Security Analysis using Flow Logics

Chiara Bodei, Pierpaolo Degano
Dipartimento di Informatica, Università di Pisa
Corso Italia, 40, I-56125 Pisa, Italy — {chiara,degano}@di.unipi.it

Flemming Nielson, Hanne Riis Nielson
Computer Science Department, Aarhus University
Ny Munkegade, DK-8000 Aarhus C, Denmark — {fn,hrn}@daimi.au.dk

Security. Originated in the 70’s, security of computer systems became soon an essential requirement for many applications, especially in the last decade, due to the widespread diffusion of distributed systems and networks. Mobility is really shaping these systems, leading to new scenarios in which security problems become more and more urgent. The software executed on a computer needs not to be produced for it anymore, as it can also be downloaded from a server, somewhere on the net. Consequently, each computational environment offers a general and distributed platform to programs that can be concurrently executed by users either locally or remotely. This makes it mandatory to fix precise policies for access rights to obtain non-interference and the protection of private information. Moreover, it is necessary to face up to the heterogeneity of administration domains and untrustability of connections, due to geographic distribution: communications between nodes have to be guaranteed, both by making it possible to identify partners during the sessions and by preserving the secrecy and integrity of the data exchanged. To this end specifications for message exchange, called security protocols, are defined on the basis of cryptographic algorithms. Even though carefully designed, protocols may have flaws, allowing malicious agents or intruders to violate security. An intruder, gaining some control over the communication network, is able to intercept or forge or invent messages to convince agents to reveal sensitive information or to believe it is one of the legitimate agents in the session. Cryptography can minimize possible malicious effects.
Nevertheless, it is not sufficient: problems move from the management of communication to the management of keys.

Therefore, it appears clear that security objectives should be considered in the very design phase and not approximately recovered after it. This observation calls for a strong attention to the foundational study of security principles and also of mobile and secure programming languages.

Security is not an easy notion, however: establishing that a system is secure can be very hard, and sometimes also impossible, because the dynamic behaviour of systems often leads to undecidability.

For this reason, several techniques have been devised to study and establish properties sufficient to guarantee that a system is secure. There are essentially three orthogonal aspects that these properties want to have under control. The first aspect regards the protection of information devised to be confidential from unauthorized disclosure (secrecy or confidentiality) and thus deals with who may receive or read the data. The second aspect regards the protection of information devised to have integrity against unauthorized modification (integrity) and thus deals with who may send or write the data. Closely related to this second aspect is that of ensuring that the agents do not lie about their identity (authentication) and thus deals with whom one is willing to communicate with. A third aspect concerns the availability of services at all times (avoiding denial of service) and thus deals with overall system performance. Actually, a slogan of the field is that all what matters is CIA: Confidentiality, Integrity and Availability.

Indeed, a lot of different properties and measures have been defined, even if security cannot amount to their simple sum. All the above shows the need for formal methods and flexible tools to catch the multiform nature of security.

**Static Analysis.** In the last years, encouraging results have been obtained by the use of static techniques – not least those based on Type Systems – exploiting notions of classifications and information flow to establish various forms of secrecy and integrity. In this approach, systems or protocols are specified as expressions of some idealized programming language. Security properties are checked through static tests, that, if passed, guarantee that there is no violation, at run-time, of the property under consideration.

We advocate here a specific static technique, based on Flow Logic, for
studying the security of systems. It is built around the more “classical”
approaches to static analysis – namely Data Flow Analysis, Constraint Based
Analysis and Abstract Interpretation – and thus links up with the pioneering
approach taken in the very early studies by Denning [14, 15]. At the same
time it offers a way to exploiting the quite advanced state-of-the-art in static
analysis also in analyses of security.

