Uniform Protection for Multi-exposed Targets

Roberto Vigo, Flemming Nielson, and Hanne Riis Nielson

DTU Compute, Technical University of Denmark, Denmark
{rvig,fnie,hrni}@dtu.dk

Abstract. Ensuring that information is protected proportionately to its value is a major challenge in the development of robust distributed systems, where code complexity and technological constraints might allow reaching a key functionality along various paths. We propose a protection analysis over the Quality Calculus that computes the combinations of data required to reach a program point and relates them to a notion of cost. In this way, we can compare the security deployed on different paths that expose the same resource. The analysis is formalised in terms of flow logic, and is implemented as an optimisation problem encoded into Satisfiability Modulo Theories, allowing us to deal with complex cost structures. The usefulness of the approach is demonstrated on the study of password recovery systems.

Keywords: Security verification, distributed systems, protection analysis, security cost, Quality Calculus, Satisfiability Modulo Theories.

1 Introduction

Common sense tells that classified information has to be protected according to its value. This simple principle is at the heart of the concept of multi-level security and the basis for developing access control policies [1]. Nonetheless, despite its simplicity, this design criterion is often ignored in the realisation of complex systems, whether they be software, physical, or cyber-physical systems. On the one hand, code complexity and composition may inadvertently give rise to various paths leading to expose the very same piece of information; on the other hand, usability best practices may prescribe, and technological constraints may oblige, to offer multiple ways to access the same information. The question whether uniform security checks are performed on paths leading to the same resource is then crucial, and its answer far from being obvious.

The Quality Calculus has been introduced in [2] for attacking the problem of Denial-of-Service resorting to default data whenever actual data cannot be obtained due to unreliable communication. More in general, the calculus offers an elegant framework for reasoning about systems where the same functionality can be triggered in multiple ways, and thus enjoy a branching control flow. Taking a process algebraic point of view, suitable for describing software systems but also organisations or physical infrastructure, we develop a static analysis on a value-passing version of the calculus to attack the questions highlighted above.

E. Ábrahám and C. Palamidessi (Eds.), FORTE 2014, LNCS 8461, pp. 182–198, 2014.

Given a system P and a location l of interest, the *protection analysis* computes the information an adversary needs in order to drive P to l, considering all possible paths. This is achieved by translating P into a set of propositional formulae describing the dependency between information and reachability of locations: each satisfying assignment of such set of formulae describes a way in which l can be attained. Moreover, associating costs to literals of such formulae, different assignments can be ranked so as to compute the minimal cost for reaching l. In particular, we consider an attacker model where adversaries are *lucky*: initially having no knowledge, they can decide to *guess* some information on their way to l, incurring the corresponding cost.

Literals represent the knowledge of communication channels by the adversary. This approach proves useful as channels are abstract and flexible entities, which can model a great many cyber and physical scenarios. In particular, we assume that channels provide given degrees of security [3], and quantify their strength: in the cyber world, a secure channel is implemented by means of cryptography, the cost of guessing it being a function of some cryptographic keys (e.g. number of bits for some cryptosystems); in the physical world, a channel may represent a door, its cost quantifying the effort needed to open it (e.g. grams of dynamite). While the correctness of such channels is investigated in the realm of protocol verification, any system with a substantial need for security is likely to have standardised mechanisms to achieve various degrees of protection, and modelling them with secure channels is a coarse yet reasonable abstraction.

The ultimate goal of the framework is to check whether the protection ensured by the implementation, that is, the cheapest way for reaching l, is matching the specification requirements, formalised as a map from locations to desired confidentiality levels.

The novelty of the protection analysis is two-fold: first, the problem of quantifying security checks and comparing desired with actual protection is interesting *per se*; second, the quest for optimal attacks relies on arbitrary cost structures and objective functions, and is implemented via a reduction to Satisfiability Modulo Theories (SMT) that can be exploited in a great many different contexts. Whilst our SMT approach to optimisation is not entirely novel, non-linearly ordered and symbolic cost structures have not been explicitly addressed so far in connection to this technique.

The usefulness of the framework is demonstrated on the study of password recovery systems. We consider the *Microsoft account* system, where a challenge is sent to a previously registered e-mail or phone number, and the *Yahoo!Mail* system, which allows resetting a password upon answering previously selected questions. The results we present are obtained through an implementation of the analysis available at `www.imm.dtu.dk/~rvig/quality-tool.html`.

Outline. Section 2 introduces a motivating example developed throughout the paper. The calculus used to specify the example is presented in Sect. 3. In Sect. 4 the protection analysis is developed. The solution technique based on SMT is discussed in Sect. 5, where we also argue on the importance of complex cost sets. Section 6 concludes and sketches a line for future developments.

Related work. Our work is inspired by a successful strand of literature in protocol verification, where a protocol is translated into a set of first-order Horn clauses and resolution-based theorem proving is used to establish security properties [4,5], and by the flow logic approach to static analysis [6]. In particular, the translation from processes to propositional formulae is inspired by ProVerif [7] translation of protocols into first-order Horn clauses, but can be more formally understood as a flow logic where the carrier logic is not the usual Alternation-free Least Fixed Point Logic, since it cannot express optimisation problems.

In order to formalise the "need for protection" of a location we resort to confidentiality lattices, that are widely used for describing levels of security in access control policies. An excellent introductory reference is [1, chps. 6,7].

As for the solution technique we exploit, SMT is surveyed in [8], while [9,10] present two different approaches to solve optimisation problems via SMT. In particular, Nieuwenhuis and Oliveras [9] proposed to modify the DPLL(T) procedure inherent to SMT solvers so as to look for optimal assignments, while Cimatti et al. [10] developed the search for an optimal assignments on top of an SMT solver, as we do in Sect. 5. Nonetheless, both these works focus on numeric weights, which in our settings are represented with linearly ordered cost structures. Our more general notion of weight is modelled after Meadow's cost sets, formalised as monoids in [11].

Finally, another perspective on the technical developments underpinning the analysis points to computing *prime implicants* of a given formula [12], where in our case primality is sought with respect to the cost set.

