ^{i_1}}, \pi_1) \cup \\ & \mathsf{Cyl}_{\mathcal{C}_{\pi_2}(5,10)}(\gamma_{q_\circ^{i_1}}, \pi_2)\end{aligned}$$

and thus

$$\begin{aligned}\mathsf{P}_{\gamma_{q_\circ^{i_1}}}(\mathsf{view}_c^{-1}((5,10))) = {} & \mathsf{P}_{\gamma_{q_\circ^{i_1}}}(\mathsf{Cyl}_{\mathcal{C}_{\pi_1}(5,10)}(\gamma_{q_\circ^{i_1}}, \pi_1)) + \\ & \mathsf{P}_{\gamma_{q_\circ^{i_1}}}(\mathsf{Cyl}_{\mathcal{C}_{\pi_2}(5,10)}(\gamma_{q_\circ^{i_1}}, \pi_2))\end{aligned}$$

Note that in Example 6.6, we calculated the probability $\mathsf{P}_{\gamma_{q_\circ^{i_1}}}(\mathsf{Cyl}_{\mathcal{C}_{\pi_1}(5,10)}(\gamma_{q_\circ^{i_1}}, \pi_1)) = \frac{2}{3}$ and working similarly we can show that $\mathsf{P}_{\gamma_{q_\circ^{i_1}}}(\mathsf{Cyl}_{\mathcal{C}_{\pi_2}(5,10)}(\gamma_{q_\circ^{i_1}}, \pi_2)) = \frac{1}{5}$ and therefore we get that

$$\mathsf{P}_{\gamma_{q_\circ^{i_1}}}(\mathsf{view}_c^{-1}((5,10))) = \frac{2}{3} + \frac{1}{5} = \frac{13}{15}$$

We work similarly for the observation $o = (5,15)$ and we get

$$\begin{aligned}\mathsf{P}_{\gamma_{q_\circ^{i_1}}}(\mathsf{view}_c^{-1}((5,15))) = {} & \mathsf{P}_{\gamma_{q_\circ^{i_1}}}(\mathsf{Cyl}_{\mathcal{C}_{\pi_1}(5,15)}(\gamma_{q_\circ^{i_1}}, \pi_1)) + \\ & \mathsf{P}_{\gamma_{q_\circ^{i_1}}}(\mathsf{Cyl}_{\mathcal{C}_{\pi_2}(5,15)}(\gamma_{q_\circ^{i_1}}, \pi_2)) \\ = {} & 0 + \frac{2}{15} = \frac{2}{15}\end{aligned}$$

We repeat the process for the input i_2, and we obtain the following timing channel

$$\mathsf{TC}(\mathsf{AS})(i,o) = \begin{cases} \frac{13}{15} & \textit{if } i = i_1 \textit{ and } o = (5,10) \\[2ex] \frac{2}{15} & \textit{if } i = i_1 \textit{ and } o = (5,15) \\[2ex] \frac{8}{10} & \textit{if } i = i_2 \textit{ and } o = (5,10) \\[2ex] \frac{2}{10} & \textit{otherwise} \end{cases}$$

6.5 Analysis of Timing Channels in Deterministic Systems

In this section, we start by analyzing the relationship between clock grain and leakage for deterministic systems. Next, we present timing techniques that have been used to bypass a low-resolution clock, we present a new timing technique, and we show how those techniques can be modelled in our framework. We finish by showing a result on the hierarchy of those techniques in terms of how much information can be extracted from the adversary.

6.5.1 Relating Clock Grain and Leakage

In our first result, we show that, contrary to popular belief a coarse-grained clock might leak more information than a fine-grained clock.

PROPOSITION 6.10 *There exist deterministic system* S *and attack scenarios* AS_1, AS_2 *of* S *with clocks* c_1, c_2 *resp., and grains* $\mathfrak{g}_1$, $\mathfrak{g}_2$ *with* $\mathfrak{g}_1 < \mathfrak{g}_2$ *and* $\mathsf{TC}(\mathsf{AS}_1) \preceq \mathsf{TC}(\mathsf{AS}_2)$.

Note that Proposition 6.10 does not talk about the well-known bypassing techniques [SWT01, KS16b, SMGM17, Wra91] that have been used to side-step the defense of a coarse-grained clock; instead it shows that the security offered by a coarse-grained clock could be worse than the one offered by a fine-grained clock, even when bypassing techniques are not in use.

Proposition 6.10 follows from the following example

EXAMPLE 6.10 *Consider a deterministic system* S*, whose input set is* $I = \{i_1, i_2\}$ *and the system is given by two automata* TA_{i_1} *and* TA_{i_2} *who both have a single edge leaving their initial location* $e_1 = (q_\circ, \mathtt{r} = 2 \rightarrow, q)$ *(for* TA_{i_1}*), and* $e_2 = (q'_\circ, \mathtt{r} = 3 \rightarrow, q')$ *(for* TA_{i_2}*) controlled by a dense clock* $\mathtt{r}$.

For TA_{i_1} *we have the run* $\rho_1 = \gamma_{q_\circ} \xrightarrow{2,e_1} \gamma_1 ...$ *and for* TA_{i_2} *we have the run* $\rho_2 = \gamma_{q'_\circ} \xrightarrow{3,e_2} \gamma'_1 ....$ *Next consider the two attack scenarios* $\mathsf{AS}_1 = (S, \{e_1, e_2\}, c_1, 1)$ *and* $\mathsf{AS}_2 = (\mathsf{S}, \{e_1, e_2\}, c_2, 1)$ *where the edges of interest are observable, the clock* c_1 *has grain* $\mathfrak{g}_1 = 2$*, the clock* c_2 *has grain* $\mathfrak{g}_2 = 3$*, and the adversary makes one timing observation. Following the algorithm from Table 6.1, for* AS_1 *we have* $O_{i_1} = \{\mathsf{view}_{c_1}(\rho_1)\} = \{c_1(2)\} = \{2\}$ *and* $O_{i_2} = \{\mathsf{view}_{c_1}(\rho_2)\} = \{c_1(3)\} = \{2\}$*, whereas for* AS_2 *we have* $O_{i_1} = \{\mathsf{view}_{c_2}(\rho_1)\} = \{c_2(2)\} = \{0\}$ *and* $O_{i_2} = \{\mathsf{view}_{c_2}(\rho_2)\} = \{c_2(3)\} = \{3\}$ *and thus we get the timing channels*

$$\mathsf{TC}(\mathsf{AS}_1)(i,o) = 1 \quad \mathsf{TC}(\mathsf{AS}_2)(i,o) = \begin{cases} 1 & \textit{if } i = i_1 \\ & \textit{and } o = 0 \\ 1 & \textit{if } i = i_2 \\ & \textit{and } o = 3 \\ 0 & \textit{otherwise} \end{cases}$$

We then have that $\equiv_{\mathsf{TC}(\mathsf{AS}_1)} = \top$*, whereas* $\equiv_{\mathsf{TC}(\mathsf{AS}_2)} = \bot$*, and thus* $\mathsf{TC}(\mathsf{AS}_1) \sqsubseteq \mathsf{TC}(\mathsf{AS}_2)$. *Next, using Theorem 6.8 we get that* $\mathsf{TC}(\mathsf{AS}_1) \preceq \mathsf{TC}(\mathsf{AS}_2)$ *showing that the attack scenario where the clock has grain* $\mathfrak{g}_2 = 3$ *leaks more than the scenario where the clock has grain* $\mathfrak{g}_1 = 2$.

Although, we showed that, in general, increasing the grain of the clock does not increase security, with our next theorem we provide sufficient conditions for when this actually happens.

THEOREM 6.11 ***(Multiple-$\mathfrak{g}$ security.)*** *Let* $\mathsf{AS}_1 = (\mathsf{S}, \mathsf{E}_{pub}, c_1, k)$ *and* $\mathsf{AS}_2 = (\mathsf{S}, \mathsf{E}_{pub}, c_2, k)$ *be two attack scenarios such that* S *is deterministic and the clocks* c_1, c_2 *have grains* $\mathfrak{g}_1$, $\mathfrak{g}_2$ *(resp.) and* $\mathfrak{g}_1$ *is a multiple of* $\mathfrak{g}_2$. *We then have that*

$$\mathsf{TC}(\mathsf{AS}_1) \preceq \mathsf{TC}(\mathsf{AS}_2)$$

In particular, Theorem 6.11 shows that whenever the system is deterministic and we increase the grain of the clock to a multiple of it, the new low-resolution clock gives better (or at least the same) security.

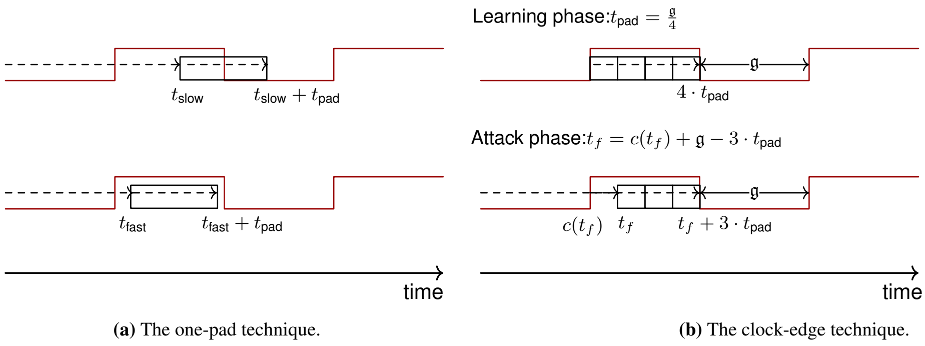

t_{slow}
$t_{\text{slow}} + t_{\text{pad}}$
t_{fast}
$t_{\text{fast}} + t_{\text{pad}}$
time
Learning phase:$t_{\text{pad}} = \frac{\mathfrak{g}}{4}$
$\mathfrak{g}$
$4 \cdot t_{\text{pad}}$
Attack phase:$t_f = c(t_f) + \mathfrak{g} - 3 \cdot t_{\text{pad}}$
$c(t_f)$
t_f
$t_f + 3 \cdot t_{\text{pad}}$
time
(a) The one-pad technique.
(b) The clock-edge technique.

Figure 6.5: Padding timing techniques.

6.5.2 Timing Techniques

Several timing techniques have successfully side-stepped the defence provided by a coarse-grained clock, by building their own fine-grained clocks from primitives such as coarse-grained clocks and simple counter processes [SWT01, KS16b, SMGM17]. We now explain how techniques such as the one-pad and the clock-edge [SMGM17, KS16b] work, and we present a new timing technique called the co-prime. We study those techniques in a setting as in [SMGM17, KS16b], where the adversary tries to measure the timing of a function f that is sent to the victim in a piece of malicious code where he performs his timing technique.

The One-Pad Technique In many cases the adversary wants only to distinguish between two different executions of f, t_{slow} and t_{fast}, where t_{fast} is smaller than t_{slow}. Using the *one-pad* technique the adversary exploits the fact that the time between two clock-edges is constant and equal to $\mathfrak{g}$. He then chooses to perform a *constant time* operation called padding.

If now t_{pad} is the time of the padding operation, the padding is chosen in such a way such that the short duration t_{slow} plus the duration of the padding t_{pad} always crosses the next clock-edge i.e $t_{\text{slow}} + t_{\text{pad}} \geq c(t_{\text{slow}}) + \mathfrak{g}$, while $t_{\text{fast}} + t_{\text{pad}} < c(t_{\text{fast}}) + \mathfrak{g}$.

After the end of the padding operation, if the value of the adversary's clock is above the next clock-edge, he infers that the slow event has happened, otherwise it is the case of the fast event. Figure 6.5 (a) depicts a scenario of the one-pad technique.

The Clock-Edge Technique The one-pad technique is good enough for distinguishing between two different execution times, however, sometimes an adversary requires a mechanism that gives more precise measurements. For those cases, the *clock-edge* technique can be used. The fact that a clock c with grain $\mathfrak{g}$ is being increased with a constant rate i.e every $\mathfrak{g}$, gives the attacker the ability to express the duration of a sequence of his operations as a portion to $\mathfrak{g}$. In particular, similarly to the one-pad technique the adversary here adds a padding for one or more times.

The technique begins with a *learning phase* (Figure 6.5 (b) top), where the adversary performs his padding operation between two consecutive clock-edges. Assuming that a number of m operations have occurred between the two edges, and using that the duration of m padding operations is equal to $\mathfrak{g}$, the attacker derives that the duration of his padding operation is $t_{\text{pad}} = \frac{\mathfrak{g}}{m}$.

The *attack phase* (Figure 6.5 (b) bottom) of the technique then begins with the adversary aligning the operation f that he wants to measure to a clock-edge and right after the execution of f completes, he inspects the value of his clock $c(t_f)$. Immediately after, he starts performing his padding operator until he observes the next clock-edge. If at this point he observes n paddings the duration t_f of f can be approximated by the number $c(t_f) + \mathfrak{g} - n \cdot t_{\text{pad}} \approx t_f$.[5]

The Co-Prime Technique Looking at the one-pad and the clock-edge technique one could think of the following questions: Do we always need a padding operation with duration less than $\mathfrak{g}$ in order to perform fine-grained measurements? What happens if we do not stop at the next clock-edge and we continue performing the padding operator for a couple of more times?

For the co-prime technique, the adversary may not necessarily use a padding with timing less than the grain of the clock, and he may also not stop at the next clock-edge. In particular, in the co-prime technique, the adversary performs a padding operation for $\mathfrak{g}$ (the grain of the clock) times, and the timing of his padding t_{pad} is co-prime with $\mathfrak{g}$. Why this technique works becomes clear in the next two subsections.

6.5.3 Modelling Timing Techniques

We use the algorithm in Table 6.2 to model the essential aspects of the timing techniques: one-pad, clock-edge and co-prime.

[5]In the clock-edge techniques presented in [KS16b, SMGM17] the padding operation corresponds to the increment of a counter.

$\mathsf{TA}(t_f, t_{pad}, m)$	$=$	let $q_\circ$, q_1, ..., q_{m+1} be fresh nodes	
		and x a fresh dense clock	
		$\mathsf{Q} =$	$\{q_\circ, q_1, ..., q_{m+1}\}$
		$\mathsf{E} =$	$\{(q_\circ, r = t_f \rightarrow r, q_1),$
			$(q_1, r = t_{\text{pad}} \rightarrow r, q_2), ...,$
			$(q_m, r = t_{\text{pad}} \rightarrow r, q_{m+1})\}$
		$\mathsf{I} =$	$[q_\circ \mapsto \mathsf{tt}][q_1 \mapsto \mathsf{tt}]...[q_{m+1} \mapsto \mathsf{tt}]$
		in $(\mathsf{Q}, \mathsf{E}, \mathsf{I}, q_\circ)$	

Table 6.2: Algorithm for constructing padding timing techniques.

Recall that we assume an adversary who tries to measure the timing of a deterministic function f that is sent to the victim in a piece of malicious code where he performs his timing technique. The timing of f varies depending on the internal state of the victim. The algorithm in Table 6.2 takes as an input the timing of the operation t_f, the timing of the padding t_{pad} and the number m of padding operations the adversary performs.

The resulting timed automaton consists of a single dense clock r and $m + 2$ locations which are being constructed in the third line. Line four constructs the edges of the automaton. The first edge corresponds to executing the operation f and thus we delay exactly t_f time expressed by the guard $r = t_f$. The next m edges correspond to the execution of the padding and similarly as for the first edge we now wait for exactly t_{pad} time. All the invariants are set to true.

Let now $\mathfrak{g}$ be the granularity of the adversary's clock c, and $t_{\text{pad}} \in \mathbb{N}$ be the execution time of his padding operator. Assume also that the function f the adversary wants to measures takes an input from the victim's set $I = \{i_1, .., i_n\}$ and let $t_{i_1}, ..., t_{i_n}$ be the execution times of f on the inputs $i_1, ..., i_n$ (resp.).

The attack scenario of the one-pad technique can then be described by

$$\mathsf{AS}_{\text{1-pad}} = ((\mathsf{TA}(t_i, t_{\text{pad}}, 1))_{i \in I}, \mathsf{E}, c, 2)$$

where E are the edges of the system and they are observable, and the adversary makes $k = 2$ observations (one before and one after its padding operation).

Similarly for the (attack phase of the) clock-edge technique we have

$$\mathsf{AS}_{\text{clock-edge}} = ((\mathsf{TA}(t_i, t_{\text{pad}}, m))_{i \in I}, \mathsf{E}, c, m + 1)$$

where

$$m = \min\ \{n \in \mathbb{N} \mid n \cdot t_{\text{pad}} \geq \mathfrak{g}\}$$

Again E are the edges of the system and they are all observable. The number m of paddings is ensuring that independently of f's timing, the padding will cross the next clock-edge.

Finally, if $\mathfrak{g}$ is co-prime with t_{pad}, the attack scenario of the co-prime technique is

$$\mathsf{AS}_{\text{co-prime}} = ((\mathsf{TA}(t_i, t_{\text{pad}}, g))_{i \in I}, \mathsf{E}, c, \mathfrak{g} + 1)$$

6.5.4 A Hierarchy of Timing Techniques

We now compare the power of the timing techniques, one-pad, clock-edge and co-prime in terms of how much information the adversary can extract using them, and we explain in more details why the co-prime technique works. We start with an example that illustrates the construction (Table 6.2) of the attack scenarios of the clock-edge and the co-prime technique and shows that the co-prime technique can distinguish more.

Example 6.11 *Assume that we want to distinguish between two timing behaviours of an operation f that takes the inputs i_1 and i_2 and runs for time $t_{i_1} = 8$ and $t_{i_2} = 9$ respectively. Assume also that we have a clock c with grain $\mathfrak{g} = 10$ and a padding with timing t_{pad}=2.*

For the clock-edge technique we need to add our padding for

$$\begin{array}{rcl} m & = & \mathit{min}\,\{n \in \mathbb{N} \mid n \cdot t_{pad} \geq \mathfrak{g}\} \\ & = & \mathit{min}\,\{n \in \mathbb{N} \mid n \cdot 2 \geq 10\} \\ & = & 5 \end{array}$$

Using the algorithm of Table 6.2 we get the system $\mathsf{S} = (\mathsf{TA}(t_i, 2, 5))_{i \in I}$ and for each $i \in I$ we have a timed automaton $\mathsf{TA}(t_i, 2, 5) = (\mathsf{Q}, \mathsf{E}, \mathsf{I}, q_\circ)$ where $\mathsf{Q} = \{q_\circ, q_1, ..., q_6\}$ and E contains the edges $e_1 = (q_\circ, r = t_i \rightarrow r, q_1)$, $e_2 = (q_1, r = 2 \rightarrow r, q_2)$,...,$e_6 = (q_5, r = 2 \rightarrow r, q_6)$, and $\mathsf{I} = \lambda q.\mathsf{tt}$.

We then have two runs

$$\rho_1 = \gamma_1 \xrightarrow{8, e_1} \gamma_2 \xrightarrow{2, e_2} \ldots \xrightarrow{2, e_6} \gamma_7$$

and

$$\rho_2 = \gamma_1' \xrightarrow{9, e_1'} \gamma_2' \xrightarrow{2, e_2'} \ldots \xrightarrow{2, e_6'} \gamma_7'$$

for i_1 and i_2 respectively.