The general nature of static analysis (see [26] for a presentation comprising
the major approaches mentioned above) is to offer static techniques for
predicting safe and computable approximations to the set of values or be-
(haviour arising dynamically. Program analysis aims at analysing properties
of a program, and more generally of a system, that hold in all executions
— regardless of the actual data upon which the program operates and re-
gardless of the specific environment in which it executes. Therefore static
analysis can serve as a basis for intervening at the source of the problem,
i.e. directly on the code. Consequently, it helps tuning up programming lan-
guages themselves. As far as network computing is concerned this could drive
the search for new programming languages that embody security properties,
without enforcing them at the implementation level. Furthermore, static
analysis provides a repertoire of automatic and decidable methods and tools
for analysing properties of systems. This feature allows one to circumvent
the undecidability issues that generally arise in security [16].

Most of these methods implicitly involve termination, and thus the prop-
erties are intended to “err on the safe side”. In other words, static analysis
provides its users with safe approximate answers. Additionally, for each
analysis it makes sense to impose an ordering on the properties, for example
stipulating that a property is larger than another if more values satisfy the
former than the latter. The properties are then interpreted in such a way that
an analysis remains correct even when it produces a larger property than ide-
ally possible. This corresponds to producing a valid inference in a program
logic for partial correctness. However, program analysis is generally more
efficient than program verification, and for that reason more approximate,
because the focus is on the fully automatic processing of large programs.

**Flow Logics for Control Flow Analysis.** Control Flow Analysis predicts
safe and computable approximations to the set of values that the objects
of a program may assume during its execution. It was mainly developed
for functional languages [34] but can be used for virtually all programming paradigms, e.g. for analysing imperative or object-oriented languages [30] and languages with concurrency [18] and mobility [7]. The approach is related to Data Flow Analysis and can be seen as an auxiliary analysis needed to establish the information about the interprocedural flow of control assumed when specifying the familiar equations of Data Flow Analysis. As pointed out in [26] it can also be viewed as an interprocedural version of the Definition-Use and Use-Definition Chainings that are commonly computed by optimizing compilers.

Our formulation is based on Flow Logics [24]. This is an approach that separates the specification of whether or not a piece of analysis information is acceptable for a given program from actually computing the least (or best) such piece of information. By now this approach has been developed to a level where it links up with state-of-the-art techniques from Data Flow Analysis, Constraint Based Analysis and Abstract Interpretation [23, 25, 28, 29]. This is performed in such a way that specifications obtain many of the characteristics usually attributed to Type Systems. More importantly, this approach naturally leads to a general treatment of semantic correctness (e.g. in the form of subject reduction results) and to the existence of least solutions (in the manner of model intersection properties). This means that all systems can be analysed and that if a system passes a static security check, then all its computations are indeed secure.

We show here how Control Flow Analysis can be used for validating security and safety issues for concurrent and mobile systems. To exemplify these issues we shall briefly survey our recent proposal [7]; see also [6].

**Control Flow Analysis for the \( \pi \)-calculus.** Process algebras offer a pure framework for studying concurrent and distributed systems and, in turn, the security issues connected to them. Systems are specified as expressions of the calculus, called processes. Processes are obtained by combining via a few operators (sequential and parallel composition, nondeterministic choice, declarations) the basic actions of sending and of receiving messages between processes along channels. Furthermore there are some scope operators, such as restrictions and hiding. We choose the \( \pi \)-calculus [21], which is a model of concurrent communicating processes based on name passing. In fact its mechanisms, i.e. its semantic rules, explicitly control the access to channels
and to data; processes can generate and pass new names, to make them available as communication channels. Thus, mobility is expressed in a direct, yet rather abstract form: processes can only communicate on the channels they know: for them, learning the name of a channel amounts to possessing the capability to communicate on the channel.

A Control Flow Analysis for the $\pi$-calculus should focus on the use of channels and of values. Since the identity of bound names can be changed due to alpha-conversion, while a process evolves\(^1\) we need to instrument the syntax in two minor ways corresponding to the ways in which alpha-conversion can be performed. In this way the statement “no communication over $b$ is possible in $(\nu a)(\nu b)(a(x) \mid b(y).\overline{y} \mid \overline{a}(a))$” remains true when considering also the process $(\nu b)(\nu a)(b(z) \mid a(w).\overline{w}(w) \mid \overline{b}(b))$ alpha-congruent to the above one.