2 Motivating Example

We demonstrate the flexibility and usefulness of the framework on the problem of password recovery. As defined in the Common Weakness Enumeration [13], "it is common for an application to have a mechanism that provides a means for a user to gain access to their account in the event they forget their password". It is then crucial to ensure that the protection to the account offered by the recovery mechanism is comparable to the protection provided by the password, otherwise we would have two paths leading to the same resource but performing a different amount of security checks. As noted by the Open Web Application Security Project [14]: "Weak password recovery processes allow stronger password authentication schemes to be bypassed. Non-existent password recovery processes are expensive to support or result in denial-of-service conditions." Below we encode such a system in the Value-Passing Quality Calculus.

$$
\begin{aligned}
\mathsf{System} &\triangleq (\nu\mathsf{access})\,(\nu\mathsf{ok})\,(\nu\mathsf{pwd})\,(!(\mathsf{Login}|\mathsf{Recover})) \\
\mathsf{Login} &\triangleq {}^{1}\&_{\forall}(\mathsf{id}?x_{id}, \mathsf{pwd}?x_p).{}^{2}\mathsf{access!ok} \\
\mathsf{Recover} &\triangleq {}^{3}\&_{\forall}(\mathsf{id}?x'_{id}, \&_{\exists}(\mathsf{mail}?x_m, \mathsf{pin}?x_c)). \\
&\qquad {}^{4}\mathsf{case}\ x_m\ \mathsf{of}\ \mathsf{some}(y_m){:}\ {}^{5}\mathsf{pwd!ok}\ \mathsf{else} \\
&\qquad\quad {}^{6}\mathsf{case}\ x_c\ \mathsf{of}\ \mathsf{some}(y_c){:}\ {}^{7}\mathsf{pwd!ok}\ \mathsf{else}\ 0
\end{aligned}
$$

Process System is modelled after *Microsoft account* login mechanism, in charge of granting access to services such as mailboxes and technical forums. The main process is composed by two parallel sub-processes, running an unbounded number of times: the first one models the normal login procedure, while the second one abstracts the password recovery mechanisms.

According to process Login, in order to be granted access a user has to provide their own id and the corresponding password. This is mimicked by the two inputs expected by the *quality binder* at label 1, which are simultaneously active. The *quality guard* $\forall$ prescribes that both inputs have to be satisfied before proceeding, in any order. These inputs simulate two security checks: a party willing to authenticate into the system has to possess proper credentials, i.e., being able to communicate over id, pwd, and thus know such channels.

In the event the user forgot their password, the recovery mechanism comes into play. Microsoft offers two ways to recover a lost password: (*i*) a reset link is sent to an e-mail address previously registered by the user, or (*ii*) a 7-digit pin is sent to a phone number previously registered by the user. This behaviour is modelled by the quality binder at label 3 in process Recover. The binder is consumed as soon as the user has provided a valid id (e.g., an Outlook.com e-mail address), and proven their identity either through option (*i*) or option (*ii*). In the first case, the user needs to access a mailbox, simulated by an input on channel mail, while in the second case he or she has to provide the correct pin: the alternative is implemented by the existential guard $\exists$ instrumenting the inner quality binder. The case clauses at labels 4, 6 determine what combination of inputs triggered passing the binder. In both cases, the user gets a valid password for the account in question, simulated by the outputs at label 5 and 7, and hence will be able to fulfil the check enforced by process Login.

The key point of the example is that no matter how strong a user's password is, an attacker can always try to guess a 7-digit sequence[1]. In particular, the requirement for a password is having at least 8 characters and containing different cases, numbers, or symbols, which (almost) automatically makes a password stronger than the pin! In terms of confidentiality levels, this suggests that the desired security architecture is not necessarily met, as the protection offered by the three paths leading to authenticating, i.e. to label 2, might not be uniform, depending on the cost notion we resort to. While in this simple case such conclusion can be drawn after careful investigation of the system, there is a general need to develop automated techniques to cope with more complex scenarios. In the remainder of the paper we shall see how the protection analysis confirms our informal reasoning.

[1] As of Oct. 2013 there seemed to be no limit to the number of attempts one could try – we stopped our experiment at about 30. Some of our findings have been informally communicated in a number of situations and now we observe that they enforced both such limit and a daily threshold. While this mutation does make it more difficult to quantify the strength of the mechanism (cf. Sect. 5), it does not affect the relevance of the framework.

3 The Value-Passing Quality Calculus

In the following, we discuss the syntax and the intended semantics of the process calculus we use as specification language. It encodes a value-passing fragment of the Quality Calculus, from which in particular we inherit quality binders and the distinction between data and optional data, that allow modelling robust systems and give rise to branching control flows. As far as this paper is concerned, however, the reader can think of the calculus as a broadcast π-calculus enriched with quality binders. Such construct enhances succinctness, thus easing the task of specifying complex systems, but does not increase the expressiveness, hence the developments carry seamlessly to a variety of process calculi.

The formal semantics of the calculus is briefly commented upon in Appendix A, for it is instrumental to formulate the correctness statements but does not constitute a novelty per se.

Syntax and intended semantics. The Value-Passing Quality Calculus is displayed in Table 1. The calculus consists of four syntactic categories: processes P, input binders b, terms t, and expressions e. A process can be prefixed by a restriction, an input binder, an output $c!t$ of term t on channel c, or be the terminated process $\mathsf{0}$, the parallel composition of two processes, a replicated process, or a case clause.

An output $c!t$ broadcasts t to all processes ready to input over channel c; if there exists no such process, the output is consumed anyway (i.e., broadcasts are non-blocking actions).

An input binder can either be a simple input $c?x$ on channel c, or a *quality binder* $\&_q(b_1, \ldots, b_n)$, which is consumed as soon as enough sub-binders have been satisfied, as dictated by the quality guard q. A quality guard can be any Boolean predicate: we use the abbreviations $\forall$ and $\exists$ for the predicates specifying that all the sub-binders or at least one sub-binder have to be satisfied before proceeding, respectively, as we have seen in the example.

When a quality binder is consumed, some input variables occurring in its sub-binders might have not been bound to any value, if this is allowed by the quality guard. In order to record which inputs have been received and which have not, we always bind an input variable x to an expression e, which is $\mathsf{some}(c)$ if c is the name received by the input binding x, or none if the input is not received but we are continuing anyway. In this sense, we say that expressions represent *optional data*, while terms represent *data*, in the wake of programming languages like Standard ML. In order to insist on this distinction, variables x are used to mark places where expressions are expected, whereas variables y stand for terms.

The case clause in the syntax of processes is then used to inspect whether or not an input variable, bound to an expression, is indeed carrying data. The process $\mathsf{case}\ x\ \mathsf{of}\ \mathsf{some}(y) : P_1\ \mathsf{else}\ P_2$ evolves into P_1 if x is bound to $\mathsf{some}(c)$, and in P_1 y is replaced by c; otherwise, if x is bound to none, P_2 is executed.

Observe that the adversary can always move into the scope of a restricted name by paying the corresponding cost, e.g. guessing pwd in the example. The standard semantics of restriction, according to which a new name c is only known within its scope, is encompassed by assigning c an infinite cost.

Table 1. The syntax of the Value-Passing Quality Calculus

$P ::= 0 \mid (\nu c)\,P \mid P_1 \mid P_2 \mid {}^{l}b.P \mid {}^{l}c!t.P \mid !P \mid {}^{l}\mathsf{case}\ x\ \mathsf{of}\ \mathsf{some}(y) : P_1\ \mathsf{else}\ P_2$		
$b ::= c?x \mid \&_q(b_1, \cdots, b_n)$	$t ::= c \mid y$	$e ::= x \mid \mathsf{some}(t) \mid \mathsf{none}$

As usual, in the following we consider closed processes (no free variable), and we make the simplifying assumption that processes are renamed apart so that names and variables are bound exactly once.