The view of the adversary on ρ_1 is

$$\begin{aligned}\mathsf{view}_c(\rho_1) &= (c(8), c(10), c(12), c(14), c(16), c(18)) \\ &= (0, 10, 10, 10, 10, 10)\end{aligned}$$

and for ρ_2 we have

$$\begin{aligned}\mathsf{view}_c(\rho_2) &= (c(9), c(11), c(13), c(15), c(17), c(19)) \\ &= (0, 10, 10, 10, 10, 10)\end{aligned}$$

Therefore the runs are indistinguishable to the adversary.

Consider now the same scenario where the padding of the adversary has duration t_{pad}=19 which is strictly greater *than the grain $\mathfrak{g} = 10$ of the clock, and let $\mathfrak{g}$ be the number of paddings that we want to add. We will construct the attack scenario of the co-prime technique.*

Using Table 6.2 we get the system $\mathsf{S} = (\mathsf{TA}(t_i, 19, 10))_{i \in I}$ and for each $i \in I$ we have a timed automaton $\mathsf{TA}(t_i, 19, 10) = (\mathsf{Q}, q_\circ, \mathsf{E}, \mathsf{I})$ where $\mathsf{Q} = \{q_\circ, q_1, ..., q_{11}\}$ and E contains the edges $e_1 = (q_\circ, r = t_i \rightarrow r, q_1)$, $e_2 = (q_1, r = 19 \rightarrow r, q_2)$,...,$e_{11} = (q_{10}, r = 19 \rightarrow r, q_{11})$, and $\mathsf{I} = \lambda q.\mathsf{tt}$.

We then have two runs

$$\rho_1 = \gamma_1 \xrightarrow{8,e_1} \gamma_2 \xrightarrow{19,e_2} ... \xrightarrow{19,e_{11}} \gamma_{12}$$

and

$$\rho_{i_2} = \gamma_1' \xrightarrow{9,e_1'} \gamma_2' \xrightarrow{19,e_2'} ... \xrightarrow{19,e_{11}'} \gamma_{12}'$$

for i_1 and i_2 respectively.

The view of the adversary on ρ_1 is

$$\begin{aligned}\mathsf{view}_c(\rho_1) &= (c(8), c(27), c(46), c(65), ..., c(179), c(198)) \\ &= \mathit{(0,20,40,60,...,170,190)}\end{aligned}$$

and for ρ_2 we have

$$\begin{aligned}\mathsf{view}_c(\rho_2) &= (c(9), c(28), c(47), c(66), ..., c(180), c(199)) \\ &= \mathit{(0,20,40,60...,180,190)}\end{aligned}$$

and thus the two runs become distinguishable at the 10th observation, because for ρ_1 the adversary observes 170 and for ρ_2 he observes 180.

To understand better why the co-prime technique works observe that in general a timing technique is distinguishing two timings t_1, t_2, when observing differences in the sequences

$$(c(t_1), c(t_1 + t_{\text{pad}}), ..., c(t_1 + m \cdot t_{\text{pad}}))$$

and

$$(c(t_2), c(t_2 + t_{\text{pad}}), ..., c(t_2 + m \cdot t_{\text{pad}}))$$

The question therefore is how to choose the appropriate number m of paddings for making the two sequences distinguishable.

However, Fact 9 shows that the two sequences are distinguishable if and only if

$$(c(t_1), c(t_1 + (t_{\text{pad}} \bmod \mathfrak{g})), ..., c(t_1 + (m \cdot t_{\text{pad}} \bmod \mathfrak{g})))$$

and

$$(c(t_2), c(t_2 + (t_{\text{pad}} \bmod \mathfrak{g})), ..., c(t_2 + (m \cdot t_{\text{pad}} \bmod \mathfrak{g})))$$

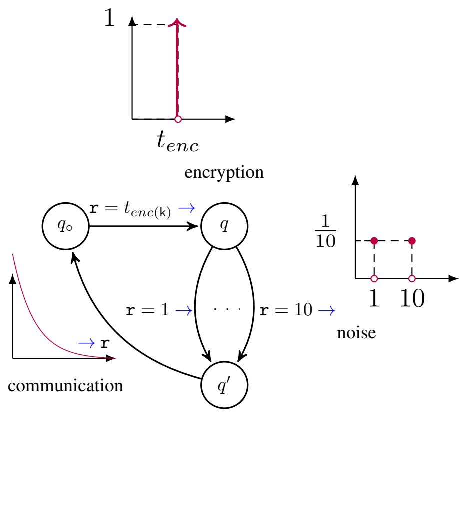
are different.

The co-prime technique exploits this fact and rephrases the question of the padding techniques to: (1) what padding is needed, and (2) how many times I need to add it so by the end of the timing technique the padding times generate the entire $\mathbb{Z}_{\mathfrak{g}}$ set (answer: (1) t_{pad} needs to be co-prime with $\mathfrak{g}$, and (2) it needs to be added $\mathfrak{g}$ times).

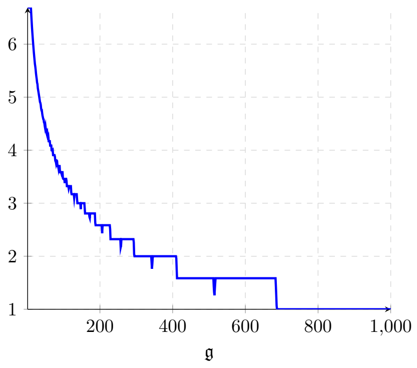
We finish by showing that the timing-techniques of the one-pad, clock-edge and the co-prime form a strict hierarchy in terms of the amount of information the adversary can extract. In particular, we show that the co-prime technique achieves the most information leakage among the other techniques, whereas the one-pad achieves the least leakage.

THEOREM 6.12 *Let f be a function that runs on the input set I and its timing behaviour is described by the family $(t_i)_{i \in I}$. Let also c be a clock with grain $\mathfrak{g}$, a padding with time t_{pad}, and $\mathsf{AS}_{\textit{1-pad}}$, and $\mathsf{AS}_{\textit{clock-edge}}$ the corresponding attack scenarios of the one-pad and the clock-edge techniques. Consider also another padding with duration t'_{pad} that is co-prime with $\mathfrak{g}$, and let $\mathsf{AS}_{\textit{co-prime}}$ be the corresponding co-prime attack scenario. We then have that*

$$\mathsf{TC}(\mathsf{AS}_{\textit{1-pad}}) \preceq \mathsf{TC}(\mathsf{AS}_{\textit{clock-edge}}) \preceq \mathsf{TC}(\mathsf{AS}_{\textit{co-prime}})$$



1
t_{enc}
encryption
$q_\circ$
$\mathtt{r} = t_{enc(\mathsf{k})} \rightarrow$
q
$\mathtt{r} = 1 \rightarrow$
$\cdots$
$\mathtt{r} = 10 \rightarrow$
q'
$\rightarrow \mathtt{r}$
communication
$\frac{1}{10}$
1
10
noise



(a) The stochastic timed automaton of the sensor when it operates on key k. **(b)** The effect of increasing the grain $\mathfrak{g}$ on the min-capacity.

Figure 6.6: Case Study:RSA

6.6 Analysis of Timing Channels in Stochastic Systems: A Case Study

In this section, we analyse timing channels of stochastic systems. In particular, we perform a case study, based on the Example 2.11. It consists of two parts. The first part is the modelling of our case study as a system of timed automata. In the second part, we consider different adversaries (with respect to their clock), we compute their timing channels, and we derive our insights about the relation between clock grain and leakage in stochastic systems.

6.6.1 Modelling the Case Study

We now recall the Example 2.11, and we give more details about it, while we leave some calculations needed in Appendix E. We consider a scenario of a distributed system that consists of a sensor and a controller. We are interested in modelling the behaviour of the sensor.

In particular, the sensor constantly computes some data and communicates it to the controller. For ensuring data integrity, the sensor always encrypts (signs) the data with his RSA private key. The RSA encryption is implemented using the modular exponentiation algorithm which computes $\mathsf{x}^{\mathsf{k}} \bmod \mathsf{n}$ for the secret key k, some data x and the constant modulus n. The implementation is the same as the one from Example 2.3 and it is given by the following piece of code

```
m := (1 * 1) mod n;
for (j = 0; j < len(k); j++) {
    m := (m * m) mod n;
    if (k[j] == 1) then
      m := (m * x) mod n;
}
```

Here the secret bits of the key are stored in the array $\mathsf{k}[]$. Due to the conditional execution of the modular multiplication operation $\mathsf{m} = \mathsf{m} * \mathsf{x} \bmod \mathsf{n}$ the running time of this program reveals information about the entries of k.

To decrease the correlation between the encryption time and the secret bits of the key, the sensor adds noise to the encryption time by delaying for some additional period after each encryption, and then it communicates the data to the controller.

In our model, we assume that the sensor performs each modular multiplication in 1 time unit. We also assume that the size of the secret key k is 1024-bits, and thus the timed needed for one encryption is

$$t_{enc(\mathsf{k})} = 1025 + \mathsf{Ham}(\mathsf{k})$$

where $\mathsf{Ham}(\mathsf{k})$ is the Hamming weight of k (i.e the number of non-zero bits).

For the noise added by the sensor, we assume that the sensor chooses randomly to wait for $t \in \{1, 2, ..., 10\}$, with respect to a uniform distribution.

We model the timing behaviour of the sensor as a stochastic timed system with input space the 2^{1024} keys and for each key, we have a stochastic timed automaton as depicted in Figure 6.6 (a).

The automaton consists of three locations and one dense clock x that is used to control the transitions between them. Starting at the initial location $q_\circ$, the automaton performs an encryption with respect to a Dirac's distribution on the time point $t_{enc(\mathsf{k})}$, modeling in that way that an encryption takes exactly $t_{enc(\mathsf{k})}$ time units. Next, it moves to location q, where we have 10 different edges leaving q, one for each possible delay, modelling the additional noise added by the sensor. A delay is chosen uniformly and the automaton moves to location q'. At location q' the automaton communicates the message to the controller with respect to an exponential distribution of parameter $\lambda = 6$ time units.

Finally, on the side of the controller, we assume an adversary who runs malicious code that uses the clock of the controller and measures the time needed for the sensor to send its data, trying to infer bits of the sensor's private key.

6.6.2 Analysing the Leakage in the Case Study

Using Table 6.1, we constructed the timing channel for 1000 different attack scenarios (for different clocks), where we have as observable edges the ones that model the communication of the message. We assumed an adversary that performs one timing observation and uses a clock with grain $\mathfrak{g} = 1, 2, ..., 1000$ and with some limit $l > 15000$. The details of the construction can be found in the Appendix E.

We then computed the min-capacity for each one of those channels. The graph in Figure 6.6 (b) shows the effect of increasing the grain of the clock on the information leaked by the channel. The maximum leakage is around 6.7-bits for grain $\mathfrak{g} = 1$, whereas we have 1-bit leakage for the attack scenarios where the grain is above 678. We can also see from the graph that increasing the grain of the clock does not always give us less information leakage. In particular, for $\mathfrak{g} = 514$, we have around 1.43-bits leaked, whereas for $\mathfrak{g} = 520$ we have around 1.58-bits, which leads to the following proposition

PROPOSITION 6.13 *There exists stochastic system* S *and attack scenarios* AS_1, AS_2 *of* S *with clocks* c_1, c_2 *resp., and grains* $\mathfrak{g}_1$, $\mathfrak{g}_2$ *with* $\mathfrak{g}_1 < \mathfrak{g}_2$, *and*

$$\mathsf{C}_{min}(\mathsf{TC}(\mathsf{AS}_1)) < \mathsf{C}_{min}(\mathsf{TC}(\mathsf{AS}_2))$$

Proposition 6.13 shows also for the case of stochastic systems that the security offered by a coarse-grained clock could be worse than the one offered by a fine-grained clock, even when bypassing timing techniques are not used.

Finally, our experiment shows that increasing the grain of the clock to a multiple of it results to a channel with less (or equal) leakage, however a proof that this holds for general stochastic systems (i.e Theorem 6.11 generalizes to stochastic systems) is still elusive.

6.7 Related Work

There is an extensive work on formally quantifying and providing bounds on the leakage of timing channels for cryptographic implementations [CRS83, KD09, KB07, BK15], remote network adversaries [ZAM11, GH02] and language-based settings [PHW08, MKP$^+$18, DFK$^+$13]. The main novelty of our approach compared to those is the modelling of coarse-grained clock adversaries, and the novel algorithms that we give for constructing timing channels for systems of timed automata.

Clocks of certain granularity and their defence power have been studied a lot in practice using empirical ways. Schwarz et al. [SMGM17] provided a wide range of techniques that can be used to build fine-grained clocks in Javascript, including similar techniques to the one-pad and the clock-edge. Wei-Ming Hu [Hu91, Hu92] proposed the concept of fuzzy time. Instead of increasing the grain of the clock, fuzzy time modifies its functionality by randomly changing its grain within a certain period. Vattikonda et al. [VDS11] proposed fuzzy time for mitigating timing channels in hypervisors, and also evaluated the impact of this countermeasure on the usability of the system. Fuzzy time has also been proposed and implemented in Firefox by Kohlbrenner et al. [KS16b] for defeating timing channels in browsers. They also showed that this mitigation is effective against timing techniques such as the clock-edge.

Mantel et al. [MS07] proposed an information-theoretic framework for comparing the effectiveness of different countermeasures on the bandwidth of interrupt-related channels, that is a special case of timing channels. In their analysis, they include the countermeasure of coarse-grained clocks and fuzzy time. For coarse-grained clocks, they perform a case-study where they show how increasing the grain of the clock reduces the capacity of the channel. Our approach is more general, while we also showed that increasing the grain of a clock might result in more leakage, and we provided formal proofs for when this is not the case.

6.8 Conclusions

We performed the first principled information-flow analysis of timing leaks w.r.t. adversaries with clocks of reduced resolution, where we derived novel insights into the effectiveness of existing attacks and countermeasures.

In particular, we introduced a model of timed automata systems which is general enough to cater for scenarios where the victim's timing behaviour is stochastic or deterministic, and a model of adversary that is parametric on the clock's granularity and the number of timing observations.

We provided novel algorithms for transforming such a model into an information-theoretic channel, allowing us to measure the leakage conveyed by it using existing techniques from quantitative information-flow.

Based on that, we showed that a coarse-grained clock might leak more than a fine-grained clock, and we provided sufficient conditions for when one can achieve better security by increasing the grain of the clock. For the techniques that have bypassed this countermeasure, we showed that they form a strict hierarchy in terms of the information an adversary can extract using them, and we introduced a new timing technique.

Scalability and Automation The rich expressiveness offered from the stochastic timed automata, and our adversary model, comes with the cost of scalability issues, while it also puts obstacles for automating our work. In particular, we have the following issues

- our approach relies on enumeration of the adversary's set of outputs O, which can be very large.
- the number k of the adversary's observations can be very long in realistic attacks.
- the construction of the timing channel in the stochastic case involves calculations of the form
$$\mathsf{P}_\gamma(...) = \int_{t_1 \in C_1} ... \int_{t_n \in C_n} \mathrm{d}\mu_n(t_n)...\mathrm{d}\mu_1(t_1)$$
which makes it difficult, or even impossible to calculate them sometimes.

As a first step, to overcome those difficulties, we could limit ourselves to deterministic systems. In this case, we know that min-capacity only depends on the cardinality $|O|$ of the output set O. In particular, we have that min-capacity=log$|O|$ [ACPS12], and thus any over-approximation $O^{\#} \supseteq O$ gives us a direct upper bound on the min-capacity i.e log$|O| \leq$ log$|O^{\#}|$.

For dealing with the arbitrary number k of the adversary's observations we can use results from quantitative information flow [KD09, SS17]. In particular, for (stochastic) systems with independent observations, calculating the channel for a *single* observation can be used to give us bounds on the leakage for k observations [KD09, SS17].

As another approach, we could approximate the security offered by a coarse-grained clock using techniques from statistical model checking, and automate our analysis using the model-checker for (stochastic) timed automata UPPAAL [DLL$^+$15]. This approach is particularly interesting. since it can allow for estimating the trade-off between security and safety properties of a system.

CHAPTER 7

Conclusions

In this thesis, we leveraged approaches from the theory of information flow and developed novel techniques for the qualitative and quantitative security analysis of real-time systems, which we modelled using timed automata.

Qualitative developments In Chapter 3, we started by defining a non-interference condition, which requires independence between the initial secret components and the final public components of a timed automaton. Our condition caters for adversaries that can infer secret information through timing and control-flow channels, upon the termination of the automaton. As a first step for checking our non-interference condition we developed the language timed commands, whose semantics is given using timed automata. We then developed a type system for our language, and we proved that type-checked programs satisfy non-interference. The prime intention of this work was to identify and solve core information security problems in timed automata, however, due to its language-based nature it cannot deal with every timed automaton.

Our work in Chapter 4 extended the work of Chapter 3 in various ways. First, we defined a bisimulation-based security condition. Our condition caters for (a) timed automata that need to protect not only secret initial information, but also fresh information computed during the execution of the automaton, (b) adversaries observing information at various locations of the automaton, and not only the final ones, and (c) scenarios where computations leading to certain locations of the automaton may release secret information. We then developed an algorithm that traverses a timed automaton and generates information flow constraints. In particular, our algorithm makes use of a novel post-dominator relation in order to deal with information flows created from the unstructured control flow in timed automata. Finally, we proved that whenever the constraints generated from our algorithm are satisfied, then the timed automaton satisfies our security condition.

In Chapter 5, we presented the **BTCL** logic, which can be used for enforcing data- and time-dependent access control on networks of timed automata. We started by defining information flow instrumented semantics for networks. Specifically, our semantics make use of labels, called behaviours, that capture the different explicit information flows in the network. Our logic is based on those behaviours and access control conditions are formulated as formulas in our logic. We then presented techniques for translating our networks and formulas into models of timed automata and TCTL-like formulas (resp.), that can be handled by standard model-checkers such as UPPAAL [UPP]. Finally, we implemented our translations and we illustrated how our approach can be fully automated.

Quantitative developments In Chapter 6, we presented the first information flow analysis of the countermeasure reducing clock resolution. Our development considers attack scenarios which are described by a victim and an adversary. In particular, we defined a (stochastic) timed automata-based system, which describes the victim who computes on some secret input. The system provides a clock of a certain granularity, which can be accessed by an adversary. The adversary observes the time of the system making queries to the system's clock during certain computations and tries to infer the system's secret.

We then presented novel algorithms for constructing the information theoretic timing channel of an attack scenario. Any information theoretic measure can then be used to calculate the leakage of the timing channel, allowing one to evaluate the effectiveness of this countermeasure.

We then used our techniques and we achieved the following: (1) we showed that contrary to the popular-belief a coarse-grained clock might leak more than a fine-grained one, (2) we gave sufficient conditions for when increasing the grain of the clock we achieve less information leakage, and (3) we showed that the attack techniques used to bypass this countermeasure form a strict hierarchy in terms of the information an adversary can extract using them.