The first annotation consists in explicitly decorating names with abstract “channels” whenever they are declared as new; in the $\pi$-calculus terminology, those are the names occurring in restrictions. Formally, we write $(\nu x^\chi)^P$, where $\chi$ is the channel associated with the name $x$, newly declared in the process $P$. The second extension annotates names with explicit “binders” when they are used to denote the value to be received in a communication, i.e., the placeholder $y$ occurring in an input prefix will be bound to a $\beta$, giving rise to the extended input prefix $x(y^\beta)$. Note that decorating a process in this way is merely mechanical and involves no knowledge about its behaviour. Indeed, a typical schema for annotating the occurrences of restricted names and of objects of inputs in a process $P$ is to keep all the $\chi$’s and the $\beta$’s distinct. Also note, that these annotations are not changed when their associated names are changed due to alpha-conversion.

A result, or solution, of our Control Flow Analysis establishes a super-set of the set of names to which a given name may be bound and of the set of names that may be sent along a given channel. The main ingredients of solutions are two mappings, $\rho$ and $\kappa$. The first is the abstract environment, and gives information about which channels names can be bound to. The abstract channel cache $\kappa$ gives information about which channels can be transmitted over channels. Since processes may contain free names we also

---

\(^1\)For example, the process $(\nu a^x)(a(y^\beta).a(z^\beta^\prime).\overline{y}(z) \mid !(\nu x^x)\overline{a}(x))$ performs a first communication, then $\alpha$-converts the name $x$ to perform a second communication and becomes $(\nu a^x)(\nu x^x)(\nu w^x)(\overline{w}(w) \mid !(\nu x^x)\overline{a}(x))$. 

need the *marker environment* that keeps track of the channels and binders associated with the free names.

**Example 1.** To give a rough idea of what a solution looks like, consider an adaptation of the Wide Mouthed Frog key exchange protocol, in its abstract and simplified version analyzed in [3]. The two processes $A$ and $B$ share the channels $c_{AS}$ and $c_{BS}$ with a trusted server $S$. In order to establish the secure channel $c_{AB}$ with $B$, $A$ sends it to the server $S$ on the restricted channel $c_{AS}$, which forwards it to $B$ on the restricted channel $c_{BS}$. Now $A$ can send to $B$ the fresh message $M$ on the secure channel $c_{AB}$. Informally:

- **Message 1** $A \rightarrow S : c_{AB}$ on $c_{AS}$
- **Message 2** $S \rightarrow B : c_{AB}$ on $c_{BS}$
- **Message 3** $A \rightarrow B : M$ on $c_{AB}$

In the $\pi$-calculus:

$$A = (\nu c_{AB}(\nu M^{xM})c_{AS}(c_{AB} \langle M \rangle)$$

$$S = c_{AS}(x^{\beta_x} \cdot c_{BS}(x))$$

$$B = c_{SB}(y^{\beta_y} \cdot y(w^{\beta_w}))$$

$$P = (\nu c_{AS}^{xAS})(\nu c_{BS}^{xBS})(A \mid S \mid B)$$

The (minimal) solution with respect to the initial empty marker environment $me = []$ ($P$ is a closed process), is given by $(\rho, \kappa)$, where, by definition, $\rho(\chi) = \{\chi\}$ for all $\chi \in \{\chi_{AS}, \chi_{BS}, \chi_{AB}\}$:

<table>
<thead>
<tr>
<th>$\beta_x$</th>
<th>$\rho$</th>
<th>$\kappa$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\chi_{AB}$</td>
<td>$\chi_{AS}$</td>
<td>$\chi_{AB}$</td>
</tr>
<tr>
<td>$\chi_{AB}$</td>
<td>$\chi_{BS}$</td>
<td>$\chi_{AB}$</td>
</tr>
<tr>
<td>$\chi_{M}$</td>
<td>$\chi_{AB}$</td>
<td>$\chi_{M}$</td>
</tr>
<tr>
<td>$\chi_{M}$</td>
<td>$\chi_{M}$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

Once a proposed solution $(\rho, \kappa)$ is available (either because it has been guessed or constructed as we will see in a while), its correctness has to be validated. In the Flow Logic approach this requires stating a number of clauses that operate upon judgments of the form:

$$(\rho, \kappa) \vdash_{me} P.$$
We reiterate the need for the marker environment $me$ to deal with the free names in $P$; this means that our analysis is capable of dealing with open systems. An open process $P$ is assumed to be plugged into an operating environment, which will also bind all the free names of $P$. Since processes may be open, our Control Flow Analysis verifies a process regardless of its actual operating environment, apart from obvious constraints (see below, after the introduction of secret and public channels). The task of a marker environment is then to associate free names with markers (i.e. channels or binders), formalising the minimal information that the operating environments should supply.