Table 1 defines a proper fragment of the original calculus, as it is not possible to check *what* is received by a given input. This simplification matches the spirit of the analysis, where the knowledge of channels corresponds to the capability of fulfilling the security checks that they represent.

Confidentiality labels. As argued in Sect. 1, different points in a system have to be protected according to the value of the information they process. This idea can be formalised introducing a simple, non-functional extension to the calculus, where a program point of interest is instrumented with a *unique* label $l \in \mathcal{L}$. This is the case of input binders, outputs, and case clauses in Table 1: we limit labels to be placed before these constructs since they represent proper actions. In the following, we will denote labels with the numerals $1, 2, \ldots$, thus taking $\mathcal{L} = \mathbb{N}$. We say that a label l is reached in an actual execution when the sub-process following l is ready to execute. Moreover, we assume to have a *confidentiality lattice* $(\Sigma = \{\sigma_1, \ldots, \sigma_n\}, \sqsubseteq_\Sigma)$, with greatest lower bound operator $\bigsqcap_\Sigma$, and a function $\gamma : \mathcal{L} \to \Sigma$ that maps labels into confidentiality levels. In particular, $\gamma(l_1) \sqsubseteq_\Sigma \gamma(l_2)$ denotes that the confidentiality (i.e., need for protection) of the program point indicated by l_2 is greater than or equal to the confidentiality of the program point indicated by l_1.

As an example of confidentiality lattice, consider the *military lattice* given by $\Sigma = \{\mathsf{unclassified}, \mathsf{confidential}, \mathsf{secret}, \mathsf{top\text{-}secret}\}$, with the ordering $\mathsf{unclassified} \sqsubset_\Sigma \mathsf{confidential} \sqsubset_\Sigma \mathsf{secret} \sqsubset_\Sigma \mathsf{top\text{-}secret}$. More complex lattices, in particular non-linearly-ordered ones, are discussed in [1, chp. 7].

In our running example the intended security architecture prescribes $\gamma(2) = \gamma(5) = \gamma(7)$, as these three program points have the effect of authenticating a user into the system. We can thus rely on a simple security lattice $\mathsf{unrestricted} \sqsubset_\Sigma \mathsf{restricted}$, where the higher element is assigned to labels $2, 5, 7$.

4 Protection Analysis

Given a process P, the protection analysis aims at estimating the amount of knowledge that a process Q has to possess in order for the execution of $P|Q$ to reach a given location l in P. Intuitively, P is running and communicating in a hostile environment, and Q represents an adversary that aims at reaching a protected location, for example authenticating into the system. We focus on

formalisations of P such that l is not reached unless some interactions with the environment take place: the running example of Sect. 2 does not specify the user who is supposed to be logging in, so as to rule out the legal way to authenticating and focus on malicious behaviours.

We consider systems in which the communication takes place over secure channels, where the knowledge of a channel entails the capability of fulfilling the security checks governing the communication. The knowledge that an attacker Q needs to interact with P is thus defined as a set of such channels. Moreover, we can quantify this information by assigning costs to channels: in the case of IT systems, for instance, the degree of security provided by a channel is related to the strength of the cryptographic mechanisms it exploits, and in some cryptosystems be determined as the size of the cryptographic keys needed to communicate over the channel.

Let *Names* be the set of channels occurring in P. We assume to have a function $\mathsf{cost} : \mathit{Names} \to \mathcal{K}$ that associates a cost $k \in \mathcal{K}$ to a channel c. Formally, we require $(\mathcal{K}, \oplus)$ to be a monoid equipped with a partial order $\sqsubseteq_{\mathcal{K}}$, such that $(\mathcal{K}, \sqsubseteq_{\mathcal{K}})$ is a lattice. Intuitively, the greater the protection a channel c ensures, the greater the effort required by Q in order to get access to the channel, and therefore the greater will be the value of $\mathsf{cost}(c)$ with respect to $\sqsubseteq_{\mathcal{K}}$. In the following, we will refer to $\mathcal{K}$ as the *cost set* of the analysis. We require $\oplus$ to be extensive, that is, the sum of two elements always dominates both the summands:

$$\forall k_1, k_2 \in \mathcal{K} \;.\; k_1 \sqsubseteq_{\mathcal{K}} (k_1 \oplus k_2) \;\wedge\; k_2 \sqsubseteq_{\mathcal{K}} (k_1 \oplus k_2) \tag{1}$$

Moreover, we assume that $\oplus$ is monotone and the least element $\bot \in \mathcal{K}$ is its identity element, that is, $\oplus$ is an upper bound operator of the lattice $(\mathcal{K}, \sqsubseteq_{\mathcal{K}})$, and therefore satisfies (1). As highlighted by Meadows [11], the monoid might not be commutative, as the order in which costs are paid might influence their combination. Whilst keeping this elegant generalisation, nonetheless we do not provide in practice any mechanism for re-determining costs dynamically, as the process is evaluated, nor do we discuss a context-sensitive notion of cost.

The outcome of the analysis is a map $\delta : \mathcal{L} \to \mathcal{P}(\mathcal{P}(\mathit{Names}))$ from labels to sets of sets of names (channels), an element $\{c_1, \ldots, c_n\} \in \delta(l)$ denoting a set of channels whose total cost under-approximates the cost that Q incurs in order to enforce reaching l. Finally, $\delta(l)$ is computed such that all its elements have minimal costs with respect to the cost set.

An element of $\delta(l)$ is related to the confidentiality lattice $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ by means of a function $\alpha : \mathcal{P}(\mathit{Names}) \to \Sigma$, compressing cost regions into confidentiality levels:

$$\alpha(\{c_1, \ldots, c_n\}) = \begin{cases} \sigma_1 & \text{if } \bigoplus_{i=1}^{n} \mathsf{cost}(c_i) \in \{k_1^1, \ldots, k_1^{h_1}\} \\ \vdots & \\ \sigma_m & \text{if } \bigoplus_{i=1}^{n} \mathsf{cost}(c_i) \in \{k_m^1, \ldots, k_m^{h_m}\} \end{cases}$$

where α is a well-defined function if the sets of costs $\{k_i^1, \ldots, k_i^{h_i}\}$ are pairwise disjoint and their union is $\mathcal{K}$. Moreover, it is natural to require that α is monotone. A simple example in the cost set $(\mathbb{Z}, +)$ is given by the choice

$$\alpha(\{c_1,\ldots,c_n\}) = \begin{cases} \mathsf{low} & \text{if } \sum_{i=1}^{n} \mathsf{cost}(c_i) \leq 1024 \\ \mathsf{medium} & \text{if } 1024 < \sum_{i=1}^{n} \mathsf{cost}(c_i) \leq 2048 \\ \mathsf{high} & \text{if } 2048 < \sum_{i=1}^{n} \mathsf{cost}(c_i) \end{cases}$$

where numbers could represent the length of cryptographic keys, and we state for instance that a program point is poorly protected if no more than 1024 bits are necessary to attain it (for a fixed cryptosystem).