APPENDIX A

Proofs of Chapter 3

A.1 Lemma 3.2

PROOF. We prove the first statement by induction on $\vdash^{[q_t:g_t]}_{[q_s:g_s]} C : \mathsf{E}, \mathsf{I}\,\&\,\chi$ using that $\rightsquigarrow$ is transitive.

We prove the second statement by induction on $\vdash^{[q_t:g_t]}_{[q_s:g_s]} T : \mathsf{E}, \mathsf{I}\,\&\,\chi$. It is immediate for the two axioms for actions because $\{q_s\} \cup \mathsf{fv}(\mathsf{fst}^{g_t}_{g_s}(T)) = \chi$. In the rule for composition for $T;^{[g]}C$ observe that $\{q_s\} \cup \mathsf{fv}(\mathsf{fst}^{g_t}_{g_s}(T;^{[g]}C)) = \{q_s\} \cup \mathsf{fv}(\mathsf{fst}^{g}_{g_s}(T))$ and that the induction hypothesis gives that $\{q_s\} \cup \mathsf{fv}(\mathsf{fst}^{g}_{g_s}(T)) \rightsquigarrow \{q\}$ because $\chi_1 \rightsquigarrow \{q\}$.

We have $\{q_s\} \cup \mathsf{fv}(\mathsf{fst}^{g}_{g_s}(T)) \rightsquigarrow \mathsf{ass}(T)$ from the induction hypothesis, $\{q\} \rightsquigarrow$ **Clocks** from the rule, and $\{q\} \rightsquigarrow \mathsf{ass}(C)$ from the previous result, and then get $\{q_s\} \cup \mathsf{fv}(\mathsf{fst}^{g_t}_{g_s}(T;^{[g]}C)) \rightsquigarrow \mathsf{ass}(T;^{[g]}C)$. Next suppose $\chi = \chi_2 \rightsquigarrow \chi'$; from the previous result we have $\{q\} \rightsquigarrow \chi'$ and hence $\{q_s\} \cup \mathsf{fv}(\mathsf{fst}^{g_t}_{g_s}(T;^{[g]}C)) \rightsquigarrow \chi'$.

A.2 Theorem 3.3

PROOF. We proceed by induction on $\vdash^{[q_t:g_t]}_{[q_s:g_s]} C : \mathsf{E}, \mathsf{I}\,\&\,\chi$.

Case: Assignment Assume that $(\sigma_0, \delta_0) \equiv_{g_s} (\sigma_0', \delta_0')$ and that

$$\eta \in \mathsf{Final}[\![(\{(q_s, g \rightarrow \boldsymbol{x} := \boldsymbol{a} \colon \boldsymbol{r}, q_t)\}), [q_s \mapsto g_s][q_t \mapsto g_t]) : q_s \mapsto q_t]\!](\sigma_0, \delta_0)$$

In case $\eta = (\sigma_1, \delta_1)$ it follows that there exists $t \geq 0$ such that $\sigma_1 = [\![\boldsymbol{x} := \boldsymbol{a}]\!]\sigma_0$ and $\delta_1 = (\delta_0 + t)[\boldsymbol{r} \mapsto \boldsymbol{0}]$, and such that $[\![g]\!](\sigma_0, (\delta_0 + t)) = \mathsf{tt}$, $[\![g_s]\!](\sigma_0, \delta_0 + t) = \mathsf{tt}$ and $[\![g_t]\!](\sigma_1, \delta_1) = \mathsf{tt}$. Defining $\eta' = (\sigma_1', \delta_1') = ([\![\boldsymbol{x} := \boldsymbol{a}]\!]\sigma_0', (\delta_0' + t)[\boldsymbol{r} \mapsto \boldsymbol{0}])$ ensures that $[\![g]\!](\sigma_0', (\delta_0' + t)) = \mathsf{tt}$, $[\![g_s]\!](\sigma_0', \delta_0' + t) = \mathsf{tt}$ and $[\![g_t]\!](\sigma_1', \delta_1') = \mathsf{tt}$ because all variables and clocks tested are low and hence

$$\eta \equiv_{g_t} \eta' \in \mathsf{Final}[\![(\{(q_s, g \rightarrow \boldsymbol{x} := \boldsymbol{a} \colon \boldsymbol{r}, q_t)\}), [q_s \mapsto g_s][q_t \mapsto g_t]) : q_s \mapsto q_t]\!](\sigma_0', \delta_0')$$

In case $\eta = \perp$ it follows that there is no value of $t \geq 0$ such that $[\![g]\!](\sigma_0, (\delta_0 + t)) = \mathsf{tt}$, $[\![g_s]\!](\sigma_0, \delta_0 + t) = \mathsf{tt}$ and $[\![g_t]\!](\sigma_1, \delta_1) = \mathsf{tt}$. Then there also is no value of $t \geq 0$ such that $[\![g]\!](\sigma_0', (\delta_0' + t)) = \mathsf{tt}$, $[\![g_s]\!](\sigma_0', \delta_0' + t) = \mathsf{tt}$ and $[\![g_t]\!](\sigma_1', \delta_1') = \mathsf{tt}$ because all variables and clocks tested are low and hence setting $\eta' = \perp$ establishes that

$$\eta \equiv \eta' \in \mathsf{Final}[\![(\{(q_s, g \rightarrow \boldsymbol{x} := \boldsymbol{a} \colon \boldsymbol{r}, q_t)\}), [q_s \mapsto g_s][q_t \mapsto g_t]) : q_s \mapsto q_t]\!](\sigma_0', \delta_0')$$

The other direction is similar and this completes the assignment case.

Case: Sequence We shall write

$$\begin{array}{rcl} F & = & \mathsf{Final}[\![(\mathsf{E}_1 \cup \mathsf{E}_2, \mathsf{I}_1 \cup \mathsf{I}_2[q_s \mapsto g_s][q \mapsto g][q_t \mapsto g_t]) : q_s \mapsto q_t]\!] \\ F_1 & = & \mathsf{Final}[\![(\mathsf{E}_1, \mathsf{I}_1[q_s \mapsto g_s][q \mapsto g]) : q_s \mapsto q]\!] \\ F_2 & = & \mathsf{Final}[\![(\mathsf{E}_2, \mathsf{I}_2[q \mapsto g][q_t \mapsto g_t]) : q \mapsto q_t]\!] \end{array}$$

and observe that $F = F_1 \diamond F_2$. The result then follows from the induction hypotheses and Fact 2.

Case: Looping We shall write

$$\begin{array}{rcl} F & = & \mathsf{Final}[\![(\bigcup_i \mathsf{E}_i, \bigcup_i \mathsf{I}_i[q_s \mapsto g_s][q_t \mapsto g_t]) : q_s \mapsto q_t]\!] \\ F_i & = & \left\{ \begin{array}{ll} \mathsf{Final}[\![(\mathsf{E}_i, \mathsf{I}_i[q_s \mapsto g_s]) : q_s \mapsto q_s]\!] & \text{whenever } i \leq n \\ \mathsf{Final}[\![(\mathsf{E}_i, \mathsf{I}_i[q_s \mapsto g_s][q_t \mapsto g_t]) : q_s \mapsto q_t]\!] & \text{whenever } i > n \end{array} \right. \end{array}$$

and this gives rise to the equation

$$F = (\bigcup_{i=1}^{n} F_i \diamond F) \cup \bigcup_{i=n+1}^{m} F_i$$

We shall consider two subcases, one where the condition $\Phi^{T_1, \cdots, T_n}_{T_{n+1}, \cdots, T_m}[{}^{q_t : g_t}_{q_s : g_s}]$ is true and one where it is false.

Subcase: Looping when $\Phi^{T_1,\cdots,T_n}_{T_{n+1},\cdots,T_m}[{}^{q_t:g_t}_{q_s:g_s}]$ is true In this case (using the notation of Table 3.2) all the variables and clocks in $\bigcup_{i=1}^{m} \mathsf{fv}(\mathsf{fst}^{g_i}_{g_s}(T_i))$ are low. Assume that $(\sigma_0, \delta_0) \equiv_{g_s} (\sigma'_0, \delta'_0)$ and that $\eta \in F(\sigma_0, \delta_0)$. This must be because of a trace as considered in Section 3.1.

If this trace visits q_s infinitely often we will be able to construct a sequence $k_1, k_2, \cdots, k_i, \cdots$ such that each $k_i \leq n$ and

$$\forall i > 0 : (\sigma_i, \delta_i) \in F_{k_i}(\sigma_{i-1}, \delta_{i-1})$$

and $\eta = \perp$. By the induction hypothesis we can find (σ'_i, δ'_i) such that

$$\forall i > 0 : (\sigma_i, \delta_i) \equiv_{g_s} (\sigma'_i, \delta'_i) \in F_{k_i}(\sigma'_{i-1}, \delta'_{i-1})$$

and this establishes that $\perp \in F(\sigma'_0, \delta'_0)$.

If the trace visits q_s only finitely often we will be able to construct a sequence $k_1, k_2, \cdots, k_j$ such that $\forall i < j : k_i \leq n$ and $k_j \leq m$ and

$$\begin{aligned}&\forall i \in \{1, \cdots, j-1\} : (\sigma_i, \delta_i) \in F_{k_i}(\sigma_{i-1}, \delta_{i-1})\\ &\eta \in F_{k_j}(\sigma_{j-1}, \delta_{j-1})\end{aligned}$$

By the induction hypothesis we can find (σ'_i, δ'_i) and η' such that

$$\begin{aligned}&\forall i \in \{1, \cdots, j-1\} : (\sigma_i, \delta_i) \equiv_{g_s} (\sigma'_i, \delta'_i) \in F_{k_i}(\sigma'_{i-1}, \delta'_{i-1})\\ &\eta \equiv \eta' \in F_{k_j}(\sigma'_{j-1}, \delta'_{j-1})\end{aligned}$$

and this establishes that $\eta' \in F(\sigma'_0, \delta'_0)$.

The other direction is similar and this completes the subcase.

Subcase: Looping when $\Phi^{T_1,\cdots,T_n}_{T_{n+1},\cdots,T_m}[{}^{q_t:g_t}_{q_s:g_s}]$ is false In this case all the variables and clocks in $\bigcup_{i=1}^{n} \mathsf{fv}(\mathsf{fst}^{g_i}_{g_s}(T_i))$ are low but this is not necessarily the case for those in $\bigcup_{i=n+1}^{m} \mathsf{fv}(\mathsf{fst}^{g_i}_{g_s}(T_i))$; however, we do know that $[\![g_s]\!](\sigma, \delta) \Rightarrow \perp \notin F(\sigma, \delta)$. Assume that $(\sigma_0, \delta_0) \equiv_{g_s} (\sigma'_0, \delta'_0)$ and that $\eta \in F(\sigma_0, \delta_0)$. The assumptions of the subcase ensure that $\eta \neq \perp$.

We will be able to construct a sequence $k_1, k_2, \cdots, k_j$ such that $\forall i < j : k_i \leq n$ and $k_j > n$ and

$$\begin{aligned}&\forall i \in \{1, \cdots, j-1\} : (\sigma_i, \delta_i) \in F_{k_i}(\sigma_{i-1}, \delta_{i-1})\\ &\eta \in F_{k_j}(\sigma_{j-1}, \delta_{j-1})\end{aligned}$$

By the induction hypothesis we can find (σ'_i, δ'_i) such that

$$\forall i \in \{1, \cdots, j-1\} : (\sigma_i, \delta_i) \equiv_{g_s} (\sigma'_i, \delta'_i) \in F_{k_i}(\sigma'_{i-1}, \delta'_{i-1})$$

There are now two scenarios for how to proceed.

Subcase scenario where all variables and clocks in $\mathsf{fv}(\mathsf{fst}^{g_t}_{g_s}(T_{k_j}))$ are low In this case we can find $\eta' \in F_{k_j}(\sigma'_{j-1}, \delta'_{j-1})$ such that $\eta \equiv \eta'$.

Subcase scenario where at least one variable or clock in $\mathsf{fv}(\mathsf{fst}^{g_t}_{g_s}(T_{k_j}))$ is high Then $\mathsf{ass}(T_{k_j})$ cannot contain any low variable or clock and hence there is $t \geq 0$ such that $\eta \equiv (\sigma_{j-1}, \delta_{j-1} + t)$ where the addition of t takes care of the potential delay in q_s. Next we use that $\perp \notin F(\sigma'_{j-1}, \delta'_{j-1})$ to obtain $k'_j, \sigma'_j, \delta'_j$ such that $(\sigma'_j, \delta'_j) \in F_{k'_j}(\sigma'_{j-1}, \delta'_{j-1})$.

It cannot be the case that $k'_j \leq n$. To see this, assume by way of contradiction that $k'_j \leq n$. Then $(\sigma_{j-1}, \delta_{j-1})$ would be a witness for $\underline{\mathrm{sat}}(\mathsf{fst}^{g_t}_{g_s}(T_{k_j}) \wedge \mathsf{fst}^{g_s}_{g_s}(T_{k'_j}))$ ensuring that $\mathsf{fst}^{g_t}_{g_s}(T_{k_j}) \rightsquigarrow \mathsf{ass}(T_{k'_j})$ so that $\mathsf{ass}(T_{k'_j})$ could not contain a low variable or clock. It would follow that there would be $t' \geq 0$ such that $(\sigma'_j, \delta'_j) \equiv_{g_s} (\sigma_{j-1}, \delta_{j-1} + t')$ where the addition of t' is due to the possibility of delay in q_s. But then we would be able to construct an infinite sequence (σ'_l, δ'_l) for $l > j$ such that $(\sigma'_l, \delta'_l) \in F_{k'_j}(\sigma'_{l-1}, \delta'_{l-1})$ and $(\sigma'_l, \delta'_l) \equiv_{g_s} (\sigma'_{j-1}, \delta'_{j-1} + t'$ would hold for $l \geq j$. But this would contradict the fact that $\perp \notin F(\sigma'_j, \delta'_j)$.

We are left with the case where $k'_j > n$. We must have that $\mathsf{ass}(T_{k'_j})$ cannot contain any low variable or clock: either one variable or clock in $\mathsf{fst}^{g_t}_{g_s}(T_{k'_j})$ is high and it follows as in a case above, or all variables and clocks in $\mathsf{fst}^{g_t}_{g_s}(T_{k'_j})$ are low and it follows because $(\sigma_{j-1}, \delta_{j-1} + t)$ is a witness for $\underline{\mathrm{sat}}(\mathsf{fst}^{g_t}_{g_s}(T_{k_j}) \wedge \mathsf{fst}^{g_t}_{g_s}(T_{k'_j}))$ and we could proceed as in a case above. Hence $(\sigma'_j, \delta'_j) = (\sigma'_{j-1}, \delta'_{j-1} + t')$ for some $t' \geq 0$.

It remains to show that t' can be chosen to be t. For this we use that all clocks in $\mathsf{fst}^{g_t}_{g_s}(T_{k_j})$ and $\mathsf{fst}^{g_t}_{g_s}(T_{k'_j})$ are low and that $\overline{\mathsf{fst}^{g_t}_{g_s}(T_{k_j})} = \overline{\mathsf{fst}^{g_t}_{g_s}(T_{k'_j})}$.

The other direction is similar and this completes the subcase.

APPENDIX B

Proofs of Chapter 4

B.1 Proposition 4.1

Assume that all the traces in $\mathsf{Traces}[\![\mathsf{TA} : q_s \mapsto q_t]\!](\sigma, \delta)$ are successful and we want to show that there exists $tr \in \mathsf{Traces}[\![\mathsf{TA} : q_s \mapsto q_t]\!](\sigma, \delta)$ with a maximal length m.

We use results from model-checking for timed automata [HSW12]. We define the region graph $RG(\mathsf{TA})$ of TA, that is a finite graph where nodes of the region graph are of the form (q, reg) where reg is a clock region, that is an equivalence class defined on the clock states (for details we refer to [HSW12]). Configurations of $RG(\mathsf{TA})$ are of the form $\langle (q, reg), \sigma \rangle$ and we have that $\langle (q, reg), \sigma \rangle \Longrightarrow \langle (q', reg'), \sigma' \rangle$ if there are $\delta \in reg$, $\delta' \in reg'$, $t \geq 0$, σ' such that the automaton TA performs the transition $\langle q, \sigma, \delta \rangle \xrightarrow{t} \langle q', \sigma', \delta' \rangle$.

Lemma 1 of [HSW12] then states that each abstract run (finite or infinite) in the region graph $RG(\mathsf{TA})$ can be instantiated by a run (finite or infinite resp.) in TA and vice versa. This is based on the property of the region graph of being *pre-stable* that is that $\langle (q, reg), \sigma \rangle \Longrightarrow \langle (q', reg'), \sigma' \rangle$ if $\forall \delta \in reg$ there are $\delta' \in reg'$, $t \geq 0$, σ' such that $\langle q, \sigma, \delta \rangle \xrightarrow{t} \langle q', \sigma', \delta' \rangle$.

Therefore, the computation tree T of $\langle q, \sigma, \delta \rangle$ in TA has the same depth as the computation tree T' of $\langle(q,[\delta]),\sigma\rangle$ in $RG(\mathsf{TA})$ where $[\delta]$ is the region that contains all the clock states that are equivalent to δ. We then recall König's infinity lemma as it applies to trees – that every tree who has infinitely-many vertices but is locally finite (each vertex has finitely-many successor vertices), has at least one infinite path [Fra97]. It is immediate that T' is a locally finite tree. Now if T' is infinite then by König's infinity lemma we have that T' has an infinite path and thus using Lemma 1 of [HSW12] we have also that T has an infinite path that corresponds to a trace $\langle q,\sigma,\delta\rangle$ in TA which contradicts our assumptions that all the traces of $\langle q,\sigma,\delta\rangle$ are finite. Therefore we can conclude that T' has a finite depth, and thus also T, and that they are equal to the number m.

B.2 Fact 4

PROOF. The first equation is straightforward by the definition of the post-dominator relation.

For the second one, that is when y is a successor (an immediate one) of q, then the only post-dominators of q is the node y. This is because there exists a non-trivial path $\pi = q\,act\,y \in \Pi_{(q,y)}$ (for some action act) such that the trivial path $\pi(1) = y$ contains only y, and therefore for any other path $\pi' \in \Pi_{(q,y)}$ in which a node q' different from y is contained in $\pi'(1)$, q' can not be a post-dominator of q since it is not contained in the trivial path $\pi(1)$.

To understand the last equation notice that if a node q'' post-dominates all of the successors of q, or it is a successor of q that post-dominates all the other successors of q then all the non-trivial paths from q to y will always visit q'' and thus $q'' \in \mathsf{ipdom}_y(q)$. Similarly, if $q'' \notin \bigcap_{q'\in\mathrm{succ}(q)} \big(\{q'\} \cup \mathsf{ipdom}_y(q')\big)$ then there exists a successor of q, $q' \neq q''$ such that q'' does not post-dominate q' and thus we can find a non-trivial path $\pi \in \Pi_{(q,Y)}$ that starts with $q\,act\,q'$ (for some action act) and does not contain q'' and thus q'' is not a post-dominator of q.