As examples of clauses, we first consider the ones for the output and the input prefixes \[7\]:

- \[(\rho, \kappa) \models_{me} \overline{x}(y).P \iff (\rho, \kappa) \models me P \forall \chi \in \rho(x) : \rho(me(y)) \subseteq \kappa(\chi).\]

- \[(\rho, \kappa) \models_{me} x(y^\beta).P \iff (\rho, \kappa) \models me[x\mapsto\beta]P \forall \chi \in \rho(x) : \kappa(\chi) \subseteq \rho(\beta).\]

It says that a solution is valid for $\overline{x}(y).P$ if and only if it is valid for $P$ and the set of channels possibly communicated along $x$ (i.e. along each $\chi$ to which $x$ can be bound) includes the channels to which $y$ can evaluate to (i.e. those that can be bound to $\rho(me(y))$). Symmetrically, the rule for input demands that the set of the channels that can pass along $x$ is included in the set of channels to which $y$ can evaluate. Furthermore, the clauses for parallel composition and restriction are the following:

- \[(\rho, \kappa) \models_{me} P_1|P_2 \iff (\rho, \kappa) \models_{me} P_1 \land (\rho, \kappa) \models_{me} P_2.\]

- \[(\rho, \kappa) \models_{me} (\nu x)P \iff (\rho, \kappa) \models_{me[x\mapsto\chi]} P.\]

All the rules dealing with a compound process, such as the first one, require that the solution is validated for each component. In case of restriction, the second clause simply requires to validate the solution for $P$ in the updated marker environment $me[x\mapsto\chi]$.

In general such specifications are to be interpreted coinductively rather than inductively \[23, 24\] and for this reason we have refrained from using inference rules; for sufficiently simple analyses (like the $\pi$-calculus without
higher-order values as in [7]), coinduction and induction coincide and thereby the clauses gain the flavour of Type Systems.

The formulation of this Control Flow Analysis, called 0-CFA, borrows from standard ideas for functional languages. In particular, this means that it is insensitive to flow and context [26], so terms can be re-arranged without affecting the acceptability of a candidate solution; in effect, restrictions can be lifted to the top-level, or to the nearest enclosing operator for recursion, and prefixing of actions can be replaced by their parallel composition.

Example 2. Back to the example above, consider the sub-process \(\pi_{AB}(M)\), skipping the steps that update the marker environment with \(me(M) = \{\chi_M\}\). One has that \(\rho(me(M)) = \{\chi_M\} \subseteq \kappa(\chi_{AB}) = \{\chi_M\}\). Consider instead the sub-process \(y(w^{\beta_w})\), skipping the steps that update the marker environment with \(me(y) = \{\beta_y\}\) and \(me(w) = \{\beta_w\}\). In this case, one has that \(\rho(me(y)) = \{\chi_{AB}\} \text{ and } \kappa(\chi_{AB}) = \{\chi_M\} \subseteq \rho(me(w)) = \{\chi_M\}\).

We have seen that a Control Flow Analysis is formulated as a specification of the correctness of a candidate solution. It is possible to show that least solutions always exist (in the manner of a model intersection property) and to establish their semantic correctness with respect to the operational semantics, in the form of a subject reduction result: roughly,

\[
(\rho, \kappa) \models me P \wedge P \xrightarrow{\mu} P' \Rightarrow (\rho, \kappa) \models me P'.
\]

Remarkably, there is also a constructive procedure for obtaining the least solution. Essentially, establishing \((\rho, \kappa) \models me P\) amounts to checking a number of individual constraints. (See e.g. the check in the above rule for output.) The procedure that generates solutions explicitly extracts these constraints, proceeding by induction on the syntactic structure of processes. For instance, in case of output \(\pi(y).P\), we shall add the set

\[
\{\{\chi\} \subseteq \rho(me(x)) \Rightarrow \rho(me(y)) \subseteq \kappa(\chi)\}
\]

to the existing constraints obtained for \(P\). For the \(\pi\)-calculus, the procedure operates in low polynomial time with respect to the size of the process under analysis.