Intuitively, the function γ is the specification expressing the target security architecture of a system with respect to a given confidentiality lattice, while the function δ captures (an under-approximation of) how this architecture has been realised in the implementation. The overall aim of the analysis, i.e., checking whether the deployed protection lives up to the required confidentiality, can thus be expressed by the property

$$\forall l \in \mathcal{L} \,.\, \gamma(l) \sqsubseteq_{\Sigma} \prod_{\mathcal{C} \in \delta(l)}^{\Sigma} \alpha(\mathcal{C})$$

A violation of this condition is referred to as a potential *inversion of protection*.

4.1 From Processes to Knowledge Constraints

The recursive function $[\![P]\!]\mathsf{tt}$, defined in Table 2, translates a process P into a set of constraints of the form $\varphi \rightsquigarrow \overline{p}$, where φ is a propositional formula and $\overline{p}$ a positive literal. The intended semantics of a constraint implements a backward style of reasoning, according to which if Q knows p, i.e., $\overline{p} = \mathsf{tt}$, then it must be because it knows (enough information to satisfy) φ. As we shall see below, the consequent φ accounts for the checks made on the path leading to disclosing p, namely input binders and case clauses. The antecedent $\overline{p}$ can either stand for a channel c, meaning that Q knows c, an input variable x, meaning that the related input is satisfied (i.e., $x = \mathsf{some}(c)$), or a label l, meaning that l is attained.

At each step of the evaluation, the first parameter of $[\![\cdot]\!]\cdot$ corresponds to the sub-process of P that has still to be translated, while the second parameter is a logic formula, intuitively carrying the hypothesis on the knowledge Q needs to attain the current point in P. The translation function is structurally defined over processes as explained below.

If P is $\mathsf{0}$, then there is no location to be attained and thus no constraint is produced. If $P = !P'$ or $P = (\nu c)\, P'$, then it spontaneously evolves to P', hence Q does not need any knowledge to attain P' and gains no knowledge since no communication is performed. A parallel composition is translated taking the union of the sets into which the components are translated.

Communication actions have instead an impact on the knowledge of Q: inputs represent checks that require knowledge, outputs fulfil those checks, and case clauses determine the flow of control. Whenever such an action $[\![^l a.P']\!]\varphi$ is reached in the translation, a constraint $\varphi \rightsquigarrow \overline{l}$ is generated: if the attacker attains l, then the security checks on a path to l must be fulfilled, and thus φ

Table 2. The translation from processes to propositional constraints

$$\llbracket \mathsf{0} \rrbracket \varphi = \emptyset \qquad \llbracket !P \rrbracket \varphi = \llbracket P \rrbracket \varphi$$
$$\llbracket P_1 | P_2 \rrbracket \varphi = \llbracket P_1 \rrbracket \varphi \cup \llbracket P_2 \rrbracket \varphi \qquad \llbracket (\nu c)\, P \rrbracket \varphi = \llbracket P \rrbracket \varphi$$
$$\llbracket {}^{l} b.P \rrbracket \varphi = \llbracket P \rrbracket (\varphi \wedge \mathsf{hp}(b)) \cup \mathsf{th}(\varphi, b) \cup \{\varphi \rightsquigarrow \overline{l}\}$$
$$\llbracket {}^{l} c!t.P \rrbracket \varphi = \llbracket P \rrbracket \varphi \cup \{\varphi \rightsquigarrow \overline{c}\} \cup \{\varphi \rightsquigarrow \overline{l}\}$$
$$\llbracket {}^{l}\mathsf{case}\ x\ \mathsf{of}\ \mathsf{some}(y) : P_1\ \mathsf{else}\ P_2 \rrbracket \varphi = \llbracket P_1 \rrbracket (\varphi \wedge \overline{x}) \cup \llbracket P_2 \rrbracket (\varphi \wedge \neg \overline{x}) \cup \{\varphi \rightsquigarrow \overline{l}\}$$

$$\mathsf{hp}(c?x) = \overline{c} \qquad \mathsf{hp}(\&_q(c_1?x_1, \ldots, c_n?x_n)) = [\![q]\!](\overline{c_1}, \ldots, \overline{c_n})$$
$$\mathsf{th}(\varphi, c?x) = \{(\varphi \wedge \overline{c}) \rightsquigarrow \overline{x}\} \qquad \mathsf{th}(\varphi, \&_q(b_1, \ldots, b_n)) = \bigcup_{i=1}^{n} \mathsf{th}(\varphi, b_i)$$

$$[\![\forall]\!](h_1, \ldots, h_n) = \bigwedge_{i=1}^{n} h_i \qquad [\![\exists]\!](h_1, \ldots, h_n) = \bigvee_{i=1}^{n} h_i$$

must evaluate to tt. Moreover, the nature of action a determines whether or not other constraints are produced and how the translation proceeds.

Consider a simple input ${}^{l}c?x.P'$: whenever such action is consumed, it must be that the attacker knows the communication channel c, therefore we translate P' under the hypothesis $\varphi \wedge \overline{c}$. Moreover, if the input is consumed, then x must be bound to $\mathsf{some}(c')$, hence we produce a constraint $(\varphi \wedge \overline{c}) \rightsquigarrow \overline{x}$. These two steps respectively accommodate the hypothesis we need for passing a binder, and the thesis we can establish whenever a binder is passed. In Table 2 functions hp and th take care of formalising this intuition, that seamlessly applies to quality binders, where the hypothesis is augmented accounting for the combinations of inputs that satisfy the binder, as dictated by the quality guard q. The last section of Table 2 shows two cases for q, but any Boolean predicate can be used.

The execution of an output ${}^{l}c!t$ satisfies all the security checks represented by inputs waiting on c. Therefore, if Q can trigger such output, it obtains the knowledge related to c without having to know the channel directly, and thus a constraint $\varphi \rightsquigarrow \overline{c}$ is generated. It is worthwhile observing that this behaviour is justified by the broadcast semantics, and by the fact that the calculus is limited to testing whether or not something has been received over a given channel, shifting the semantic load on the notion of secure channel. Moreover, note that the asymmetry between input and output is due to the fact that outputs are non-blocking.

A case construct is translated by taking the union of the constraints into which the two branches are translated: as the check is governed by the content of the case variable x, we record that the then branch is followed only when x is bound to $\mathsf{some}(c)$ by adding a literal $\overline{x}$ to the hypothesis, as we do for inputs, and we add $\neg\overline{x}$ if the else branch is followed.