B.3 Fact 5

PROOF. To prove that $\mathsf{ipdom}_Y(q)$ is singleton we consider two cases. In the case that $\mathsf{ipdom}_Y(q) = \{q'\}$ then the proof is trivial.

Assume now that $\mathsf{ipdom}_Y(q) = \{q_1, ..., q_n\}$ ($n \geq 2$), take an arbitrary non-trivial path $\pi \in \Pi_{(q,Y)}$, and find the closest to q (the one that appears first in the path) Y post-dominator $q_j \in \mathsf{ipdom}_Y(q)$ in that path. Next, note that $q_j \notin Y$, since if $q_j \in Y$, we could shorten that path to the point that we meet q_j for the first time, and thus we have found a non trivial path $\pi' \in \Pi_{(q,Y)}$ (since $q_j \in Y$) in which $\forall i \neq j : q_i \notin \pi'(1)$, and thus $\forall i \neq j : q_i \notin \mathsf{ipdom}_Y(q)$, which contradicts our assumption.

Next, to prove that $\forall i \neq j : q_i \in \mathsf{ipdom}_Y(q_j)$, assume that this is not the case, and thus we can find $q_l \neq q_j : q_l \notin \mathsf{ipdom}_Y(q_j)$. Therefore, we can find a path $\pi'' \in \Pi_{(q_j,Y)}$ such that $q_l \notin \pi''(1)$, but this means that if we concatenate the paths π' and π'', we have a path in $\Pi_{(q,Y)}$ in which q_l does not belong to it, and thus q_l does not belong in its 1- suffix either. Therefore $q_l \notin \mathsf{ipdom}_Y(q)$, which again contradicts our assumption.

Finally, to prove that $\mathsf{ipdom}_Y(q)$ is singleton, assume that there exists another Y post-dominator of q, q_l, such that $q_l \neq q_j$, and $q_l \notin Y$, and $q_j \in \mathsf{ipdom}(q_l)$. Then this means that q_j belongs in all the 1-suffixes of the paths in the set $\Pi_{(q_l,Y)}$. Therefore take $\pi = q_l...q_j...y \in \Pi_{(q_l,Y)}$ (for some $y \in Y$) such that π contains no cycles (e.g each node occurs exactly once in the path), but then there exists a path $\pi' = q_j...y$ (the suffix of the path π) such that $q_l \notin \pi'$, and thus $q_l \notin \mathsf{pdom}_Y(q_j)$, which contradicts our assumption. Therefore, we have proved that q_j is the unique immediate Y post-dominator of q.

B.4 Theorem 4.6

Proof. Assume that $\langle q, \sigma_1, \delta_1 \rangle \xRightarrow{D_1}_Y \langle q', \sigma'_1, \delta'_1 \rangle$ because of the trace

$$\langle q, \sigma_1, \delta_1 \rangle = \langle q, \sigma_{01}, \delta_{01} \rangle \xrightarrow{t_1} ... \xrightarrow{t_k} \langle q_{k1}, \sigma_{k1}, \delta_{k1} \rangle = \langle q', \sigma'_1, \delta'_1 \rangle \quad (*)$$

where $k > 0$ and $\forall i \in \{1, .., k-1\} : q_{i1} \notin Y$ and $D_1 = \sum_{j=1}^{k} t_j$ and the first transition of the trace has happened because of the edge $e \in \mathsf{E}_q$.

We shall consider two main cases. The one where q is in $\mathsf{Q}_{\rightsquigarrow\mathsf{w}}$ and one where it is not.

Main Case 1: q is in $\mathsf{Q}_{\rightsquigarrow\mathsf{w}}$ In that case $q' \in Y_w$ and thus we only have to prove that (σ_2, δ_2) can reach q'. We start by proving a small fact.

First for a set of variables and clocks $\mathcal{Z}$, and two pairs (σ, δ), (σ', δ') we write

$$(\sigma, \delta) \equiv^{\mathcal{Z}} (\sigma', \delta') \quad \text{iff} \quad \begin{array}{l} \forall x : (x \in \mathcal{Z} \wedge \mathcal{L}(x) = L) \Rightarrow \sigma(x) = \sigma'(x) \wedge \\ \forall r : (r \in \mathcal{Z} \wedge \mathcal{L}(r) = L) \Rightarrow \delta(r) = \delta'(r) \end{array}$$

Next, for a finite path $\pi = q_0 act_1 q_1 ... q_{n-1} act_n q_n$ we define the auxiliary operator

$\mathcal{Z}(.)$ as

$$\mathcal{Z}(\pi) = \bigcup_{i=0}^{n-1} (\bigcup_{e' \in \mathsf{E}_{q_i}} \mathsf{fv}(\mathsf{con}(e')) \cup \mathsf{fv}(\mathsf{expr}(e')))$$

Now we will prove that for a path $\pi = q'_{01} act'_1 q'_{11} ... q'_{(n-1)1} act'_n q'_n \in \Pi_{(e,Y)}$, if

$$\langle q, \sigma_1, \delta_1 \rangle = \langle q'_{01}, \sigma'_{01}, \delta'_{01} \rangle \xrightarrow{t'_1} ... \xrightarrow{t'_l} \langle q'_{l1}, \sigma'_{l1}, \delta'_{l1} \rangle \quad (l \leq n) \qquad (1)$$

using the edges $(q'_{01}, act'_1, q'_{11}), ..., (q'_{(l-1)1}, act'_l, q'_l)$ and $(\sigma_1, \delta_1) \equiv^{\mathcal{Z}(\pi)} (\sigma_2, \delta_2)$ then $\exists (\sigma'_{l2}, \delta'_{l2})$:

$$\langle q, \sigma_2, \delta_2 \rangle = \langle q'_{01}, \sigma'_{02}, \delta'_{02} \rangle \xrightarrow{t'_1} ... \xrightarrow{t'_l} \langle q'_{l1}, \sigma'_{l2}, \delta'_{l2} \rangle \qquad (a)$$

and

$$l < n \Rightarrow (\sigma'_{l1}, \delta'_{l1}) \equiv^{\mathcal{Z}(\pi(l))} (\sigma'_{l2}, \delta'_{l2}) \qquad (b)$$

where recall that $\pi(l)$ is the l-suffix of π. The proof proceeds by induction on l.

Base case: $l = 1$ To prove (a), let $e = (q'_{01}, g \to \boldsymbol{x} := \boldsymbol{a} \colon \boldsymbol{r}, q'_{11})$ and note that because $(\sigma_1, \delta_1) \equiv^{\mathcal{Z}(\pi)} (\sigma_2, \delta_2)$ and $\mathsf{con}(e)$ contains only low variables (since $q'_{01} = q \in \mathsf{Q}_{\leadsto \mathsf{w}}$ and **C1** (a)) it is immediate that there exists $\sigma'_{12} = \sigma_2[\boldsymbol{x} \mapsto [\![\boldsymbol{a}]\!]\sigma_2]$, $\delta'_{12} = (\delta_2 + t'_1)[\boldsymbol{r} \mapsto \boldsymbol{0}]$ such that $[\![\mathsf{I}(q'_{01})]\!](\sigma_2, \delta_2 + t'_1) = \mathsf{tt}$ and $[\![\mathsf{I}(q'_{11})]\!](\sigma'_{12}, \delta'_{12}) = \mathsf{tt}$, and $\langle q'_{01}, \sigma_2, \delta_2 \rangle \xrightarrow{t'_1} \langle q'_{11}, \sigma'_{12}, \delta'_{12} \rangle$.

Now if $l < n$, to prove (b) we consider two cases. One where A_e is true and one where it is false. If A_e is true we note that $(\sigma'_{11}, \delta'_{11}) \equiv^{\mathcal{Z}(\pi)} (\sigma'_{12}, \delta'_{12})$, and then it is immediate that also $(\sigma'_{11}, \delta'_{11}) \equiv^{\mathcal{Z}(\pi(1))} (\sigma'_{12}, \delta'_{12})$ as required. Otherwise, if A_e is false then Ψ_e is true and thus $(\sigma'_{11}, \delta'_{11}) \equiv^{\mathcal{Z}(\pi(1))} (\sigma'_{12}, \delta'_{12})$, because the two pairs are still low equivalent for the variables that are not used in the assignment of e, while the ones used in the assignment of e they do not appear in any condition (or expression) of an edge of a node q that belongs in $\pi(1)$.

Inductive case: $l = l_0 + 1$ **(**$l_0 > 0$**)** Because of the trace in (1) we have that

$$tr_1 = \langle q'_{01}, \sigma'_{01}, \delta'_{01} \rangle \xrightarrow{t'_1} \langle q'_{11}, \sigma'_{11}, \delta'_{11} \rangle$$

and

$$tr_2 = \langle q'_{11}, \sigma'_{11}, \delta'_{11} \rangle \xrightarrow{t'_2} ... \xrightarrow{t'_l} \langle q'_{l1}, \sigma'_{l1}, \delta'_{l1} \rangle$$

Using our induction hypothesis on tr_1 we have that there exists $(\sigma'_{12}, \delta'_{12})$ such that $\langle q'_{01}, \sigma_2, \delta_2 \rangle \xrightarrow{t'_1} \langle q'_{11}, \sigma'_{12}, \delta'_{12} \rangle$ and $(\sigma'_{11}, \delta'_{11}) \equiv^{\mathcal{Z}(\pi(1))} (\sigma'_{12}, \delta'_{12})$ and the proof is completed using our induction hypothesis on tr_2.

The proof of *Main Case 1* follows by the result (a) of the fact from above, taking the path π that corresponds to the trace $(*)$ and using that $(\sigma_1, \delta_1) \equiv^{\mathcal{Z}(\pi)} (\sigma_2, \delta_2)$ (since $(\sigma_1, \delta_1) \equiv (\sigma_2, \delta_2)$ and all the nodes in π except q_{k1} have edges whose conditions contain only low variables). Therefore, since (σ_1, δ_1) creates the trace (*) we also have that $\exists(\sigma'_2, \delta'_2)$:

$$\langle q, \sigma_2, \delta_2 \rangle = \langle q_{01}, \sigma_{02}, \delta_{02} \rangle \xrightarrow{t_1} \ldots \xrightarrow{t_k} \langle q_{k1}, \sigma_{k2}, \delta_{k2} \rangle = \langle q', \sigma'_2, \delta'_2 \rangle$$

and thus for $D_2 = t_1 + \ldots + t_k$ we have that

$$\langle q, \sigma_2, \delta_2 \rangle \overset{D_2}{\Longrightarrow}_Y \langle q', \sigma'_2, \delta'_2 \rangle$$

where $q' \in Y_w$ and this completes the proof for this case.

Main Case 2: When q is not in $\mathsf{Q}_{\rightsquigarrow \mathsf{w}}$ The proof proceeds by induction on the length k of the trace $(*)$.

Base case: k=1 We have that

$$\langle q, \sigma_1, \delta_1 \rangle \xrightarrow{t_1} \langle q', \sigma'_1, \delta'_1 \rangle$$

and let $e = (q, g \rightarrow \boldsymbol{x} := \boldsymbol{a} \colon \boldsymbol{r}, q')$, then it is immediate that $D_1 = t_1$, $\sigma'_1 = \sigma_1[\boldsymbol{x} \mapsto [\![\boldsymbol{a}]\!]\sigma_1]$, $\delta'_1 = (\delta_1 + t_1)[\boldsymbol{r} \mapsto \boldsymbol{0}]$ and $[\![\mathsf{I}(q)]\!](\sigma_1, \delta_1 + t_1) = \mathsf{tt}$ and $[\![\mathsf{I}(q')]\!](\sigma'_1, \delta'_1) = \mathsf{tt}$.

We shall consider two subcases one where the unique immediate Y post-dominator of q is defined and one where it is not.

Subcase 1: When the unique immediate Y post-dominator $\mathsf{ipd}_Y(q)$ is defined It has to be the case then that q'=$\mathsf{ipd}_Y(q)$ since $q' \in Y$ and in particular, we have that $q' \in Y_s$. We will proceed by considering two other subcases of the *Subcase 1*, one where the condition Φ_q is *true* and one which it is *false*.

Subcase 1 (a): When Φ_q is true Then it is the case that all the variables of the condition $\mathsf{con}(e)$ are low and thus it is immediate that there exists $t_2 = t_1$ and $\sigma'_2 = \sigma_2[\boldsymbol{x} \mapsto$

$[\![\boldsymbol{a}]\!]\sigma_2]$, $\delta_2' = (\delta_2 + t_2)[\boldsymbol{r} \mapsto \mathbf{0}]$ and $[\![\mathsf{I}(q)]\!](\sigma_2, \delta_2 + t_2) = \mathsf{tt}$ and $[\![\mathsf{I}(q')]\!](\sigma_2', \delta_2') = \mathsf{tt}$ such that $\langle q, \sigma_2, \delta_2 \rangle \xrightarrow{t_2} \langle q', \sigma_2', \delta_2' \rangle$ which implies that for $D_2 = t_2$

$$\langle q, \sigma_2, \delta_2 \rangle \overset{D_2}{\Longrightarrow}_Y \langle q', \sigma_2', \delta_2' \rangle$$

Finally, because $\mathrm{sec}_{Y,\mathcal{L}}(\mathsf{TA})$, condition **C2** (a) gives us that A_e is true, and thus all the *explicit flows* arising from the assignments $\boldsymbol{x} := \boldsymbol{a}$ are permissible and thus $(\sigma_1', \delta_1') \equiv (\sigma_2', \delta_2')$ as required.

Subcase 1 (b): When Φ_q is false If it is the case that all the variables in the condition $\mathsf{con}(e)$ are low then the proof proceeds as in *Subcase 1(a).*

For the case now that at least one variable in the condition $\mathsf{con}(e)$ is high then because $\mathrm{sec}_{Y,\mathcal{L}}(\mathsf{TA})$, condition **C2** (a) and Fact 6 ensure that

$$\forall x : \mathcal{L}(x) = L \Rightarrow \sigma_1'(x) = \sigma_1(x)$$

and

$$\forall r : \mathcal{L}(r) = L \Rightarrow \delta_1'(r) = \delta_1(r) + t_1$$

Since Φ_q is false (σ_1, δ_1) and (σ_2, δ_2) have the same *termination behaviour* and thus there exists $t_2 = t_1$ and (σ_2', δ_2') such that $\langle q, \sigma_2, \delta_2 \rangle \xrightarrow{t_2} \langle q', \sigma_2', \delta_2' \rangle$ and therefore for $D_2 = t_2$ we have that

$$\langle q, \sigma_2, \delta_2 \rangle \overset{D_2}{\Longrightarrow}_Y \langle q', \sigma_2', \delta_2' \rangle$$

We just showed that $(\sigma_1', \delta_1') \equiv (\sigma_1, \delta_1 + t_1) \equiv (\sigma_2, \delta_2 + t_2)$ and we will now show that $(\sigma_2', \delta_2') \equiv (\sigma_2, \delta_2 + t_2)$.

Now if

$$\langle q, \sigma_2, \delta_2 \rangle \xrightarrow{t_2} \langle q', \sigma_2', \delta_2' \rangle$$

using the edge e or an edge $e' \neq e$ such that $\mathsf{con}(e')$ contains a high variable, since $\mathrm{sec}_{Y,\mathcal{L}}(\mathsf{TA})$, condition **C2** (a) and Fact 6 gives that

$$\forall x : \mathcal{L}(x) = L \Rightarrow \sigma_2'(x) = \sigma_2(x)$$

and

$$\forall r : \mathcal{L}(r) = L \Rightarrow \delta_2'(r) = \delta_2(r) + t_2$$

and therefore $(\sigma_2', \delta_2') \equiv (\sigma_2, \delta_2 + t_2)$ as required.

If now $\mathsf{con}(e')$ contains only low variables, (σ_1, δ_1) is a witness of $\underline{\mathrm{sat}}(\mathsf{con}(e) \wedge \mathsf{con}(e'))$ and therefore because $\mathrm{sec}_{Y,\mathcal{L}}(\mathsf{TA})$, using the condition **C2** (b) and Fact 6 we work as before and we obtain that $(\sigma_2', \delta_2') \equiv (\sigma_2, \delta_2 + t_2)$.

Subcase 2: When the unique immediate Y post-dominator of q is not defined In that case, all the variables in $\mathsf{con}(e)$ are low. If q' is in Y_w we have that $e \rightsquigarrow w$ and we proceed as in *Main Case 1.* Otherwise, we proceed as in *Subcase 1(a).*

This completes the case for $k = 1$.

Inductive case: ($k = k_0 + 1$) We have that

$$\langle q, \sigma_1, \delta_1 \rangle = \langle q, \sigma_{01}, \delta_{01} \rangle \xrightarrow{t_1} ... \xrightarrow{t_k} \langle q_{k1}, \sigma_{k1}, \delta_{k1} \rangle = \langle q', \sigma_1', \delta_1' \rangle$$

and recall that the first transition happened because of the edge e and that q is not in $\mathsf{Q}_{\rightsquigarrow \mathsf{w}}$.

We shall consider two cases again, one where the unique immediate Y post-dominator of q is defined and one where it is not.

Subcase 1: When the unique immediate-post dominator $\mathsf{ipd}_Y(q)$ is defined We will procceed by considering two subcases of *Subcase 1*, one where Φ_q is *true* and one where Φ_q is *false*.

Subcase 1 (a): When Φ_q is true Since Φ_q is true we have that all the variables in $\mathsf{con}(e)$ are low and thus $\exists t_1' = t_1$ and $(\sigma_{12}, \delta_{12}) \equiv (\sigma_{11}, \delta_{11})$ (this is ensured by our assumptions that $\mathrm{sec}_{Y,\mathcal{L}}(\mathsf{TA})$ and the predicate A_e of the condition **C2** (a) that takes care of the *explicit flows* arising from the assignment in the edge e) such that

$$\langle q, \sigma_2, \delta_2 \rangle = \langle q_{01}, \sigma_{02}, \delta_{02} \rangle \xrightarrow{t_1'} \langle q_{11}, \sigma_{12}, \delta_{12} \rangle \quad (1)$$

Since q is not in $\mathsf{Q}_{\rightsquigarrow \mathsf{w}}$, note that it is also the case that q_{11} is not in $\mathsf{Q}_{\rightsquigarrow \mathsf{w}}$ and thus using that $(\sigma_{12}, \delta_{12}) \equiv (\sigma_{11}, \delta_{11})$ and our induction hypothesis on the trace

$$\langle q_{11}, \sigma_{11}, \delta_{11} \rangle \xrightarrow{t_2} ... \xrightarrow{t_k} \langle q_{k1}, \sigma_{k1}, \delta_{k1} \rangle$$

we have that $\exists(\sigma_2', \delta_2')$ and $D_2' = t_2 + ... + t_k$ such that

$$\langle q_{11}, \sigma_{12}, \delta_{12} \rangle \overset{D_2'}{\Longrightarrow}_Y \langle q', \sigma_2', \delta_2' \rangle \quad (2)$$

and therefore by (1) and (2) and for $D_2 = t_1' + D_2'$ we have that

$$\langle q, \sigma_2, \delta_2 \rangle \overset{D_2}{\Longrightarrow}_Y \langle q', \sigma_2', \delta_2' \rangle$$

and $(\sigma_1', \delta_1') \equiv (\sigma_2', \delta_2') \vee q' \in Y_w$ as required.