We would like to stress that subject reduction and least solutions always exist for “well-formed” analyses; also the extraction of a set of explicit constraints can always be performed for inductive specifications operating over
a known universe; however, whether or not the constraints can be solved in polynomial time depends on the actual properties being tracked and the precision of the analysis.

**Security based on Control Flow Analysis.** We see now how to exploit our Control Flow Analysis to address security issues. Once given some security requirements on the *dynamic* behaviour of a process, a solution of the analysis sketched above can serve as a basis for checking the corresponding *static* security properties. Of course, one has to prove, once and for all, that the static property is safe, i.e. that it implies the dynamic property. Besides guaranteeing that a process will behave in a secure way if these static tests on solutions are successful, they give, when they fail, also an insight into the possible reasons for the unsafe behaviour of processes.

To illustrate the above consider, for example, the secrecy property of confinement, discussed in [7] and similar to the notion in [17] (see also [2]). The dynamic property requires that classified information is never disclosed; in other words a process has no leaks. In $\pi$-calculus terms this property can be rephrased as follows. Channels represent the capability to access some information, e.g. a web link can be seen as a channel that permits to read the information reachable through it. Now, as the data are classified in secret and public, so is the set of channels: they are partitioned into $S$, the set of secret channels, and $P$, the set of public ones. A natural constraint on the operating environment is that it has no knowledge about secret channels and cannot guess any of them (if secret channels are implemented through cryptography, then this assumption amounts to supposing that a key cannot be broken by brute-force-attack). Then, the dynamic requirement is that a name $x$, bound to a secret channel, is never made available to the external environment through an action that enlarges the scope of the declaration of $x$ (technically, this is called *extrusion*). The static property, called *confinement*, is expressed on solutions and, needless to say, implies the dynamic property of having no leaks. Confinement requires that a secret channel may only be communicated over secret channels, because everybody can listen to the public ones (of course, every channel can pass along a secret channel). Formally, the static check consists in verifying the following condition on a solution $(\rho, \kappa)$:

$$\forall \chi \in P : \kappa(\chi) = P.$$
Example 3. Consider again our running example, and suppose that the set of channels is partitioned into $S = \{\chi_M, \chi_{AS}, \chi_{BS}\}$ and $P = \{\chi_{AB}\}$. The check on confinement fails, because $\chi_{AB} \in P$ and $\kappa(\chi_{AB}) = \{\chi_M\} \not\subseteq P$. Indeed, the process $A$ may extrude $M$ (bound to the secret channel $\chi_M$) along $c_{AB}$ (bound to the public channel $\chi_{AB}$). Of course, the process would have no leaks if $c_{AB}$ (i.e. $\chi_{AB}$) was secret or if $M$ (i.e. $\chi_M$) was public.

More Properties of the $\pi$-Calculu. We applied our technique to study other security properties, and we plan to consider more. We briefly pass through a few of them.

A variant of the no read-up/no write-down property of Bell and La-Padula [5] (for controlling the access to files in a file system) is adapted in [8] to deal with communication. Processes are given levels of security clearance, and the dynamic property demands that those at high level never send information to those at low level, while communications in the other direction is permitted. A little extension to the machinery discussed so far is sufficient to define a static check (called discreetness) for when a process respects the classification hierarchy, and to prove it safe with respect to the dynamic notion. The solutions are now triples $(\rho, \kappa, \sigma)$ and judgements have the form

$$(\rho, \kappa, \sigma) \models_{me} P.$$

The rôle of $\rho, \kappa, me$ is exactly as in the previous case. The purpose of the new entries is as follows. The clearance label $l$ is the current security level of the process under analysis. The abstract communication structure

$$\sigma = \langle \sigma_{in}, \sigma_{out} \rangle$$

gives a super-set of the channels that can be received by an input (i.e. $\sigma_{in}(l)$) and that can be sent by an output action (i.e. $\sigma_{out}(l)$) within a sub-process with the given clearance level (i.e. $l$). The rules of the Flow Logic consider the new component, in particular, by imposing two further conditions while checking an