The set of constraints $\llbracket P \rrbracket \mathsf{tt}$ computed according to Table 2 can be normalised so as to produce a compact representation of system P. Whenever two rules $\varphi \rightsquigarrow \overline{p}$ and $\varphi' \rightsquigarrow \overline{p}$ are in $\llbracket P \rrbracket \mathsf{tt}$, they are replaced with a single rule $(\varphi \vee \varphi') \rightsquigarrow \overline{p}$.

This simplification is intuitively sound since if φ leads to obtain p and φ' leads to obtain p, then p is available to the attacker under the condition that $\varphi \vee \varphi'$ is known. In the following, we assume to deal with sets of constraints in such format.

4.2 Attacker Model: Lucky Attackers and Insiders

A rule $\varphi \leftsquigarrow \overline{p}$ in $[\![P]\!]\mathsf{tt}$ describes how Q can attain p playing according to the rules of the system, namely fulfilling the checks described by φ. Nonetheless, when p is a channel, an attacker can always try to obtain it directly, for instance guessing some cryptographic keys. In order to account for this possibility, we enrich each rule $\varphi \leftsquigarrow \overline{c}$ by replacing the consequent with the disjunction $g_c \vee \varphi$, where literal g_c (for "guess c") represents the possibility of learning c directly, incurring the corresponding cost. Finally, for each channel c such that no rule $\varphi \leftsquigarrow \overline{c}$ is in $[\![P]\!]\mathsf{tt}$, we add to $[\![P]\!]\mathsf{tt}$ a constraint $g_c \leftsquigarrow \overline{c}$, expressing that Q has no option but guessing the channel.

It is worthwhile observing how resorting to propositional logic integrates with the under-approximating nature of the analysis: a channel can either be learnt or not, and its cost contribute or not to the cost of an attack. This means that we do not keep track of the number of attempts made to guess some information, and always assume that guessing c is successful whenever g_c is found to be true. When looking for optimal combinations of channels with respect to the cost structure, we will then count the cost of those channels that Q must guess in order to attain the label of interest, thus obtaining an under-approximation of the cost of running the attack.

Finally, notice how the cost map allows modelling insiders: by decreasing the cost of guessing a given piece of information we can model an attacker who has access to part of the secret, for instance to the rules according to which a random password is generated or to its first letter.

Example. The translation of our example returns a set containing the following constraints, augmented as explained above:

$$\overline{1} \qquad \overline{\mathsf{id}} \leftsquigarrow \overline{x_{id}} \qquad \overline{\mathsf{pwd}} \leftsquigarrow \overline{x_p}$$
$$\overline{\mathsf{id}} \wedge \overline{\mathsf{pwd}} \leftsquigarrow \overline{2} \qquad g_{\mathsf{access}} \vee (\overline{\mathsf{id}} \wedge \overline{\mathsf{pwd}}) \leftsquigarrow \overline{\mathsf{access}} \qquad g_{\mathsf{id}} \leftsquigarrow \overline{\mathsf{id}}$$
$$\overline{3}$$
$$\overline{\mathsf{id}} \leftsquigarrow \overline{x'_{id}} \qquad \varphi \wedge \overline{x_m} \leftsquigarrow \overline{5} \qquad g_{\mathsf{mail}} \leftsquigarrow \overline{\mathsf{mail}}$$
$$\overline{\mathsf{mail}} \leftsquigarrow \overline{x_m} \qquad \varphi \wedge \neg\overline{x_m} \leftsquigarrow \overline{6} \qquad g_{\mathsf{pin}} \leftsquigarrow \overline{\mathsf{pin}}$$
$$\overline{\mathsf{pin}} \leftsquigarrow \overline{x_c} \qquad \varphi \wedge (\neg\overline{x_m}) \wedge \overline{x_c} \leftsquigarrow \overline{7}$$
$$\underbrace{\overline{\mathsf{id}} \wedge (\overline{\mathsf{mail}} \vee \overline{\mathsf{pin}})}_{\varphi} \leftsquigarrow \overline{4} \qquad g_{\mathsf{pwd}} \vee (\varphi \wedge \overline{x_m}) \vee (\varphi \wedge (\neg\overline{x_m}) \wedge \overline{x_c}) \leftsquigarrow \overline{\mathsf{pwd}}$$

where the only way for Q to know $\mathsf{id}, \mathsf{mail}, \mathsf{pin}$ is to guess them. Observe that the capability of using the password channel is obtained either by satisfying the recovery mechanism or by guessing pwd. For the sake of conciseness, we have omitted the conjunct tt in all the constraints.

5 Computing Optimal Assignments via SMT

In order to compute the set of sets of channels $\delta(l)$ that allow reaching l incurring minimal costs, we need to solve an optimisation problem subject to the Boolean constraints $[\![P]\!]\mathsf{tt}$, where $\rightsquigarrow$ can be freely interpreted as the propositional backward implication $\Leftarrow$. There exist various techniques to cope with such problems, each suitable for particular choices of cost sets and objective functions. In the following, we show how to exploit an SMT solver to tackle the problem in its most general form. We limit to mention here that Pseudo-Boolean optimisation or Answer Set Programming with weighted rules are efficient and elegant alternatives for dealing with the monoid $(\mathbb{Z}, +)$ and linear objective functions.

The problem. In a nutshell, our task reduces to compute a satisfying assignment M of the propositional formula

$$\psi = \overline{l} \wedge \bigwedge_{(\varphi \rightsquigarrow \overline{p}) \in [\![P]\!]\mathsf{tt}} (\varphi \Leftarrow \overline{p})$$

namely, we assume that l is reached and we look for the consequences (i.e., implicants of $\overline{l}$). In particular, we are interested in all the assignments M that satisfy ψ, denoted $M \models \psi$, and are minimal with respect to a function of the literals g_i's representing the need for guessing channels:

$$f(g_1, \ldots, g_n) = (g_1 \cdot \mathsf{cost}(c_1)) \oplus \cdots \oplus (g_n \cdot \mathsf{cost}(c_n))$$

where the cost of obtaining a channel contributes to the total cost of the attack only if Q decides to guess the channel, incurring the corresponding cost, $g \cdot \mathsf{cost}(c)$ standing for $\mathsf{if}\ g\ \mathsf{then}\ \mathsf{cost}(c)\ \mathsf{else}\ \bot$ (recall that $\oplus$ is assumed to be monotone and $\bot$ its neutral element). If no solution is found, i.e. ψ is unsatisfiable, then the program point indicated by l is not reachable.

Denoting M a satisfying assignment of ψ, we write $\delta(M)$ for the set of literals that are true in M. Eventually, it might be that there exists a number of assignments with minimal costs, and thus $\delta(l)$ is defined as

$$\delta(l) = \{\delta(M) \mid M \models \psi \ \wedge\ \mathrm{minimal}(M)\}$$

and finally we shall take as result of the analysis the security level σ minimal among those onto which α maps the sets of channels in $\delta(l)$, or their greatest lower bound in case they are incomparable (recall that Σ is a lattice).