Subcase 1 (b): When Φ_q is false In the case that all the variables in con(e) are low then the proof proceeds as in Subcase 1(a).

Assume now that at least one variable in $\mathsf{con}(e)$ is high. Since $\mathsf{ipd}_Y(q)$ is defined then there exists $j \in \{1, ..., k\}$ such that q_{j1}=$\mathsf{ipd}_Y(q)$ and $\forall i \in \{1, .., j-1\} : q_{i1} \neq \mathsf{ipd}_Y(q)$. Therefore we have that

$$\langle q_{01}, \sigma_{01}, \delta_{01}\rangle \xrightarrow{t_1} ... \xrightarrow{t_j} \langle q_{j1}, \sigma_{j1}, \delta_{j1}\rangle \xrightarrow{t_{j+1}} ... \xrightarrow{t_k} \langle q_{k1}, \sigma_{k1}, \delta_{k1}\rangle$$

Next, using that $\text{sec}_{Y,\mathcal{L}}(\mathsf{TA})$, condition **C2** (a) and Fact 6 gives us that

$$\forall x : \mathcal{L}(x) = L \Rightarrow \sigma_{j1}(x) = \sigma_{01}(x)$$

and

$$\forall r : \mathcal{L}(r) = L \Rightarrow \delta_{j1}(r) = \delta_{01}(r) + t_1 + ... + t_j$$

Since Φ_q is false, (σ_1, δ_1) and (σ_2, δ_2) have the same *termination behaviour* and thus there exists trace $tr' \in \mathsf{Traces}[\![\mathsf{TA} : q \mapsto \mathsf{ipd}_Y(q)]\!](\sigma_2, \delta_2)$ and $t'_1, ..., t'_l$ such that $t_1 + ... + t_j = t'_1 + ... + t'_l$ and $(\sigma_{l2}, \delta_{l2})$ such that tr' is

$$\langle q, \sigma_2, \delta_2\rangle = \langle q, \sigma_{02}, \delta_{02}\rangle \xrightarrow{t'_1} ... \xrightarrow{t'_l} \langle q_{l2}, \sigma_{l2}, \delta_{l2}\rangle \quad (3)$$

and q_{l2}=$\mathsf{ipd}_Y(q)$.

It is immediate that

$$\forall x : \mathcal{L}(x) = L \Rightarrow \sigma_{l2}(x) = \sigma_{02}(x)$$

and

$$\forall r : \mathcal{L}(r) = L \Rightarrow \delta_{l2}(r) = \delta_{02}(r) + t'_1 + ... + t'_l$$

To see how we obtain this result, we have that if tr' has started using the edge e or an edge $e' \neq e$, where $\mathsf{con}(e')$ contains at least one high variable, then this result follows by our assumptions that $\text{sec}_{Y,\mathcal{L}}(\mathsf{TA})$, condition **C2** (a) and Fact 6. Now if the tr' has started using an edge $e' \neq e$ and $\mathsf{con}(e')$ contains only low variables then (σ_1, δ_1) is a witness of $\underline{\text{sat}}(\mathsf{con}(e) \wedge \mathsf{con}(e'))$ and the result follows by our assumptions that $\text{sec}_{Y,\mathcal{L}}(\mathsf{TA})$, condition **C2** (b) and Fact 6. Therefore in any case $(\sigma_{j1}, \delta_{j1}) \equiv (\sigma_{l2}, \delta_{l2})$.

Now if $\mathsf{ipd}_Y(q)$=q_{k1} the proof has been completed. Otherwise, we have that $\mathsf{ipd}_Y(q)$ is not in $\mathsf{Q}_{\rightsquigarrow\mathsf{w}}$ and the proof follows by an induction on the trace

$$\langle q_{j1}, \sigma_{j1}, \delta_{j1}\rangle \xrightarrow{t_j} ... \xrightarrow{t_k} \langle q_{k1}, \sigma_{k1}, \delta_{k1}\rangle$$

using that $(\sigma_{j1}, \delta_{j1}) \equiv (\sigma_{l2}, \delta_{l2})$

Subcase 2: When the unique immediate Y post-dominator of q is not defined In that case, all the variables in $\mathsf{con}(e)$ are low. Therefore, if $e \rightsquigarrow w$ we proceed similar to *Main Case 1*, otherwise we proceed as in Subcase 1(a).

This completes our proof.

B.5 Theorem 4.7

PROOF. Assume that $\langle q, \sigma_1, \delta_1 \rangle \overset{\infty}{\Longrightarrow}_Y \perp$ and thus either there exists a finite unsuccessful trace tr

$$\langle q, \sigma_1, \delta_1 \rangle = \langle q_{01}, \sigma_{01}, \delta_{01} \rangle \xrightarrow{t_1} ... \xrightarrow{t_n} \langle q_{n1}, \sigma_{n1}, \delta_{n1} \rangle \quad (n \geq 0)$$

such that $\forall i \in \{1, ..., n\} : q_{i1} \notin Y$ and $\langle q_{n1}, \sigma_{n1}, \delta_{n1} \rangle$ is stuck, or there exists an infinite unsuccessful trace tr

$$\langle q, \sigma_1, \delta_1 \rangle = \langle q_{01}, \sigma_{01}, \delta_{01} \rangle \xrightarrow{t_1} ... \xrightarrow{t_n} \langle q_{n1}, \sigma_{n1}, \delta_{n1} \rangle \xrightarrow{t_{n+1}} ...$$

such that $\forall i > 0 : q_{i1} \notin Y$.

Assume now that all the traces from $\langle q, \sigma_2, \delta_2 \rangle$ to a node $q' \in Y$ are successful, which means that $\langle q, \sigma_2, \delta_2 \rangle \overset{\infty}{\not\Longrightarrow}_Y \perp$ and thus by Proposition 4.1 the set

$$\begin{array}{ll} \{k \mid \langle q'_0, \sigma'_0, \delta'_0 \rangle \xrightarrow{t'_1} ... \xrightarrow{t'_k} \langle q'_k, \sigma'_k, \delta'_k \rangle : & \langle q'_0, \sigma'_0, \delta'_0 \rangle = \langle q, \sigma_2, \delta_2 \rangle \wedge q'_k \in Y \wedge \\ & \forall i \in \{1, ..., k-1\} : q'_i \notin Y\} \end{array}$$

has a maximum m.

The proof proceeds by contradiction where we show that we can either construct an unsuccessful trace of $\langle q, \sigma_2, \delta_2 \rangle$ or a “long" trace tr'

$$\langle q, \sigma_2, \delta_2 \rangle = \langle q_{02}, \sigma_{02}, \delta_{02} \rangle \xrightarrow{t'_1} ... \xrightarrow{t'_l} \langle q_{l2}, \sigma_{l2}, \delta_{l2} \rangle \quad (l > 0)$$

where $\forall i \in \{1, ..., l\} : q_{i2} \notin Y$ and $m \leq l$ and that would mean that this trace will either terminate later (at a node in Y) and thus it will have a length greater than m, or it will result into an unsuccessful trace.

We consider two main cases one where q is in $\mathsf{Q}_{\rightsquigarrow \mathsf{w}}$ and one where it isn’t.

Main Case 1: When q is in $\mathsf{Q}_{\rightsquigarrow \mathsf{w}}$ If the trace tr of $\langle q, \sigma_1, \delta_1 \rangle$ visits only nodes that can reach Y ($\forall i : \Pi_{q_{i1}} \neq \emptyset$) then we proceed similar to the proof of Main Case 1 of Theorem 4.6, using the result (a) and (b) of the fact proven there. Therefore if tr is infinite we can show that (σ_2, δ_2) can simulate the first m steps of (σ_1, δ_1) and this give us the desired trace tr'. Similarly, in case of tr being a finite unsuccessful trace that stops at the node q_{n1}, and $\langle q_{n1}, \sigma_{n1}, \delta_{n1}\rangle$ is a stuck, we can also show that (σ_2, δ_2) can reach the node q_{n1} (using the result (a)) and the resulting configuration will be stuck (using the result (b)).

Now if the first $j > 0$ nodes $q_{01}...q_{j1}$ (visited by tr) can reach Y and then for the node $q_{(j+1)1}$ we have that $\Pi_{(q_{(j+1)1}, Y)} = \emptyset$, we can show similarly as before that (σ_2, δ_2) can reach the node $q_{(j+1)1}$ (using the results (a) and (b)), and thus any further computation will lead to an unsuccessful trace since $\Pi_{(q_{(j+1)1}, Y)} = \emptyset$.

Finally if tr visits only nodes that cannot reach Y ($\forall i : \Pi_{q_{i1}} = \emptyset$) and thus also q cannot reach Y, the proof is trivial since all the traces of $\langle q, \sigma_2, \delta_2\rangle$ will be unsuccessful with respect to Y. This completes the proof of Main Case 1.

Main Case 2: When q is not in $\mathsf{Q}_{\rightsquigarrow\mathsf{w}}$ We will now present a finite construction strategy for the desired trace tr'.

Construction We start by looking at the configurations $\langle q, \sigma_1, \delta_1\rangle$, $\langle q, \sigma_2, \delta_2\rangle$ the unsuccessful trace tr of (σ_1, δ_1), and we remember that so far we have created a trace $tr' = \langle q, \sigma_2, \delta_2\rangle$ of length $l = 0$. We proceed according to the following cases:

Case 1: When the unique immediate Y post-dominator $\mathsf{ipd}_Y(q)$ of q is defined We then consider two subcases, one where Φ_q is *false* and one where Φ_q is *true*.

Subcase (a): Φ_q is false Now if the trace tr does *not* visit $\mathsf{ipd}_Y(q)$, we have that (σ_1, δ_1) and (σ_2, δ_2) have the same *termination behaviour* (using that Φ_q is false) and thus there exists a trace tr' of (σ_2, δ_2) that never visits $\mathsf{ipd}_Y(q)$. However, then we would have the case that tr' is an unsuccessful trace with respect to q and the set Y which contradicts our assumptions.

If the trace tr does visit $\mathsf{ipd}_Y(q)$, then it has to be the case that $\mathsf{ipd}_Y(q)$ is not in Y. Assume now that tr starts with an edge $e \in \mathsf{E}_q$. If $\mathsf{con}(e)$ contains only low variables then $\exists t'_1 = t_1$ and $(\sigma_{12}, \delta_{12}) \equiv (\sigma_{11}, \delta_{11})$ (this is ensured by our assumptions that $\mathrm{sec}_{Y,\mathcal{L}}(\mathsf{TA})$ and the predicate A_e of condition **C2** (a) that takes care of the *explicit flows* arising from the assignment in the edge e) such that

$$\langle q, \sigma_2, \delta_2\rangle = \langle q_{02}, \sigma_{02}, \delta_{02}\rangle \xrightarrow{t'_1} \langle q_{12}, \sigma_{12}, \delta_{12}\rangle$$

where $q_{12} = q_{11}$. If now $m \leq l + 1$ then we have our desired trace tr' and we stop.

Otherwise, notice that also q_{11} is not in $\mathsf{Q}_{\rightsquigarrow \mathsf{w}}$ and we repeat the *Construction* by looking at the configurations $\langle q_{11}, \sigma_{11}, \delta_{11}\rangle$, $\langle q_{11}, \sigma_{12}, \delta_{12}\rangle$, the suffix of tr that starts with $\langle q_{11}, \sigma_{11}, \delta_{11}\rangle$ and we remember that so far we have created the trace

$$tr' = \langle q_{02}, \sigma_{02}, \delta_{02}\rangle \xrightarrow{t'_1} \langle q_{12}, \sigma_{12}, \delta_{12}\rangle \quad (\langle q, \sigma_2, \delta_2\rangle = \langle q_{02}, \sigma_{02}, \delta_{02}\rangle)$$

that has length equal to l+1.

Now if $\mathsf{con}(e)$ contains at least one high variable then we look at the first occurrence of $\mathsf{ipd}_Y(q)$ in tr and let that to be the configuration $\langle q_{h1}, \sigma_{h1}, \delta_{h1}\rangle$ for some $h > 0$. Therefore, since $\mathrm{sec}_{Y,\mathcal{L}}(\mathsf{TA})$, using the condition **C2** (a) and Fact 6 we have that that

$$\forall x : \mathcal{L}(x) = L \Rightarrow \sigma_{h1}(x) = \sigma_{01}(x)$$

and

$$\forall r : \mathcal{L}(r) = L \Rightarrow \delta_{h1}(r) = \delta_{01}(r) + t_1 + ... + t_h$$

Since Φ_q is false (σ_1, δ_1) and (σ_2, δ_2) have the same *termination behaviour* and thus there exists trace $tr' \in \mathsf{Traces}[\![\mathsf{TA} : q \mapsto \mathsf{ipd}_Y(q)]\!](\sigma_2, \delta_2)$ and $t'_1, ..., t'_j$ such that $t_1 + ... + t_h = t'_1 + ... + t'_j$ and $(\sigma_{j2}, \delta_{j2})$ such that tr' is

$$\langle q, \sigma_2, \delta_2\rangle = \langle q_{02}, \sigma_{02}, \delta_{02}\rangle \xrightarrow{t'_1} ... \xrightarrow{t'_j} \langle q_{j2}, \sigma_{j2}, \delta_{j2}\rangle$$

where q_{j2} =$\mathsf{ipd}_Y(q)$.

Now if $j + l \geq m$ we have constructed the required trace tr'.

Otherwise, we have that

$$\forall x : \mathcal{L}(x) = L \Rightarrow \sigma_{j2}(x) = \sigma_{02}(x)$$

and

$$\forall r : \mathcal{L}(r) = L \Rightarrow \delta_{j2}(r) = \delta_{02}(r) + t'_1 + ... + t'_j$$

To see how we obtain this result, we have that if tr' has started using the edge e or an edge $e' \neq e$, where $\mathsf{con}(e')$ contains at least one high variable, then this result follows by our assumptions that $\mathrm{sec}_{Y,\mathcal{L}}(\mathsf{TA})$, condition **C2** (a) and Fact 6. Now if the tr' has started using an edge $e' \neq e$ and $\mathsf{con}(e')$ has only low variables then (σ_1, δ_1) is a witness of $\underline{\mathrm{sat}}(\mathsf{con}(e) \wedge \mathsf{con}(e'))$ and the result follows again by our assumptions that $\mathrm{sec}_{Y,\mathcal{L}}(\mathsf{TA})$, condition **C2** (b) and Fact 6. Therefore in any case $(\sigma_{h1}, \delta_{h1}) \equiv (\sigma_{j2}, \delta_{j2})$ and thus we repeat the *Construction* by looking at the configurations $\langle q_{h1}, \sigma_{h1}, \delta_{h1}\rangle$, $\langle q_{j2}, \sigma_{j2}, \delta_{j2}\rangle$ the suffix of tr that starts with $\langle q_{h1}, \sigma_{h1}, \delta_{h1}\rangle$ and we remember that so far we have created the trace tr'

$$\langle q, \sigma_2, \delta_2\rangle = \langle q_{02}, \sigma_{02}, \delta_{02}\rangle \xrightarrow{t'_1} ... \xrightarrow{t'_j} \langle q_{j2}, \sigma_{j2}, \delta_{j2}\rangle$$

of length equal to $l + j$.

Subcase (b): Φ_q is true Then if tr starts with the edge e, because $\mathrm{sec}_{Y,\mathcal{L}}(\mathsf{TA})$, $\mathsf{con}(e)$ contains only low variables and we proceed as in Subcase (a).

Case 2: When the unique immediate Y post-dominator $\mathsf{ipd}_Y(q)$ of q is not defined In this case, if tr starts with the edge e, because $\mathrm{sec}_{Y,\mathcal{L}}(\mathsf{TA})$ we have that $\mathsf{con}(e)$ contains only low variables. Now if $e \rightsquigarrow w$ working as in Main Case 1 we can get an unsuccessful trace tr', otherwise we proceed as in Subcase (a).

B.6 Corollary 4.8

PROOF. Let

$$\begin{array}{rcl} Z & = & \{(\langle q,\sigma,\delta\rangle,\langle q,\sigma',\delta'\rangle) \mid [\![\mathsf{I}(q)]\!](\sigma,\delta) \wedge [\![\mathsf{I}(q)]\!](\sigma',\delta')\} \\ & & \cup\{(\bot,\bot)\} \end{array}$$

It is immediate by Theorem 4.6, and Theorem 4.7 that Z is a $Y-$bisimulation and that

$$\begin{array}{rl} \forall q \in \{q_\circ\} \cup Y : \forall(\sigma,\delta),(\sigma',\delta') : & [\![\mathsf{I}(q)]\!](\sigma,\delta) \ \wedge \ [\![\mathsf{I}(q)]\!](\sigma',\delta') \\ & \Downarrow \\ & (\langle q,\sigma,\delta\rangle,\langle q,\sigma',\delta'\rangle) \in Z \end{array}$$

Therefore since $\sim_Y$ is the largest $Y-$bisimulation we have that $Z \subseteq \sim_Y$ and thus TA satisfies Y-bisimulation security.

APPENDIX C

Proofs of Chapter 5

C.1 Proof of Theorem 5.2

PROOF. The proof proceeds by structural induction on ϕ.

The base cases are trivial since $\mathcal{T}[\![\phi]\!] = \phi$, the formula ϕ does not include any constraint about the clock r_u, and $\gamma \cong \gamma'$.