- output prefix: $\rho(me(y)) \subseteq \sigma_{out}(l)(\chi)$; and an
- input prefix: $\kappa(\chi) \subseteq \sigma_{in}(l)(\chi)$. 
The channels that can be bound to the object of an output (resp. an input) action along channel $\chi$ must be included in $\sigma_{out}(l)(\chi)$ (resp. $\sigma_{in}(l)(\chi)$), where $l$ is the current security level.

Discreetness is expressed by a requirement on the $\sigma$ component of a solution: a channel cannot be used for sending an object from a process with high level $l$ to a process with low level $l'$. Formally,

$$\forall l', l : l' < l : \forall \chi : \sigma_{out}(l)(\chi) \cap \sigma_{in}(l')(\chi) = \emptyset.$$ 

If a system is discreet, then, at run-time, a high level process cannot write any value to a process at low level, while the converse is allowed; symmetrically a process at low level cannot read data from one of a high level.

A further possible extension of our analysis concerns the static control of the directionality of channels [31], stating that a channel is used only for input or only for output or in the remaining combinations. Besides an obvious use for code optimization, directionality can be applied to check access rights of processes for the usage of channels.

The corresponding analysis is a variant of that used for the read-up/no write-down property, in which the rules are exactly the same, but where the interpretation of the label $l$ changes: each different process is annotated with a distinguished label that acts as a static identifier, rather than as a security clearance level. For instance, one may be sure that a process $\langle P \rangle^l$ uses the channel $\chi$ for input only when $\sigma_{out}(l)(\chi) = \emptyset$.

**Firewalls in the Ambient Calculus.** Another line of work has considered the ambient calculus, and was intended at showing the robustness of our approach to deal also with other forms of calculi. Ambients is a calculus of computation that allows active processes to move between sites; it thereby extends the notion of mobility found in Java where only passive code may move between sites. The untyped calculus was introduced in [10] and a type system for a polyadic variant was presented in [11]. The calculus is modelled on traditional process algebras (such as the $\pi$-calculus) but rather than focusing on communication (of values, channels, or processes) it focuses on the movement of processes between different sites; the sites correspond to administrative domains and are modelled using a notion of ambients.

Since processes may evolve when moving around, it becomes harder to analyse what processes may turn up inside what other processes. In [27] it
is shown how to adapt the techniques described above to develop a Control Flow Analysis and the paper also establishes a subject reduction result and the existence of least solutions. Two accompanying papers, [19, 29], show how to perform a rational reconstruction of the analysis in the framework of Abstract Interpretation as well as how to specify stronger analyses using powerful counting analyses (motivated by state-of-the-art techniques for pointer analysis in imperative programming languages).

The Control Flow Analysis takes the form of judgements of the form

$$(I, H) \models_{me} P$$

where $I$ gives information about what ambients can turn up inside what other ambients and $H$ gives information about the markers associated with classes of ambients; as before $me$ is the marker environment and $l$ designates one of the contexts in which $P$ can appear. The specification of the analysis is somewhat more complex than for the $\pi$-calculus because the notion of “context” is dynamically evolving. Again, least solutions always exist and the semantic correctness of solutions is established in the form of a subject-reduction result with respect to an operational semantics.

An interesting property that can be expressed in the ambient calculus is the behaviour of firewalls. We consider here a firewall that is only supposed to be entered by agents knowing the right passwords; these do not include the name of the firewall which is private to the firewall. In [10] a concrete firewall is studied; it operates by sending out a small probe that can be entered by the agent and the probe then guides the agent inside the firewall. In [10] it is shown that the firewall will never fail to admit agents knowing the proper passwords (and displaying them in the appropriate manner).