An SMT-based algorithm. Such an optimisation problem can be tackled by computing assignments for a list $\pi_1, \ldots, \pi_n$ of SMT problems, where π_i is a more constrained version of π_{i-1} that requires to improve on the cost of the current solution. The initial problem π consists of the propositional constraints ψ and of the objective function f, whose value is stored in variable goal. Then, while the problem is satisfiable, we improve on the cost of the current assignment by asserting new constraints which tighten the value of goal, until unsatisfiability is

Data: The problem $\pi \equiv \psi \wedge (\mathsf{goal} := f(g_1, \ldots, g_n))$
Result: the set $\mathcal{M}$ of pairs (assignment,cost) such that the assignment is minimal w.r.t. f
$\mathcal{M} \leftarrow \emptyset$;
while π *satisfiable* **do**
  $M \leftarrow \mathsf{get\text{-}model}(\pi)$;
  $k \leftarrow M(\mathsf{goal})$;
  forall the $(M', k') \in \mathcal{M} \mid k \sqsubset_{\mathcal{K}} k'$ **do** // M `outperforms` M'
    $\mathcal{M} \leftarrow \mathcal{M} \setminus \{(M', k')\}$
  end
  $\mathcal{M} \leftarrow \mathcal{M} \cup \{(M, k)\}$;
  $\pi \leftarrow \pi \wedge \neg(\bigwedge_{i=1}^{n}(\overline{c_i} = M(\overline{c_i}))) \wedge \neg(\mathsf{goal} \sqsupset_{\mathcal{K}} k)$;
end

Algorithm 1. The SMT-based solving procedure

reported. As for the cost set (comparing and combining costs), SMT solvers offer native support for numeric costs and common mathematical functions, while more complex cost sets have to be encoded manually.

Algorithm 1 displays the pseudo-code of the procedure. In particular, observe that when a new problem π_i is generated, additional constraints are asserted that ask for (i) a different assignment and (ii) a non-greater cost: the former condition speeds up the search, while the latter explores the cost frontier.

The termination of the algorithm is ensured by the finiteness of possible assignments to the propositional variables of the π_i's, whose propositional structure does not change throughout the loop, and by the fact that the same assignment cannot occur twice as solution due to the new constraints we generate in each iteration: at most, we need to solve as many π_i's as models of ψ (ALL-SAT). The correctness of the procedure stems from the fact that when unsatisfiability is claimed, by construction of the π_i's there cannot exist further assignments that comply with the cost constraints. Finally, observe that the Boolean satisfiability problem is NP-complete, but in the last decade solvers have been developed that can handle instances with millions of variables, and are increasingly used in software verification. Our implementation is based on Microsoft Z3.

Example. Consider the login system discussed in Sect. 2, and assume to work in the cost set $(\mathbb{Z}, +)$. The engineering of a sensible cost map is a delicate task that involves cryptographic arguments. Due to space constraint we make here the simplistic assumption that costs to channels are given by the number of bits to be guessed: $\mathsf{cost}(\mathsf{pin}) = 28$ (7 digits, 4 bits each) and $\mathsf{cost}(\mathsf{pwd}) = 56$ (8 symbols, 7 bit per ASCII symbol). We assume that the user id is known (the target e-mail itself, for instance), and that the password of the third-party mailbox is comparable to pwd (as the constraints put by Microsoft are customary also to other e-mail providers). Finally, we disregard channel access as it is only used after the label of interest is reached. The problem is thus to minimise

$$f(g_{\mathsf{id}}, g_{\mathsf{pwd}}, g_{\mathsf{mail}}, g_{\mathsf{pin}}) = g_{\mathsf{id}} \cdot \mathsf{cost}(\mathsf{id}) + g_{\mathsf{pwd}} \cdot \mathsf{cost}(\mathsf{pwd}) + g_{\mathsf{pin}} \cdot \mathsf{cost}(\mathsf{pin}) + g_{\mathsf{mail}} \cdot \mathsf{cost}(\mathsf{mail})$$

under the constraints given by ψ where label 2 is asserted. Instructed with such input, our procedure finds that the formula is satisfiable and the single cheapest assignment is $M = [g_{\mathsf{id}} \mapsto \mathsf{tt}, g_{\mathsf{pwd}} \mapsto \mathsf{ff}, g_{\mathsf{mail}} \mapsto \mathsf{ff}, g_{\mathsf{pin}} \mapsto \mathsf{tt}]$, with cost given by $\mathsf{cost}(\mathsf{id}) + \mathsf{cost}(\mathsf{pin}) = 28$, and entailing $\delta(2) = \{\{\mathsf{id}, \mathsf{pin}\}\}$.

As for the desired security levels, we observed in Sect. 3 that $\gamma(2) = \gamma(5) = \gamma(7) = \mathsf{restricted}$ should hold, for we want the account to be equally protected on all the paths leading to granting access. As the cost for accessing an account in normal condition is 56, it is reasonable to set

$$\alpha(\{c_1, \ldots, c_n\}) = \begin{cases} \mathsf{restricted} & \text{if } \sum_{i=1}^{n} c_i \geq 56 \\ \mathsf{unrestricted} & \text{otherwise} \end{cases}$$

We would like to verify that $\mathsf{restricted} \sqsubseteq_\Sigma \alpha(\{\mathsf{id}, \mathsf{pin}\})$, which is false since $\mathsf{cost}(\mathsf{id}) + \mathsf{cost}(\mathsf{pin}) = 28$. It is thus the case that the implementation potentially guarantees less protection than the amount required by the specification, and therefore we shall issue a warning to the designer of the system.

Obviously, costs are central in determining the outcome of the analysis. For example, we could consider to choose pwd from a password dictionary, for it is unrealistic to assume randomly generated bits: the size of such dictionaries is usually less than the number of sequences of 7 digits, and thus with this cost map the analysis might say that the recovery mechanism offers enough protection. Likewise, a limit to the number of attempts can equalise the protection.

Finally, it is worthwhile noticing that the framework can be exploited to measure the *distance* between the implementation and the specification, and not only there qualitative compliance.

5.1 Motivating Complex Cost Sets

So far we have worked with an example in the cost set $(\mathbb{Z}, +)$, for it is natively encoded into SMT solvers and matches a first intuition of the notion of cost. More realistic numeric considerations can be formalised in the increasingly popular frameworks for the *quantification of information leakage*, e.g. [15], obtaining more robust definitions of function α. Nonetheless, it is not always the case that we are able to describe with numbers the security provided by channels: sometimes the techniques used to ensure security are even incomparable.

Observe that Algorithm 1 is already equipped to cope with the more general problem of optimising on partially ordered cost sets. An interesting case of non-linear cost sets is offered by the study of security in Cyber-Physical Systems, where components combine both software and physical features. In particular, in such systems an attack could require to assemble cyber actions with physical tampering [16], whose costs can either be comparable or not depending on the nature of the quantities we are interested in (for instance, energy and memory are not directly comparable).