For the $\forall\Box_{\mathfrak{b}}(\phi_1, \phi_2)$ case, assume that $\gamma \models \forall\Box_{\mathfrak{b}}(\phi_1, \phi_2)$ and thus by definition

$$\begin{aligned}&\forall \gamma_0 \overset{\mathfrak{b}_1}{\Longrightarrow} \gamma_1 \overset{\mathfrak{b}_2}{\Longrightarrow} .. \in \mathsf{Runs}_{\mathsf{N}}(\gamma) : \\ &\qquad \forall i \geq 1 : \mathfrak{b}_i = \mathfrak{b} \Rightarrow \gamma_{i-1} \models \phi_1 \text{ and } \gamma_i \models \phi_2 \qquad (1)\end{aligned}$$

Now take arbitrary run $\rho' = \gamma'_0 \longrightarrow \gamma'_1 \longrightarrow ... \in \mathsf{Runs}_{\mathsf{BA}}(\gamma')$, where $\gamma' = \gamma'_0$ and prove that

$$\forall j \geq 0 : \gamma'_j \models \mathfrak{b} \Rightarrow (\mathcal{T}[\![\phi_1]\!] \wedge \exists(\mathfrak{b}\ U(\neg\mathfrak{b} \wedge \mathcal{T}[\![\phi_2]\!])))$$

Now if ρ' has length 0 then $\rho' = \gamma'$ and the proof is trivial since γ' is a genuine configuration and thus $\gamma' \not\models \mathfrak{b}$. Similarly, if ρ' has length greater than 0 and γ'_j is a genuine configuration then the proof holds. Now if γ'_j is an auxiliary configuration, consider the macro tranistion run $\mathsf{MTR}(\rho) = \mathsf{t}'_1\mathsf{t}'_2...$ of the run ρ' and let t'_i be the macro tranistion which corresponds to the transition in which γ'_j is being involved and thus we have that $\mathsf{MTR}(\rho')(i) = \gamma'_{j-1} \longrightarrow \gamma'_j \longrightarrow \gamma'_{j+1}$. Next, let $\gamma'_j = \langle v, \sigma, \delta, \kappa \rangle$ and using Fact 7 we have that $\exists \rho \in \mathsf{Runs_N}(\gamma) : \rho \cong \rho'$ and thus

$$\begin{array}{ll} & \forall h \geq 1 : \mathsf{TR}(\rho)(h) \cong \mathsf{MTR}(\rho')(h) \\ \Rightarrow & \mathsf{TR}(\rho)(i) \cong \mathsf{MTR}(\rho')(i) \\ \Leftrightarrow & \gamma_{i-1} \overset{\mathfrak{b}_i}{\Longrightarrow} \gamma_i \cong \gamma'_{j-1} \longrightarrow \gamma'_j \longrightarrow \gamma'_{j+1} \\ \Leftrightarrow & \gamma_{i-1} \cong \gamma'_{j-1} \text{ and } \gamma_i \cong \gamma'_{j+1} \text{ and } \mathsf{L}(v) = \mathfrak{b}_i \quad (2) \end{array}$$

Now if $\gamma'_j \not\models \mathfrak{b}$ then the proof is trivial. Otherwise, because of (2) ($\mathsf{L}(v) = \mathfrak{b}_i$) we have that also $\mathfrak{b}_i = \mathfrak{b}$ and using (1) we have that $\gamma_{i-1} \models \phi_1$ and $\gamma_i \models \phi_2$. Next, using (2) ($\gamma_{i-1} \cong \gamma'_{j-1}$) and our induction hypothesis we have also that $\gamma'_{j-1} \models \mathcal{T}[\![\phi_1]\!]$ and since ϕ_1 does not contain any nested formulas we also have that $\gamma'_j \models \mathcal{T}[\![\phi_1]\!]$ as required. Finally, using (2) ($\gamma_i \cong \gamma'_{j+1}$) and our induction hypothesis we have also that $\gamma'_{j+1} \models \mathcal{T}[\![\phi_2]\!]$ and thus $\gamma'_j \models \exists(\mathfrak{b}\ U\ (\neg\mathfrak{b} \wedge \mathcal{T}[\![\phi_2]\!]))$ as required.

For the other direction now assume that

$$\gamma' \models \forall\Box\mathfrak{b} \Rightarrow (\mathcal{T}[\![\phi_1]\!] \ \wedge \exists(\mathfrak{b}\ U(\neg\mathfrak{b} \wedge \mathcal{T}[\![\phi_2]\!])))$$

and thus

$$\begin{array}{l} \forall \gamma'_0 \longrightarrow \gamma'_1 \longrightarrow \gamma'_2 .... \in \mathsf{Runs_{BA}}(\gamma') : \\ \qquad \forall i \geq 0 : \gamma'_i \models \mathfrak{b} \Rightarrow (\mathcal{T}[\![\phi_1]\!] \ \wedge \exists(\mathfrak{b}\ U(\neg\mathfrak{b} \wedge \mathcal{T}[\![\phi_2]\!]))) \quad (3) \end{array}$$

and take arbitrary run $\rho = \gamma_0 \overset{\mathfrak{b}_1}{\Longrightarrow} \gamma_1 \overset{\mathfrak{b}_2}{\Longrightarrow} ... \in \mathsf{Runs_N}(\gamma)$, where $\gamma_0 = \gamma$ and prove that

$$\forall j \geq 1 : \mathfrak{b}_j = \mathfrak{b} \Rightarrow \gamma_{j-1} \models \phi_1 \text{ and } \gamma_j \models \phi_2$$

For the cases where the length of ρ is 0 or $\mathfrak{b}_j \neq \mathfrak{b}$ then the proof is trivial. Therefore take j such that $\mathfrak{b}_j = \mathfrak{b}$ and consider the transition run $\mathsf{TR}(\rho) = \mathsf{t}_1\mathsf{t}_2....$ of the run ρ and thus, using Fact 7 we have that $\exists \rho' \in \mathsf{Runs_{BA}}(\gamma') : \rho \cong \rho'$ and consequently

$$\begin{array}{ll} & \forall h \geq 1 : \mathsf{TR}(\rho)(h) \cong \mathsf{MTR}(\rho')(h) \\ \Rightarrow & \mathsf{TR}(\rho)(j) \cong \mathsf{MTR}(\rho')(j) \\ \Leftrightarrow & \gamma_{j-1} \overset{\mathfrak{b}_j}{\Longrightarrow} \gamma_j \cong \gamma'_s \longrightarrow \gamma_{aux} \longrightarrow \gamma'_t \\ \Leftrightarrow & \gamma_{j-1} \cong \gamma'_s \text{ and } \gamma_j \cong \gamma'_t \text{ and if } \gamma_{aux} = \langle v, \sigma, \delta, \kappa \rangle \text{ then } \mathsf{L}(v) = \mathfrak{b}_j \quad (4) \end{array}$$

Therefore because of (4) ($\mathsf{L}(v) = \mathfrak{b}_j$) and (3) we have that

$$\gamma_{aux} \models \mathcal{T}[\![\phi_1]\!] \wedge \exists(\mathfrak{b}\ U\ (\neg\mathfrak{b} \wedge \mathcal{T}[\![\phi_2]\!]))$$

and thus since ϕ_1 does not contain any nested formulas, $\gamma'_s \models \mathcal{T}[\![\phi_1]\!]$ and $\gamma'_t \models \mathcal{T}[\![\phi_2]\!]$; but then using (4) ($\gamma_{j-1} \cong \gamma'_s$ and $\gamma_j \cong \gamma'_t$) and our induction hypothesis we get the required result.

Finally, the cases $\phi_1 \wedge \phi_2$ and $\neg\phi$ can be proved straightforwardly using structural induction on ϕ_1, ϕ_2 and ϕ.

APPENDIX D

Proofs of Chapter 6

D.1 Proof of Fact 8

PROOF. A clock c with grain $\mathfrak{g}$, defines an equivalence relation on $\mathbb{R}_{\geq 0}$ by for t_1, $t_2 \in \mathbb{R}_{\geq 0}$ we have that

$$t_1 \equiv_{\mathfrak{g}} t_2 \text{ iff } c(t_1) = c(t_2)$$

and observe that the equivalence classes of $\equiv_{\mathfrak{g}}$ are then described by the intervals $[0, \mathfrak{g})$, $[\mathfrak{g}, 2 \cdot \mathfrak{g})$, $[2 \cdot \mathfrak{g}, 3 \cdot \mathfrak{g})$.....

Now for two clocks c_1 and c_2 with grains $\mathfrak{g}_1$ and $\mathfrak{g}_2$ (resp.) where $\mathfrak{g}_2 = n \cdot \mathfrak{g}_1$ (for n being a positive integer) we have that $\equiv_{\mathfrak{g}_1}$ refines every equivalence class of $\equiv_{\mathfrak{g}_2}$ in exactly n equivalence classes (e.g for the first equivalence class $[0, \mathfrak{g}_2)$ of $\equiv_{\mathfrak{g}_2}$ we have the n equivalence classes $[0, \mathfrak{g}_1), ..., [(n-1) \cdot \mathfrak{g}_1, n \cdot \mathfrak{g}_1)$ of $\equiv_{\mathfrak{g}_1}$). Therefore for t_1, $t_2 \in \mathbb{R}_{\geq 0}$ we have that

$$t_1 \equiv_{\mathfrak{g}_1} t_2 \Rightarrow t_1 \equiv_{\mathfrak{g}_2} t_2$$

which give us that

$$c_1(t_1) = c_1(t_2) \Rightarrow c_2(t_1) = c_2(t_2)$$

as required.

D.2 Proof of Fact 9

PROOF. Since $n \in \mathbb{N}$ is a natural number we have that $n = m \cdot \mathfrak{g} + (n \bmod \mathfrak{g})$ for some integer m. We then have that $c(t_1 + n) = c(t_2 + n)$ iff

$$\begin{array}{l}
\left\lfloor \frac{t_1+n}{\mathfrak{g}} \right\rfloor \cdot \mathfrak{g} = \left\lfloor \frac{t_2+n}{\mathfrak{g}} \right\rfloor \cdot \mathfrak{g} \\
\Leftrightarrow \left\lfloor \frac{t_1+m\cdot\mathfrak{g}+(n \bmod \mathfrak{g})}{\mathfrak{g}} \right\rfloor \cdot \mathfrak{g} = \left\lfloor \frac{t_2+m\cdot\mathfrak{g}+(n \bmod \mathfrak{g})}{\mathfrak{g}} \right\rfloor \cdot \mathfrak{g} \\
\Leftrightarrow \left\lfloor \frac{t_1+(n \bmod \mathfrak{g})}{\mathfrak{g}} + m \right\rfloor \cdot \mathfrak{g} = \left\lfloor \frac{t_2+(n \bmod \mathfrak{g})}{\mathfrak{g}} + m \right\rfloor \cdot \mathfrak{g} \\
\Leftrightarrow (\left\lfloor \frac{t_1+(n \bmod \mathfrak{g})}{\mathfrak{g}} \right\rfloor + m) \cdot \mathfrak{g} = (\left\lfloor \frac{t_2+(n \bmod \mathfrak{g})}{\mathfrak{g}} \right\rfloor + m) \cdot \mathfrak{g} \\
\Leftrightarrow \left\lfloor \frac{t_1+(n \bmod \mathfrak{g})}{g} \right\rfloor \cdot \mathfrak{g} + \mathfrak{g} \cdot m = \left\lfloor \frac{t_2+(n \bmod \mathfrak{g})}{\mathfrak{g}} \right\rfloor \cdot \mathfrak{g} + \mathfrak{g} \cdot m \\
\Leftrightarrow \left\lfloor \frac{t_1+(n \bmod \mathfrak{g})}{\mathfrak{g}} \right\rfloor \cdot \mathfrak{g} = \left\lfloor \frac{t_2+(n \bmod \mathfrak{g})}{\mathfrak{g}} \right\rfloor \cdot \mathfrak{g} \\
\Leftrightarrow c(t_1 + (n \bmod \mathfrak{g}))=c(t_2 + (n \bmod \mathfrak{g}))
\end{array}$$

and thus we have proved the required result.

D.3 Proof of Theorem 6.11

PROOF. Let $\mathsf{AS}_1 = (\mathsf{S}, \mathsf{E}_{\mathsf{pub}}, c_1, k)$ and $\mathsf{AS}_2 = (\mathsf{S}, \mathsf{E}_{\mathsf{pub}}, c_2, k)$ be two attack scenarios such that S is deterministic and the clocks c_1, c_2 have grains $\mathfrak{g}_1$, $\mathfrak{g}_2$ (resp.), and $\mathfrak{g}_1$ is a multiple of $\mathfrak{g}_2$. We will prove that

$$\mathsf{TC}(\mathsf{AS}_1) \preceq \mathsf{TC}(\mathsf{AS}_2)$$

Since S is deterministic then also $\mathsf{TC}(\mathsf{AS}_1)$ and $\mathsf{TC}(\mathsf{AS}_2)$ are, and thus by Theorem 6.8 in order to prove that $\mathsf{TC}(\mathsf{AS}_1) \preceq \mathsf{TC}(\mathsf{AS}_2)$ it is sufficient to show that $\mathsf{TC}(\mathsf{AS}_1) \sqsubseteq \mathsf{TC}(\mathsf{AS}_2)$.

Let i_1, $i_2 \in I$, with $i_1 \equiv_{\mathsf{TC}(\mathsf{AS}_2)} i_2$. Let also ρ_1, $\rho_2 \in \mathsf{Runs}(\mathsf{S})$ be their corresponding runs, and t'_1,...,t'_k and t''_1,...,t''_k be the k-time sequence of ρ_1 and ρ_2 resp. Since, $i_1 \equiv_{\mathsf{TC}(\mathsf{AS}_2)} i_2$ we have that the view of the adversary on the runs of them is the same, that is

$$\begin{array}{rcl}
\mathsf{view}_{c_2}(\rho_1) & = & (c_2(t'_1), c_2(t'_2), ..., c_2(t'_k)) \\
& = & \mathsf{view}_{c_2}(\rho_2) \\
& = & (c_2(t''_1), c_2(t''_2), ..., c_2(t''_k)) \quad (1)
\end{array}$$

Next, using that the grain $\mathfrak{g}_1$ of the clock c_1, is a multiple of the grain $\mathfrak{g}_2$, of the clock c_2, Fact 8 and (1) we get that

$$\begin{aligned} \mathsf{view}_{c_1}(\rho_1) &= (c_1(t'_1), c_1(t'_2), ..., c_1(t'_k)) \\ &= \mathsf{view}_{c_1}(\rho_2) \\ &= (c_1(t''_1), c_1(t''_2), ..., c_1(t''_k)) \end{aligned}$$

and this give us that $i_1 \equiv_{\mathsf{TC}(\mathsf{AS}_1)} i_2$ and thus we showed that $\mathsf{TC}(\mathsf{AS}_1) \sqsubseteq \mathsf{TC}(\mathsf{AS}_2)$ as required.

D.4 Proof of Theorem 6.12

PROOF. Let $\mathsf{S}_{\text{1-pad}}$, $\mathsf{S}_{\text{clock-edge}}$ and $\mathsf{S}_{\text{co-prime}}$ be the deterministic systems that correspond to the three scenarios $\mathsf{AS}_{\text{1-pad}}$, $\mathsf{AS}_{\text{clock-edge}}$ and $\mathsf{AS}_{\text{co-prime}}$ (resp.) Since $\mathsf{S}_{\text{1-pad}}$, $\mathsf{S}_{\text{clock-edge}}$, $\mathsf{S}_{\text{co-prime}}$ are deterministic, we also have that $\mathsf{TC}(\mathsf{AS}_{\text{1-pad}}) : I \times O_1 \mapsto [0,1]$, $\mathsf{TC}(\mathsf{AS}_{\text{clock-edge}}) : I \times O_2 \mapsto [0,1]$ and $\mathsf{TC}(\mathsf{AS}_{\text{co-prime}}) : I \times O_3 \mapsto [0,1]$ are deterministic. Therefore, using Theorem 6.8, in order to prove that

$$\mathsf{TC}(\mathsf{AS}_{\text{1-pad}}) \preceq \mathsf{TC}(\mathsf{AS}_{\text{clock-edge}}) \preceq \mathsf{TC}(\mathsf{AS}_{\text{co-prime}})$$

it is sufficient to show that

$$\mathsf{TC}(\mathsf{AS}_{\text{1-pad}}) \sqsubseteq \mathsf{TC}(\mathsf{AS}_{\text{clock-edge}}) \sqsubseteq \mathsf{TC}(\mathsf{AS}_{\text{co-prime}})$$

that is that the partition of $\mathsf{TC}(\mathsf{AS}_{\text{1-pad}})$ is refined by the one of $\mathsf{TC}(\mathsf{AS}_{\text{clock-edge}})$, and the partition of $\mathsf{TC}(\mathsf{AS}_{\text{clock-edge}})$ is refined by the one of $\mathsf{TC}(\mathsf{AS}_{\text{co-prime}})$.

We will start by showing that

$$\mathsf{TC}(\mathsf{AS}_{\text{clock-edge}}) \sqsubseteq \mathsf{TC}(\mathsf{AS}_{\text{co-prime}})$$

Let $i_1, i_2 \in I$, with their timings t_{i_1}, t_{i_2}, such that

$$i_1 \equiv_{\mathsf{TC}(\mathsf{AS}_{\text{co-prime}})} i_2$$

which means that there exists $o \in O_3$ such that

$$\mathsf{TC}(\mathsf{AS}_{\text{co-prime}})(i_1, o) = 1 = \mathsf{TC}(\mathsf{AS}_{\text{co-prime}})(i_2, o) \quad (1)$$

Next, let ρ_{i_1} and ρ_{i_2} to be the runs of the automata of the system $\mathsf{S}_{\text{co-prime}}$ which correspond to i_1 and i_2 (resp.). Expanding (1) we have that

$$\begin{aligned}\mathsf{view}_c(\rho_{i_1}) &= (c(t_{i_1}), c(t_{i_1} + t'_{\text{pad}}), ..., c(t_{i_1} + \mathfrak{g} \cdot t'_{\text{pad}}))\\ &= o\\ &= \mathsf{view}_c(\rho_{i_2})\\ &= (c(t_{i_2}), c(t_{i_2} + t'_{\text{pad}}), ..., c(t_{i_2} + \mathfrak{g} \cdot t'_{\text{pad}})) \quad (2)\end{aligned}$$

Now since t'_{pad} is co-prime with $\mathfrak{g}$, t'_{pad} is a generator of the group $(\mathbb{Z}_{\mathfrak{g}}, +)$ and thus

$$\begin{aligned}\mathbb{Z}_{\mathfrak{g}} &= \{0, 1, .., \mathfrak{g} - 1\}\\ &= \left\{0 \bmod \mathfrak{g}, t'_{\text{pad}} \bmod \mathfrak{g}, ..., (\mathfrak{g} - 1) \cdot t'_{\text{pad}} \bmod \mathfrak{g}\right\} \quad (3)\end{aligned}$$

Therefore using (3) and (2) we have that

$$\forall z \in \mathbb{Z}_{\mathfrak{g}} : c(t_{i_1} + z) = c(t_{i_2} + z) \quad (4)$$

Now using (4) we will show that $i_1 \equiv_{\mathsf{TC}(\mathsf{AS}_{\text{clock-edge}})} i_2$. Let ρ'_{i_1} and ρ'_{i_2} to be the runs of the automata of the system $\mathsf{S}_{\text{clock-edge}}$ that correspond to i_1 and i_2 (resp.). We then have that

$$\mathsf{view}_c(\rho'_{i_1}) = (c(t_{i_1}), c(t_{i_1} + t_{\text{pad}}), ..., c(t_{i_1} + m \cdot t_{\text{pad}}))$$

and

$$\mathsf{view}_c(\rho'_{i_1}) = (c(t_{i_2}), c(t_{i_2} + t_{\text{pad}}), ..., c(t_{i_2} + m \cdot t_{\text{pad}}))$$

where m is the number of paddings needed for the clock-edge technique. Using Fact 9, we have that for proving $\mathsf{view}_c(\rho'_{i_1}) = \mathsf{view}_c(\rho'_{i_1})$ it is sufficient to show that

$$\begin{aligned}&\forall z \in \{0 \bmod \mathfrak{g}, t_{\text{pad}} \bmod \mathfrak{g}, ..., (m \cdot t_{\text{pad}}) \bmod \mathfrak{g}\} :\\ &\quad c(t_{i_1} + z) = c(t_{i_2} + z) \ (6)\end{aligned}$$

However, since

$$\{0 \bmod \mathfrak{g}, t_{\text{pad}} \bmod \mathfrak{g}, ..., (m \cdot t_{\text{pad}}) \bmod \mathfrak{g}\} \subseteq \mathbb{Z}_{\mathfrak{g}}$$

and by (4), we have that (6) holds and we have proved that

$$\mathsf{TC}(\mathsf{AS}_{\text{clock-edge}}) \sqsubseteq \mathsf{TC}(\mathsf{AS}_{\text{co-prime}})$$

Finally, we will show that $\mathsf{TC}(\mathsf{AS}_{\text{1-pad}}) \sqsubseteq \mathsf{TC}(\mathsf{AS}_{\text{clock-edge}})$. Let i_1, $i_2 \in I$, with times t_{i_1} and t_{i_2} (resp.), such that $i_1 \equiv_{\mathsf{TC}(\mathsf{AS}_2)} i_2$, which means that there exists $o \in O_2$ such that $\mathsf{TC}(\mathsf{AS}_{\text{clock-edge}})(i_1, o) = 1 = \mathsf{TC}(\mathsf{AS}_{\text{clock-edge}})(i_2, o)$. Next let ρ_{i_1}, ρ_{i_2} to be the runs of the automata of the system $\mathsf{S}_{\text{clock-edge}}$ that correspond to i_1 and i_2 (resp.), and ρ'_{i_1}, ρ'_{i_2} to be the runs of the automata of the system $\mathsf{S}_{\text{1-pad}}$ that correspond to i_1 and i_2 (resp.). By our assumptions we have that

$$\begin{aligned}\mathsf{view}_c(\rho_{i_1}) &= (c(t_{i_1}), c(t_{i_1} + t_{\text{pad}}), ..., c(t_{i_1} + m \cdot t_{\text{pad}}))\\ &= o\\ &= \mathsf{view}_c(\rho_{i_2})\\ &= (c(t_{i_2}), c(t_{i_2} + t_{\text{pad}}), ..., c(t_{i_2} + m \cdot t_{\text{pad}}))\end{aligned}$$

for m being the padding needed for the clock-edge technique, and thus we also have that

$$\begin{aligned}\mathsf{view}_c(\rho'_{i_1}) &= (c(t_{i_1}), c(t_{i_1} + t_{\text{pad}}))\\ &= (c(t_{i_2}), c(t_{i_2} + t_{\text{pad}}))\\ &= \mathsf{view}_c(\rho'_{i_2})\end{aligned}$$

which give us that $i_1 \equiv_{\mathsf{TC}(\mathsf{AS}_{\text{1-pad}})} i_2$.