In general, three elements characterise the problem and push for the comprehensive SMT-based approach: the non-linearity of the cost set, its symbolic nature, and the non-linearity of the objective function.

A mundane approach to password recovery. Yahoo!Mail password recovery mechanism differs from the one provided by *Microsoft account* in that it is (also) possible to recover a password by answering two personal questions like "What is your mother's maiden name?". It is unclear how to quantify the difficulty of such questions in terms of numbers. Nonetheless, it is simpler for an attacker to get hold of such secrets than guessing a randomly generated pin [17].

A symbolic quantification of the two paths leading to logging into the mailbox can then be modelled in the cost set $(\{\mathsf{cheap}, \mathsf{expensive}\}, \oplus)$, where the cost of answering a question is cheap and the cost of guessing the password is expensive. As for the monoid operator, a suggestion is to use $\mathtt{max}(\cdot, \cdot)$, since asking one, two, or three questions will annoy an attacker but does not really make their task much harder.

The SMT implementation of this example requires to encode the cost structure, that is, declaring its elements and defining the ordering relation as well as the monoid operator. Running the analysis in this cost set, we verified the existence of a cheap path leading to authenticating, namely the path that exploits the question-based recovery mechanism. The framework thus suggests that such option should not be allowed, otherwise we would not provide uniform security on all paths guarding the protected region.

6 Conclusion

In the design of software, physical, and cyber-physical systems, security is often perceived as a qualitative need, but can only be attained quantitatively. Especially when physical components are involved, it is simply impossible to predict and confront any possible attack. Even if it were possible, it would be unrealistic to have an unlimited budget to implement security mechanisms.

The protection analysis we presented has both the merit of automatically inferring the attacks to which a system is subject, among those accountable for in the framework, and to estimate the effort required by a lucky attacker for bypassing the protection mechanisms in place. Hence, the approach enables to identify potential weak paths and compare desired with actual protection. Moreover, the framework allows reasoning with symbolic and non-linearly ordered cost structures, as it is often more natural and informative to describe the relationships between different protection mechanisms instead of assigning them absolute numbers. We showed how the analysis applies to real scenarios giving meaningful insights on the problem of password recovery. Finally, the SMT-based optimisation technique proposed for computing the analysis is exploitable in all the contexts where propositional models have to be ranked.

Future developments include extending the approach beyond Boolean considerations about guessing capabilities, as they intuitively integrate with a probabilistic view of the world. On the practical side we aim at a stand-alone version of the SMT-based solution engine, so as to boost its applicability to other problems.

Acknowledgement. This work is supported by the IDEA4CPS project, granted by the Danish Research Foundations for Basic Research (DNRF86-10).

References

1. Amoroso, E.: Fundamentals of Computer Security Technology. Prentice-Hall (1994)
2. Nielson, H.R., Nielson, F., Vigo, R.: A Calculus for Quality. In: Păsăreanu, C.S., Salaün, G. (eds.) FACS 2012. LNCS, vol. 7684, pp. 188–204. Springer, Heidelberg (2013)
3. Mödersheim, S., Viganò, L.: Secure Pseudonymous Channels. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 337–354. Springer, Heidelberg (2009)
4. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. Journal of Computer Security 6(1-2), 85–128 (1998)
5. Weidenbach, C.: Towards an automatic analysis of security protocols in first-order logic. In: Ganzinger, H. (ed.) CADE 1999. LNCS (LNAI), vol. 1632, pp. 314–328. Springer, Heidelberg (1999)
6. Riis Nielson, H., Nielson, F.: Flow Logic: A Multi-paradigmatic Approach to Static Analysis. In: Mogensen, T.Æ., Schmidt, D.A., Sudborough, I.H. (eds.) The Essence of Computation. LNCS, vol. 2566, pp. 223–244. Springer, Heidelberg (2002)
7. Blanchet, B.: Automatic verification of correspondences for security protocols. Journal of Computer Security 17(4), 363–434 (2009)
8. de Moura, L., Bjørner, N.: Satisfiability modulo theories: introduction and applications. Communications of the ACM 54(9), 69–77 (2011)
9. Nieuwenhuis, R., Oliveras, A.: On SAT Modulo Theories and Optimization Problems. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 156–169. Springer, Heidelberg (2006)
10. Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R., Stenico, C.: Satisfiability Modulo the Theory of Costs: Foundations and Applications. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 99–113. Springer, Heidelberg (2010)
11. Meadows, C.: A cost-based framework for analysis of denial of service in networks. Journal of Computer Security 9(1), 143–164 (2001)
12. Dillig, I., Dillig, T., McMillan, K., Aiken, A.: Minimum Satisfying Assignments for SMT. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 394–409. Springer, Heidelberg (2012)
13. The MITRE Corporation: Common Weakness Enumeration. Weak Password Recovery Mechanism for Forgotten Password (ID:640), `http://cwe.mitre.org`
14. Open Web Application Security Project: Choosing and Using Security Questions Cheat Sheet, `http://www.owasp.org`
15. Alvim, M.S., Chatzikokolakis, K., Palamidessi, C., Smith, G.: Measuring Information Leakage Using Generalized Gain Functions. In: 25th IEEE Computer Security Foundations Symposium (CSF 2012), pp. 265–279. IEEE (2012)
16. Vigo, R.: The Cyber-Physical Attacker. In: Ortmeier, F., Daniel, P. (eds.) SAFECOMP 2012 Workshops. LNCS, vol. 7613, pp. 347–356. Springer, Heidelberg (2012)
17. Griffith, V., Jakobsson, M.: Messin' with Texas Deriving Mother's Maiden Names Using Public Records. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 91–103. Springer, Heidelberg (2005)
18. Nielson, F., Nielson, H.R., Hansen, R.R.: Validating firewalls using flow logics. Theoretical Computer Science 283(2), 381–418 (2002)

A Broadcast Semantics and Correctness Statements

The semantics of Table 4 is based on the transition relation $P \Longrightarrow P'$, which is enabled by combining a local transition $\xrightarrow{\lambda}$ with the structural congruence defined in Table 3.

The semantics models asynchronous broadcast communication, and makes use of a label $\lambda ::= \tau \mid c_1!c_2$ to record whether or not a broadcast is available in the system. Output is thus a non-blocking action, and when it is performed the broadcast is recorded on the arrow. When a process guarded by a binder receives an output, the broadcast remains available to other processes willing to input. This behaviour is encoded in rules (In-ff) and (In-tt), where we distinguish the case in which a binder has not received enough input, and thus keeps waiting, from the case in which a binder is satisfied and thus the computation may proceed. These rules rely on two auxiliary relations, one defining how an output affects a binder, and one describing when a binder is satisfied (enough inputs have been received), displayed in the third and fourth section of Table 4, respectively, and originally introduced in [2].

The semantics is instrumental to phrase the correctness of the analysis:

$$\text{if} \quad P|Q \Longrightarrow^* C[^l P'] \quad \text{then} \quad \exists \mathcal{C} \in \delta(l) \text{ s.t. } \mathcal{C} \subseteq \mathsf{fn}(Q)$$

i.e., for all the executions in which Q can drive P to l, the analysis provides a set of channels that under-approximates the knowledge required by Q.

Technically, it seems convenient to organise a formal proof in two steps. First, if $P|Q$ reaches l then $P|H[\mathsf{fn}(Q)]$ reaches l, where process H is the hardest attacker possible and is parametrised on the knowledge of Q. H can be thought as the (infinite) process executing all possible actions on $\mathsf{fn}(Q)$, and the proof simply argues that whatever Q can, H can (Lemma 1). A similar approach is explained in detail in [18].

Finally, the second step shows that if $P|H[\mathcal{C}']$ reaches l, then there must be a set $\mathcal{C} \in \delta(l)$ such that $\mathcal{C} \subseteq \mathcal{C}'$ (Theorem 1).

Definition 1 (Hardest attacker). *Let $\mathcal{N} = \{c_1, \ldots, c_n\}$ be a finite set of names. $H[\mathcal{N}]$ is the process that does all possible sequence of output actions on $\mathcal{N}$: $H[\mathcal{N}] \triangleq (\nu d)\,(!(c_1!d)) \mid \ldots \mid (!(c_n!d))$.*

Table 3. The structural congruence of the calculus

$P \equiv P$	$P_1 \equiv P_2 \Rightarrow P_2 \equiv P_1$	$P_1 \equiv P_2 \wedge P_2 \equiv P_3 \Rightarrow P_1 \equiv P_3$
$P \mid 0 \equiv P$	$P_1 \mid P_2 \equiv P_2 \mid P_1$	$P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3$
$(\nu c)\,P \equiv P$ if $c \notin \mathsf{fn}(P)$	$(\nu c_1)\,(\nu c_2)\,P \equiv (\nu c_2)\,(\nu c_1)\,P$	$(\nu c)\,(P_1 \mid P_2) \equiv ((\nu c)\,P_1) \mid P_2$ if $c \notin \mathsf{fn}(P_2)$
$!P \equiv P \mid !P$	$P_1 \equiv P_2 \Rightarrow C[P_1] \equiv C[P_2]$	

Table 4. The transition rules of the calculus

$$\frac{P_1 \equiv (\nu \overrightarrow{c})\, P_2 \quad P_2 \xrightarrow{\lambda} P_3}{P_1 \Longrightarrow P_3} \quad \text{(Sys)}$$

$$^{l}c_1!c_2.P \xrightarrow{c_1!c_2} P \quad \text{(Out)}$$

$$\frac{P_1 \xrightarrow{c_1!c_2} P_1' \quad c_1!c_2 \vdash b \to b' \quad b' ::_{\mathsf{ff}} \theta}{P_1 \mid {}^{l}b.P_2 \xrightarrow{c_1!c_2} P_1' \mid {}^{l}b'.P_2} \quad \text{(In-ff)} \qquad \frac{P_1 \xrightarrow{c_1!c_2} P_1' \quad c_1!c_2 \vdash b \to b' \quad b' ::_{\mathsf{tt}} \theta}{P_1 \mid {}^{l}b.P_2 \xrightarrow{c_1!c_2} P_1' \mid P_2\theta} \quad \text{(In-tt)}$$

$$^{l}\mathsf{case\ some}(c)\ \mathsf{of\ some}(y) : P_1\ \mathsf{else}\ P_2 \xrightarrow{\tau} P_1[c/y] \quad \text{(Then)}$$

$$^{l}\mathsf{case\ none\ of\ some}(y) : P_1\ \mathsf{else}\ P_2 \xrightarrow{\tau} P_2 \quad \text{(Else)}$$

$$\frac{P \xrightarrow{\tau} P'}{(\nu c)\, P \xrightarrow{\tau} (\nu c)\, P'} \quad \text{(Res-tau)} \qquad \frac{P \xrightarrow{c_1!c_2} P'}{(\nu c)\, P \xrightarrow{c_1!c_2} (\nu c)\, P'}\ \mathsf{if}\ c \neq c_1 \wedge c \neq c_2 \quad \text{(Res-out)}$$

$$\frac{P_1 \xrightarrow{\tau} P_1'}{P_1|P_2 \xrightarrow{\tau} P_1'|P_2} \quad \text{(Par-tau)} \qquad \frac{P_1 \xrightarrow{c_1!c_2} P_1'}{P_1|P_2 \xrightarrow{c_1!c_2} P_1'|P_2}\ \mathsf{if}\ P_2 = !P_2' \vee P_2 = {}^{l}c_1'!c_2'P_2' \quad \text{(Par-out)}$$

$$c_1!c_2 \vdash c_1?x \to [\mathsf{some}(c_2)/x] \qquad c_1!c_2 \vdash c_3?x \to c_3?x \ \ \mathsf{if}\ c_1 \neq c_3$$

$$\frac{c_1!c_2 \vdash b_1 \to b_1' \quad \cdots \quad c_1!c_2 \vdash b_n \to b_n'}{c_1!c_2 \vdash \&_q(b_1, \ldots, b_n) \to \&_q(b_1', \ldots, b_n')}$$

$$c?x ::_{\mathsf{ff}} [\mathsf{none}/x] \qquad [\mathsf{some}(c)/x] ::_{\mathsf{tt}} [\mathsf{some}(c)/x]$$

$$\frac{b_1 ::_{v_1} \theta_1 \quad \cdots \quad b_n ::_{v_n} \theta_n}{\&_q(b_1, \ldots, b_n) ::_v \theta_n \cdots \theta_1} \quad [\![q]\!](v_1, \ldots, v_n) = v$$

Lemma 1. *Let P, Q be processes. If $P|Q \Longrightarrow^* C[^{l}P']$ then $P|H[\mathsf{fn}(Q)] \Longrightarrow^* C[^{l}P']$.*

Proof. By induction on the length of the derivation sequence.

Lemma 2. *Let P be a process and $\mathcal{N}$ a set of names. If $P|H[\mathcal{N}] \Longrightarrow^* C[^{l}P']$, then, for all processes R with $R \equiv C[^{l}P']$ and for all contexts C', $P|H[\mathcal{N}] \Longrightarrow^* C'[R']$.*

Proof. By induction on the shape of the inference tree for the step $R \equiv C[^{l}P']$.

Theorem 1 (Correctness of the protection analysis). *Let P be a process and $\mathcal{N}$ a set of names. If $P|H[\mathcal{N}] \Longrightarrow^* C[^{l}P']$, then there exists $\mathcal{N}' \in \delta(l)$ such that $\mathcal{N}' \subseteq \mathcal{N}$.*

Proof. By induction on the length of the derivation sequence, exploiting Lemma 2 in the inductive step.