Therefore we can conclude that

$$\mathsf{TC}(\mathsf{AS}_{\text{1-pad}}) \sqsubseteq \mathsf{TC}(\mathsf{AS}_{\text{clock-edge}})$$

and this completes the proof.

APPENDIX E

Details of the Case Study 6.6

We provide some more details with regards to the calculations presented in Section 6.6.

In particular, we show **Step 1**, and **Step 2** of the algorithm in Table 6.1 for an arbitrary key k, a clock c_l with $\mathfrak{g} \in [1, 1000]$ and limit $l > 15000$.

Let $e_1 = (q_\circ, \mathtt{r} = t_{enc(\mathsf{k})} \rightarrow, q)$ be the edge that corresponds to the encryption, $e_2 = (q, \mathtt{r} = 1 \rightarrow, q')$,...,$e_{11} = (q, \mathtt{r} = 10 \rightarrow, q')$ the edges that correspond to the noise, and $e_{12} = (q', \rightarrow \mathtt{r}, q_\circ)$ to be the edge of the communication.

For the initial configuration $\gamma_{q_\circ}$ we have the Dirac's distribution $\mu_{\gamma_{q_\circ}}$, where for a Borel set A we have

$$\mu_{\gamma_{q_\circ}}(A) = \delta_{t_{enc}(\mathsf{k})}(A) = \begin{cases} 1 & \text{if } t_{enc}(\mathsf{k}) \in A \\ 0 & \text{otherwise} \end{cases}$$

For a configuration γ_q of the location q, we have a discrete uniform probability μ_{γ_q} over the set $1, ..., 10$ given by the probability mass function

$$p(t) = \frac{1}{10} \cdot 1_{\{1,...,10\}}(t)$$

Finally, for the configuration $\gamma_{q'}$ of the location q' we have an exponential distribution $\mu_{\gamma_{q'}}$ given by the density function

$$f(t) = 6 \cdot \mathsf{exp}(-6 \cdot t) \cdot 1_{[0,+\infty)}(t)$$

Next, for any $t_1 \in \mathsf{Int}(\gamma_\circ) = \{t_{enc(\mathsf{k})}\}$, $t_2 \in \mathsf{Int}(\gamma_q,) = \{1, ..., 10\}$, and $t_3 \in \mathsf{Int}(\gamma_{q'}) = [0, +\infty)$ we have $\kappa_{\gamma_{q_\circ}+t_1}(e_1) = 1$, $\kappa_{\gamma_q+t_2}(e) = 1$ (if $t_2 = z$ and $e = e_{z+1}$) and $\kappa_{\gamma_{q'}+t_3}(e_{12}) = 1$ respectively.

Regarding **Step 1**, we have that the possible observations of the adversary for the input k, is given by the set

$$O_\mathsf{k} = \{c(t_{enc(\mathsf{k})} + 1), c(t_{enc(\mathsf{k})} + 1) + g, ..., l\}$$

Next, we need to compute the probabilities of those outputs (**Step 2**). We start by computing the 1-observable (i.e $k = 1$) paths of the automaton and we get

$$\textbf{Paths} = \{e_1e_2e_{12}, e_1e_3e_{12}, ..., e_1e_{11}e_{12}\}$$

Therefore for an observation $o \in O_\mathsf{k}$ we have that

$$\mathsf{view}_{c_l}(o) = \bigcup_{\pi \in \textbf{Paths}} \mathsf{Cyl}_{\mathcal{C}_\pi(o)}(\gamma_{q_\circ}, \pi)$$

and thus

$$\mathsf{P}_{\gamma_{q_\circ}}(\mathsf{view}_{c_l}^{-1}(o)) = \sum_{\pi \in \textbf{Paths}} \mathsf{P}_{\gamma_{q_\circ}}(\mathsf{Cyl}_{\mathcal{C}_\pi(o)}(\gamma_{q_\circ}, \pi))$$

For an observation $o \in O_\mathsf{k}$, and a path $\pi \in$ **Paths** we will show how we compute the probability

$$\mathsf{P}_{\gamma_{q_\circ}}(\mathsf{Cyl}_{\mathcal{C}_\pi(o)}(\gamma_{q_\circ}, \pi))$$

We distinguish the following cases $o < l$ and $o = l$.

Case (a) For $o < l$, and $\pi = e_1e_ze_{12} \in$ **Paths**, where $z \in \{2, ..., 11\}$, we have that

$$\mathcal{C}_\pi(o) = \{(t_1, t_2, t_3) \in \mathbb{R}^3_{\geq 0} \mid o \leq t_1 + t_2 + t_3 < o + g\}$$

and

$$\mathsf{P}_{\gamma_{q_\circ}}(\mathsf{Cyl}_{\mathcal{C}_\pi(o)}(\gamma_{q_\circ}, \pi))$$

is equal to

$$\int_{t_1 \in \mathsf{Int}(\gamma_{q_\circ}, e_1)} \kappa_{\gamma_{q_\circ}+t_1}(e_1) \cdot \mathsf{P}_{\gamma_q}(\mathsf{Cyl}_{\mathcal{C}_\pi^{t_1}(o)}(\gamma_q, \pi(1)))d\mu_{\gamma_{q_\circ}}(t_1)$$

where $\pi(1) = e_ze_{12}$. Since we integrate with respect to a Dirac's distribution over $t_{enc(\mathsf{k})}$, we have that the previous integral is equal to

$$\mathsf{P}_{\gamma_q}(\mathsf{Cyl}_{\mathcal{C}_\pi^{t_{enc(\mathsf{k})}}(o)}(\gamma_q, \pi(1))) \quad (1)$$

Next let $\mathcal{C}_\pi^{t_{enc(\mathsf{k})}}(o) = \mathcal{C}_1$ and then (1) is equal to

$$
\begin{aligned}
&\int_{t_2 \in \mathsf{Int}(\gamma_q, e_z)} \kappa_{\gamma_q + t}(e_z) \cdot \mathsf{P}_{\gamma_{q'}}(\mathsf{Cyl}_{\mathcal{C}_1^{t_2}}(\gamma_{q'}, e_{12})) d\mu_{\gamma_q}(t_2) \\
&= p(z-1) \cdot \mathsf{P}_{\gamma_{q'}}(\mathsf{Cyl}_{\mathcal{C}_1^{z-1}}(\gamma_{q'}, e_{12})) \\
&= \frac{1}{10} \cdot \mathsf{P}_{\gamma_{q'}}(\mathsf{Cyl}_{\mathcal{C}_1^{z-1}}(\gamma_{q'}, e_{12})) \\
&= \frac{1}{10} \cdot \int_{t \in \mathsf{Int}(\gamma_{q'}, e_{12})} \kappa_{\gamma_{q'} + t}(e_{12}) \cdot 1_{\mathcal{C}_1^{z-1}}(t) d\mu_{\gamma_{q'}}(t) \\
&= \frac{1}{10} \cdot \int_{t \in [0, +\infty)} 6 \cdot \mathsf{exp}(-6 \cdot t) \cdot 1_{[0,+\infty)}(t) \cdot 1_{\mathcal{C}_1^{z-1}}(t) dt \quad (2)
\end{aligned}
$$

We next distinguish between three subcases based on the constraint

$$\mathcal{C}_1^{z-1} = [o - (t_{enc(\mathsf{k})} + z - 1), o + g - (t_{enc(\mathsf{k})} + z - 1))$$

First, let

$$L = o - (t_{enc(\mathsf{k})} + z - 1)$$

and

$$U = o + g - (t_{enc(\mathsf{k})} + z - 1)$$

Now if $L < 0$, and $U > 0$, (2) is

$$
\begin{aligned}
&\frac{1}{10} \cdot \int_{t \in [0, U)} 6 \cdot \mathsf{exp}(-6 \cdot t) dt \\
&= \frac{1}{10} \cdot (-\mathsf{exp}(-6 \cdot U) + 1)
\end{aligned}
$$

Next, if $L \geq 0$, and $U > 0$, (2) is

$$
\begin{aligned}
&\frac{1}{10} \cdot \int_{t \in [L, U)} 6 \cdot \mathsf{exp}(-6 \cdot t) dt \\
&= \frac{1}{10} \cdot (-\mathsf{exp}(-6 \cdot U) + \mathsf{exp}(-6 \cdot L))
\end{aligned}
$$

Otherwise, when $U \leq 0$ (2) is equal to 0.

Case (b) For $o = l$, and $\pi = e_1 e_z e_{12} \in$ **Paths** where $z \in \{2, ..., 11\}$ we have that

$$\mathcal{C}_\pi(l) = \left\{(t_1, t_2, t_3) \in \mathbb{R}^3_{\geq 0} \mid t_1 + t_2 + t_3 \geq l\right\}$$

and

$$\mathsf{P}_{\gamma_{q_\circ}}(\mathsf{Cyl}_{\mathcal{C}_\pi(l)}(\gamma_{q_\circ}, \pi))$$

is equal to

$$\int_{t_1 \in \mathsf{Int}(\gamma_{q_\circ}, e_1)} \kappa_{\gamma_{q_\circ}+t_1}(e_1) \cdot \mathsf{P}_{\gamma_q}(\mathsf{Cyl}_{\mathcal{C}_\pi^{t_1}(l)}(\gamma_q, \pi(1)))d\mu_{\gamma_{q_\circ}}(t_1)$$

where $\pi(1) = e_z e_{12}$. Since we integrate with respect to a Dirac's distribution over $t_{enc(\mathsf{k})}$, we have that the previous integral is equal to

$$\mathsf{P}_{\gamma_q}(\mathsf{Cyl}_{\mathcal{C}_\pi^{t_{enc(\mathsf{k})}}(l)}(\gamma_q, \pi(1))) \quad (1)$$

Next, let $\mathcal{C}_\pi^{t_{enc(\mathsf{k})}}(l) = \mathcal{C}_1$ and then (1) is equal to

$$\begin{aligned}
&\int_{t_2 \in \mathsf{Int}(\gamma_q, e_z)} \kappa_{\gamma_q+t}(e_z) \cdot \mathsf{P}_{\gamma_{q'}}(\mathsf{Cyl}_{\mathcal{C}_1^{t_2}}(\gamma_{q'}, e_{12}))d\mu_{\gamma_q}(t_2)\\
=\,& p(z-1) \cdot \mathsf{P}_{\gamma_{q'}}(\mathsf{Cyl}_{\mathcal{C}_1^{z-1}}(\gamma_{q'}, e_{12}))\\
=\,& \frac{1}{10} \cdot \mathsf{P}_{\gamma_{q'}}(\mathsf{Cyl}_{\mathcal{C}_1^{z-1}}(\gamma_{q'}, e_{12}))\\
=\,& \frac{1}{10} \cdot \int_{t \in \mathsf{Int}(\gamma_{q'}, e_{12})} \kappa_{\gamma_{q'}+t}(e_{12}) \cdot 1_{\mathcal{C}_1^{z-1}}(t)d\mu_{\gamma_{q'}}(t)\\
=\,& \frac{1}{10} \cdot \int_{t \in [0,+\infty)} 6 \cdot \mathsf{exp}(-6 \cdot t) \cdot 1_{[0,+\infty)}(t) \cdot 1_{\mathcal{C}_1^{z-1}}(t)dt \quad (2)
\end{aligned}$$

Next, we distinguish between two subcases based on the constraint

$$\mathcal{C}_1^{z-1} = [l - (t_{enc(\mathsf{k})} + z - 1), +\infty)$$

First, let

$$L = l - (t_{enc(\mathsf{k})} + z - 1)$$

Now if $L < 0$, (2) is

$$\begin{aligned}
&\frac{1}{10} \cdot \int_{t \in [0,+\infty)} 6 \cdot \mathsf{exp}(-6 \cdot t)dt\\
=\,& \frac{1}{10}
\end{aligned}$$

Otherwise, if $L \geq 0$, (2) is

$$
\begin{aligned}
&\frac{1}{10} \cdot \int_{t \in [L,+\infty)} 6 \cdot \mathsf{exp}(-6 \cdot t) dt \\
&= \frac{1}{10} \cdot \lim_{n \to \infty} [-\mathsf{exp}(-6 \cdot t)]_L^n \\
&= \frac{1}{10} \cdot \lim_{n \to \infty} -\mathsf{exp}(-6 \cdot n) + \mathsf{exp}(-6 \cdot L) \\
&= \frac{1}{10} \cdot \mathsf{exp}(-6 \cdot L)
\end{aligned}
$$

Bibliography

[ABB$^+$16] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, François Dupressoir, and Michael Emmi. Verifying constant-time implementations. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 53–70, 2016.

[ACD93] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking in dense real-time. *Inf. Comput.*, 104(1):2–34, 1993.

[ACPS12] Mário S. Alvim, Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Geoffrey Smith. Measuring information leakage using generalized gain functions. In *25th IEEE Computer Security Foundations Symposium, CSF 2012, Cambridge, MA, USA, June 25-27, 2012*, pages 265–279, 2012.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[Aga00] Johan Agat. Transforming out timing leaks. In *Proc. POPL*, pages 40–53, 2000.

[AILS07] Luca Aceto, Anna Ingolfsdottir, Kim Guldstrand Larsen, and Jiri Srba. *Reactive Systems: Modelling, Specification and Verification*. Cambridge University Press, 2007.

[AKM$^+$15] Marc Andrysco, David Kohlbrenner, Keaton Mowery, Ranjit Jhala, Sorin Lerner, and Hovav Shacham. On subnormal floating point and abnormal timing. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 623–639, 2015.

[Apt81] Krzysztof R. Apt. Ten years of Hoare's logic: A survey – part 1. *ACM Trans. Program. Lang. Syst.*, 3(4):431–483, 1981.

[ARF16] Omar I. Al-Bataineh, Mark Reynolds, and Tim French. Finding minimum and maximum termination time of timed automata models with cyclic behaviour. *CoRR*, abs/1610.09795, 2016.

[ARF17] Omar I. Al-Bataineh, Mark Reynolds, and Tim French. Finding minimum and maximum termination time of timed automata models with cyclic behaviour. *Theor. Comput. Sci.*, 665:87–104, 2017.

[AS07] Aslan Askarov and Andrei Sabelfeld. Localized delimited release: combining the what and where dimensions of information release. In *Proceedings of the 2007 Workshop on Programming Languages and Analysis for Security, PLAS 2007, San Diego, California, USA, June 14, 2007*, pages 53–60, 2007.

[AS19] Étienne André and Jun Sun. Parametric timed model checking for guaranteeing timed opacity. *CoRR*, abs/1907.00537, 2019.

[ATM10] Ravi Akella, Han Tang, and Bruce M. McMillin. Analysis of information flow security in cyber-physical systems. *IJCIP*, 3(3-4):157–173, 2010.

[ATT19] Cyber attack trends analysis. 1, 2019.

[BB93] Jean-Pierre Banâtre and Ciarán Bryce. Information flow control in a parallel language framework. In *6th IEEE Computer Security Foundations Workshop - CSFW'93, Franconia, New Hampshire, USA, June 15-17, 1993, Proceedings*, pages 39–52, 1993.

[BBB$^+$14] Nathalie Bertrand, Patricia Bouyer, Thomas Brihaye, Quentin Menet, Christel Baier, Marcus Größer, and Marcin Jurdzinski. Stochastic timed automata. *Logical Methods in Computer Science*, 10(4), 2014.

[BBF01] Elisa Bertino, Piero A. Bonatti, and Elena Ferrari. Trbac: A temporal role-based access control model. *ACM Trans. Inf. Syst. Secur.*, 4(3):191–233, 2001.

[BBM94] Jean-Pierre Banatre, Ciaran Bryce, and Daniel Le Metayer. Compile-time detection of information flow in sequential programs. In *Proc. ESORICS*, pages 55–73, 1994.

[BBM95] Ciarán Bryce, Jean-Pierre Banâtre, and Daniel Le Métayer. An approach to information security in distributed systems. In *5th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS 1995), August 28-30, 1995, Chenju, Korea, Proceedings*, pages 384–394, 1995.

[BDK13] Michael Backes, Goran Doychev, and Boris Köpf. Preventing side-channel leaks in web traffic: A formal approach. In *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013*, 2013.

[BFG10] Moritz Y. Becker, Cédric Fournet, and Andrew D. Gordon. Secpal: Design and semantics of a decentralized authorization language. *Journal of Computer Security*, 18(4):619–665, 2010.

[BFST02] Roberto Barbuti, Nicoletta De Francesco, Antonella Santone, and Luca Tesei. A notion of non-interference for timed automata. *Fundam. Inform.*, 51(1-2):1–11, 2002.

[BHKZ11] David A. Basin, Matús Harvan, Felix Klaedtke, and Eugen Zalinescu. Monitoring usage-control policies in distributed systems. *TIME*, pages 88–95, 2011.

[BK15] Michael Backes and Boris Köpf. Quantifying information flow in cryptographic systems. *Mathematical Structures in Computer Science*, 25(2):457–479, 2015.

[BMP19] M Balliu, Merro M., and M. Pasqua. Securing cross-app interactions in iot platforms. *IEEE Computer Security Foundations Symposium.*, 2019.

[BP18] Brandon Bohrer and André Platzer. A hybrid, dynamic logic for hybrid-dynamic information flow. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 115–124, 2018.

[BSJ93] Elisa Bertino, Pierangela Samarati, and Sushil Jajodia. Authorizations in relational database management systems. In *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993.*, pages 130–139, 1993.

[BT03] Roberto Barbuti and Luca Tesei. A decidable notion of timed non-interference. *Fundam. Inform.*, 54(2-3):137–150, 2003.

[BT11] Billy Bob Brumley and Nicola Tuveri. Remote timing attacks are still practical. In *Computer Security - ESORICS 2011 - 16th European Symposium on Research in Computer Security, Leuven, Belgium, September 12-14, 2011. Proceedings*, pages 355–371, 2011.

[Car17] Pierre Carlier. *Verification of Stochastic Timed Automata. (Vérification des automates temporisés et stochastiques).* PhD thesis, University of Paris-Saclay, France, 2017.

[Cas09] Franck Cassez. The dark side of timed opacity. In *Advances in Information Security and Assurance, Third International Conference and Workshops, ISA 2009, Seoul, Korea, June 25-27, 2009. Proceedings*, pages 21–30, 2009.

[CHM05] David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantified interference for a while language. *Electr. Notes Theor. Comput. Sci.*, 112:149–166, 2005.

[CRS83] David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors. *Advances in Cryptology: Proceedings of CRYPTO '82, Santa Barbara, California, USA, August 23-25, 1982*. Plenum Press, New York, 1983.

[CT06] Thomas M. Cover and Joy A. Thomas. *Elements of information theory (2. ed.)*. Wiley, 2006.

[CWW$^+$10] Hsing-Chung Chen, Shiuh-Jeng Wang, Jyh-Horng Wen, Yung-Fa Huang, and Chung-Wei Chen. A generalized temporal and spatial role-based access control model. *JNW*, 5(8):912–920, 2010.

[DD77] Dorothy E. Denning and Peter J. Denning. Certification of programs for secure information flow. *Commun. ACM*, 20(7):504–513, 1977.

[Den76] Dorothy E. Denning. A lattice model of secure information flow. *Commun. ACM*, 19(5):236–243, 1976.

[Den82] Dorothy E. Denning. *Cryptography and Data Security*. Addison-Wesley, 1982.

[DFK$^+$13] Goran Doychev, Dominik Feld, Boris Köpf, Laurent Mauborgne, and Jan Reineke. Cacheaudit: A tool for the static analysis of cache side channels. In *Proceedings of the 22th USENIX Security Symposium, Washington, DC, USA, August 14-16, 2013*, pages 431–446, 2013.

[DGP08] Henry DeYoung, Deepak Garg, and Frank Pfenning. An authorization logic with explicit time. *CSF*, pages 143–165, 2008.

[Dij75] Edsger W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Commun. ACM*, 18(8):453–457, 1975.

[DLL$^+$15] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, and Danny Bøgsted Poulsen. Uppaal SMC tutorial. *STTT*, 17(4):397–415, 2015.

[dMB08] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software,* *ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, pages 337–340, 2008.

[dVSS14] Sabrina De Capitani di Vimercati, Pierangela Samarati, and Ravi Sandhu. Access control. *Computing Handbook, 3rd ed.*, 1:47:1–25, 2014.

[Fag78] Ronald Fagin. On an authorization mechanism. *ACM Trans. Database Syst.*, 3(3):310–319, 1978.

[FGM03] Riccardo Focardi, Roberto Gorrieri, and Fabio Martinelli. Real-time information flow analysis. *IEEE Journal on Selected Areas in Communications*, 21(1):20–35, 2003.

[Fra97] M. Franchella. On the origins of Dénes König's infinity lemma. *Archive for History of Exact Sciences*, 51:3–27, 1997.

[FS00] Edward W. Felten and Michael A. Schneider. Timing attacks on web privacy. In *CCS 2000, Proceedings of the 7th ACM Conference on Computer and Communications Security, Athens, Greece, November 1-4, 2000.*, pages 25–32, 2000.

[GA17] B. B. Gupta and Tafseer Akhtar. A survey on smart power grid: frameworks, tools, security issues, and solutions. *Annales des Télécommunications*, 72(9-10):517–549, 2017.

[GBO12] Emsaieb Geepalla, Behzad Bordbar, and Kozo Okano. Verification of spatio-temporal role based access control using timed automata. *NESEA*, pages 1–6, 2012.

[GH02] James Giles and Bruce E. Hajek. An information-theoretic and game-theoretic study of timing channels. *IEEE Trans. Information Theory*, 48(9):2455–2477, 2002.

[GLMS14] Sylvia Grewe, Alexander Lux, Heiko Mantel, and Jens Sauer. A formalization of declassification with what-and-where-security. *Archive of Formal Proofs*, 2014, 2014.

[GM82] Joseph A. Goguen and José Meseguer. Security policies and security models. In *1982 IEEE Symposium on Security and Privacy, Oakland, CA, USA, April 26-28, 1982*, pages 11–20, 1982.

[GMR07] Guillaume Gardey, John Mullins, and Olivier H. Roux. Non-interference control synthesis for security timed automata. *Electr. Notes Theor. Comput. Sci.*, 180(1):35–53, 2007.

[GSB18] Christopher Gerking, David Schubert, and Eric Bodden. Model checking the information flow security of real-time systems. In *Engineering Secure Software and Systems - 10th International Symposium, ESSoS 2018, Paris, France, June 26-27, 2018, Proceedings*, pages 27–43, 2018.

[HHD08] Yong-Zhong He, Zhen Han, and Ye Du. Context active rbac and its applications. *ISECS*, pages 1041–1044, 2008.

[HLH14] Xuezhen Huang, Jiqiang Liu, and Zhen Han. A privacy-aware access model on anonymized data. *INTRUST*, pages 201–212, 2014.

[HSW12] Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Efficient emptiness check for timed büchi automata. *Formal Methods in System Design*, 40(2):122–146, 2012.

[Hu91] Wei-Ming Hu. Reducing timing channels with fuzzy time. In *IEEE Symposium on Security and Privacy*, pages 8–20, 1991.

[Hu92] Wei-Ming Hu. Reducing timing channels with fuzzy time. *Journal of Computer Security*, 1(3-4):233–254, 1992.

[III90] James W. Gray III. Probabilistic interference. In *Proceedings of the 1990 IEEE Symposium on Security and Privacy, Oakland, California, USA, May 7-9, 1990*, pages 170–179, 1990.

[JBLG05] James Joshi, Elisa Bertino, Usman Latif, and Arif Ghafoor. A generalized temporal role-based access control model. *IEEE Trans. Knowl. Data Eng.*, 17(1):4–23, 2005.

[JW02] David N. Jansen and Roel Wieringa. Extending ctl with actions and real time. *J. Log. Comput.*, 12(4):607–621, 2002.

[KB07] Boris Köpf and David A. Basin. An information-theoretic model for adaptive side-channel attacks. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, pages 286–296, 2007.

[KD09] Boris Köpf and Markus Dürmuth. A provably secure and efficient countermeasure against timing attacks. *IACR Cryptology ePrint Archive*, 2009:89, 2009.

[Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, pages 104–113, 1996.

[KS16a] M. Fahim Ferdous Khan and Ken Sakamura. A discretionary delegation framework for access control systems. *OTM Conferences*, pages 865–882, 2016.

[KS16b] David Kohlbrenner and Hovav Shacham. Trusted browsers for uncertain times. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 463–480, 2016.

[Lam74] Butler W. Lampson. Protection. *Operating Systems Review*, 8(1):18–24, 1974.

[LJC16] Robert M. Lee, Michael J., and Tim Conway. Analysis of the cyber attack on the ukrainian power grid. defense use case. *E-ISAC.*, 2016.

[LMMV19] Ruggero Lanotte, Massimo Merro, Andrei Munteanu, and Luca Viganò. A formal approach to physics-based attacks in cyber-physical systems (extended version). *CoRR*, abs/1902.04572, 2019.

[LMP12] Alexander Lux, Heiko Mantel, and Matthias Perner. Scheduler-independent declassification. In *Mathematics of Program Construction - 11th International Conference, MPC 2012, Madrid, Spain, June 25-27, 2012. Proceedings*, pages 25–47, 2012.

[LMST10] Ruggero Lanotte, Andrea Maggiolo-Schettini, and Angelo Troina. Time and probability-based information flow analysis. *IEEE Trans. Software Eng.*, 36(5):719–734, 2010.

[LNN15] Ximeng Li, Flemming Nielson, and Hanne Riis Nielson. Factorization of behavioral integrity. In *Computer Security - ESORICS 2015 - 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part II*, pages 500–519, 2015.

[LNNF15] Ximeng Li, Flemming Nielson, Hanne Riis Nielson, and Xinyu Feng. Disjunctive information flow for communicating processes. In *Trustworthy Global Computing - 10th International Symposium, TGC 2015, Madrid, Spain, August 31 - September 1, 2015 Revised Selected Papers*, pages 95–111, 2015.

[LT79] Thomas Lengauer and Robert Endre Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Trans. Program. Lang. Syst.*, 1(1):121–141, 1979.

[McL90] John McLean. Security models and information flow. In *Proceedings of the 1990 IEEE Symposium on Security and Privacy, Oakland, California, USA, May 7-9, 1990*, pages 180–189, 1990.

[MKP$^+$18] Pasquale Malacaria, M. H. R. Khouzani, Corina S. Pasareanu, Quoc-Sang Phan, and Kasper Søe Luckow. Symbolic side-channel analysis for probabilistic programs. In *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*, pages 313–327, 2018.

[ML97] Andrew C. Myers and Barbara Liskov. A decentralized model for information flow control. In *ACM Symposium on Operating System Principles, SOSP 1997*, pages 129–142. ACM, 1997.

[MPTB12] Kevin Mueller, Michael Paulitsch, Sergey Tverdyshev, and Holger Blasum. Mils-related information flow control in the avionic domain: A view on security-enhancing software architectures. In *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, DSN 2012, Boston, MA, USA, June 25-28, 2012*, pages 1–6, 2012.

[MR17] Bruce M. McMillin and Thomas P. Roth. *Cyber-Physical Security and Privacy in the Electric Smart Grid.* Synthesis Lectures on Information Security, Privacy, and Trust. Morgan & Claypool Publishers, 2017.

[MS04] Heiko Mantel and David Sands. Controlled declassification based on intransitive noninterference. In *Programming Languages and Systems: Second Asian Symposium, APLAS 2004, Taipei, Taiwan, November 4-6, 2004. Proceedings*, pages 129–145, 2004.

[MS07] Heiko Mantel and Henning Sudbrock. Comparing countermeasures against interrupt-related covert channels in an information-theoretic framework. In *20th IEEE Computer Security Foundations Symposium, CSF 2007, 6-8 July 2007, Venice, Italy*, pages 326–340, 2007.

[MS08] Samrat Mondal and Shamik Sural. Security analysis of temporal-rbac using timed automata. *IAS*, pages 37–40, 2008.

[MSA11] Samrat Mondal, Shamik Sural, and Vijayalakshmi Atluri. Security analysis of gtrbac and its variants using model checking. *Computers and Security*, 30(2-3):128–147, 2011.

[MSZ04] Andrew C. Myers, Andrei Sabelfeld, and Steve Zdancewic. Enforcing robust declassification. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*, pages 172–186, 2004.

[MSZ06] Andrew C. Myers, Andrei Sabelfeld, and Steve Zdancewic. Enforcing robust declassification and qualified robustness. *Journal of Computer Security*, 14(2):157–196, 2006.

[MZZ$^+$06] Andrew C. Myers, Lantian Zheng, Steve Zdancewic, Stephen Chong, and Nathaniel Nystrom. Jif 3.0: Java information flow, July 2006.

[NN19] Flemming Nielson and Hanne Riis Nielson. Lightweight information flow. In *Models, Languages, and Tools for Concurrent and Distributed Programming - Essays Dedicated to Rocco De Nicola on the Occasion of His 65th Birthday*, pages 455–470, 2019.

[NNV17] Flemming Nielson, Hanne Riis Nielson, and Panagiotis Vasilikos. Information flow for timed automata. In *Models, Algorithms, Logics and Tools - Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday*, pages 3–21, 2017.

[OKSK15] Yossef Oren, Vasileios P. Kemerlis, Simha Sethumadhavan, and Angelos D. Keromytis. The spy in the sandbox: Practical cache attacks in javascript and their implications. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 1406–1418, 2015.

[OSM00] Sylvia L. Osborn, Ravi S. Sandhu, and Qamar Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Trans. Inf. Syst. Secur.*, 3(2):85–106, 2000.

[PHW08] Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky. Quantifying timing leaks and cost optimisation. In *Information and Communications Security, 10th International Conference, ICICS 2008, Birmingham, UK, October 20-22, 2008, Proceedings*, pages 81–96, 2008.

[Pin95] Sylvan Pinsky. Absorbing covers and intransitive non-interference. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy, Oakland, California, USA, May 8-10, 1995*, pages 102–113, 1995.

[PSL$^+$15] Martin Leth Pedersen, Michael Hedegaard Sørensen, Daniel Lux, Ulrik Nyman, and René Rydhof Hansen. The timed decentralised label model. *NordSec*, pages 27–43, 2015.

[RG99] A. W. Roscoe and M. H. Goldsmith. What is intransitive noninterference? In *Proceedings of the 12th IEEE Computer Security Foundations Workshop, CSFW 1999, Mordano, Italy, June 28-30, 1999*, pages 228–238, 1999.

[RZFG01] Carlos Ribeiro, Andre Zuquete, Paulo Ferreira, and Paulo Guedes. Spl: An access control language for security policies and complex constraints. *NDSS*, 2001.

[Rn61] Alfrd Rnyi. On measures of entropy and information. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*, pages 547–561, Berkeley, Calif., 1961. University of California Press.

[SAU19] A cyberattack in saudi arabia had a deadly goal. experts fear another try. 2019.

[SF15] Ingmar Baumgart Sören Finster. Privacy-aware smart metering: A survey. *IEEE Communications Surveys and Tutorials*, 17(2):1088–1101, 2015.

[Sha01] Claude E. Shannon. A mathematical theory of communication. *Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[SI95] Paul F. Syverson and James W. Gray III. The epistemic representation of information flow security in probabilistic systems. In *The Eighth IEEE Computer Security Foundations Workshop (CSFW '95), March 13-15, 1995, Kenmare, County Kerry, Ireland*, pages 152–166, 1995.

[SM03] Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003.

[SMGM17] Michael Schwarz, Clémentine Maurice, Daniel Gruss, and Stefan Mangard. Fantastic timers and where to find them: High-resolution microarchitectural attacks in javascript. In *Financial Cryptography and Data Security - 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers*, pages 247–267, 2017.

[Smi09] Geoffrey Smith. On the foundations of quantitative information flow. In *Foundations of Software Science and Computational Structures, 12th International Conference, FOSSACS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, pages 288–302, 2009.

[SS00] Andrei Sabelfeld and David Sands. Probabilistic noninterference for multi-threaded programs. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop, CSFW '00, Cambridge, England, UK, July 3-5, 2000*, pages 200–214, 2000.

[SS09] Andrei Sabelfeld and David Sands. Declassification: Dimensions and principles. *Journal of Computer Security*, 17(5):517–548, 2009.

[SS17] David M. Smith and Geoffrey Smith. Tight bounds on information leakage from repeated independent runs. In *30th IEEE Computer Security Foundations Symposium, CSF 2017, Santa Barbara, CA, USA, August 21-25, 2017*, pages 318–327, 2017.

[SV98] Geoffrey Smith and Dennis M. Volpano. Secure information flow in a multi-threaded imperative language. In *POPL '98, Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Diego, CA, USA, January 19-21, 1998*, pages 355–364, 1998.

[SWT01] Dawn Xiaodong Song, David A. Wagner, and Xuqing Tian. Timing analysis of keystrokes and timing attacks on SSH. In *10th USENIX Security Symposium, August 13-17, 2001, Washington, D.C., USA*, 2001.

[Thu09] Bhavani Thuraisingham. *Mandatory Access Control*, pages 1684–1685. Springer US, Boston, MA, 2009.

[UPP] UPPALL. `http://www.uppaal.com/index.php?sida=200&rubrik=95`.

[VDS11] Bhanu C. Vattikonda, Sambit Das, and Hovav Shacham. Eliminating fine grained timers in xen. In *Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW 2011, Chicago, IL, USA, October 21, 2011*, pages 41–46, 2011.

[VK17] Pepe Vila and Boris Köpf. Loophole: Timing attacks on shared event loops in chrome. In *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017.*, pages 849–864, 2017.

[VNN17] Panagiotis Vasilikos, Flemming Nielson, and Hanne Riis Nielson. Time dependent policy-based access control. In *24th International Symposium on Temporal Representation and Reasoning, TIME 2017, October 16-18, 2017, Mons, Belgium*, pages 21:1–21:18, 2017.

[VNN18] Panagiotis Vasilikos, Flemming Nielson, and Hanne Riis Nielson. Secure information release in timed automata. In *Principles of Security and Trust - 7th International Conference, POST 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, pages 28–52, 2018.

[VNNK] Panagiotis Vasilikos, Flemming Nielson, Hanne Riis Nielson, and Boris Koepf. Timing leaks and coarse-grained clocks. In *32nd IEEE Computer Security Foundations Symposium, CSF 2019.*

[VSI96] Dennis M. Volpano, Geoffrey Smith, and Cynthia E. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(2/3):167–188, 1996.

[Wra91] John C. Wray. An analysis of covert timing channels. In *IEEE Symposium on Security and Privacy*, pages 2–7, 1991.

[XAC] OASIS eXtensible Access Control Markup Language. `https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml`.

[XML] eXtensible Markup Language(XML) . `https://www.w3.org/XML/`.

[ZAM11] Danfeng Zhang, Aslan Askarov, and Andrew C. Myers. Predictive mitigation of timing channels in interactive systems. In *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, USA, October 17-21, 2011*, pages 563–574, 2011.

[ZM01] Steve Zdancewic and Andrew C. Myers. Robust declassification. In *14th IEEE Computer Security Foundations Workshop (CSFW-14 2001), 11-13 June 2001, Cape Breton, Nova Scotia, Canada*, pages 15–23, 2001.

[ZYL14] Wenrong Zeng, Yuhao Yang, and Bo Luo. Content-based access control: Use data content to assist access control for large-scale content-centric databases. *BigData Conference*, pages 701–710, 2014.