The converse scenario is to ensure that the firewall is protective in the sense that it is successful in denying entry to agents not knowing the required passwords. Since checking the absence of explicit occurrences of sensitive passwords in would-be attackers is a simple linear-time scan, this presents a useful approach to screening a system against attackers. However, to establish this property it would seem that one has to consider an infinity of attackers and that therefore automatic techniques would not apply. None the less, in [27] the analysis sketched above is used to show that among the infinitely many attackers it is possible to identify one (called the hardest attacker) that is as hard to protect against as any other attacker (cf. the notion
of hardness for a given class of problems). If an analysis of the firewall operating in parallel with this hardest attacker shows that the attacker cannot enter, then indeed the firewall is protective in all possible execution contexts.

Overall, the development yields a polynomial-time procedure for rejecting a class of non-protective firewalls.

**Security in the Spi-Calculus.** A third line of work, still under our investigation, is the analysis of cryptographic protocols, expressed in the spi-calculus [4, 1]. The data that can be exchanged in communications have a richer structure in the spi-calculus than in the π-calculus that it extends; indeed, messages can be pairs or data encrypted under some key. In particular, there are primitives for

- encryption: \(\{M\}_K\) meaning that a message \(M\), built from numbers and names, can be encrypted under a secret key \(K\); and for

- decryption: \(\text{case } L \text{ of } \{x\}_{K'} \text{ in } P\) meaning that the process attempts to decrypt \(L\) with the key \(K'\). If \(L\) has the form \(\{M\}_{K'}\), then the process behaves as \(P[M/x]\), and otherwise is stuck.

Therefore, the analysis should track the way terms are manipulated. This implies another extension to the syntax, i.e. assigning explicit “labels” to the occurrences of terms, so as to give names to the program points of interest. The desired information is recorded in solutions under a component, called \(\zeta\), that associates abstract values with each datum and its components. Solutions then have the form \((\rho, \kappa, \zeta)\). The other ingredients of the analysis are quite similar to the one developed for the π-calculus. Again, the Flow Logic approach calls for formulating a specification of the correctness of a candidate solution; least solutions always exist and the semantic correctness of solutions is established in the form of a subject-reduction result with respect to an operational semantics.

Also in the spi-calculus case, Control Flow Analysis can be exploited to analyse security properties. It is possible to adapt the notion of *no leaking* and of *confinement* seen for the π-calculus to this framework. The first notion says that a process is *careful* if it does not perform any outputs of secret terms on (names associated with) public channels. The static test amounts to checking that the data that may be sent along public channels can only be made of public components.
Another notion of secrecy has been previously used by Abadi, in [1] to obtain a non-interference property, formalized in terms of testing equivalence [13, 9]. Roughly, a process $P(x)$ (i.e. a process where the variable $x$ is free) does not interfere on $x$ if a second process $Q$ cannot distinguish running in parallel with $P(M)$ from running in parallel with $P(M')$, for every message $M$ and $M'$. There, type soundness guarantees that there is no interference. This notion relies on a different idea of secrecy than our carefulness. However, the two approaches can be compared in a slightly simplified case. If a process $P$ is confined (according to our rules) then it typechecks (according to the Abadi’s typing rules) and therefore $P$ enjoys the non-interference property. Moreover, if a process $P$ typechecks, then it is possible to build a Control Flow Analysis solution that makes $P$ confined. Alternatively, we conjecture that a result similar to Abadi’s can be proved directly for the analysis using techniques based on Partial Equivalence Relations as they have previously been used for “liveness” analyses (talking about the future use of values); see [22] and Section 2.2 of [26].

Finally, our analysis can address message authentication, i.e. the property requiring that the receiver of a message can ascertain its origin, for terms without replication. A first step in this direction is represented by the study of a weak form of message authentication, called authorship, that states when a message is made of packets of information, all generated by a selected process, that is therefore authenticated.

**Conclusion.** In all the examples discussed above, the development of the Control Flow Analysis for a particular language follows the same schema. So do its applications to the verification of the selected properties. We recapitulate this pattern below.

- Choose those values of interest for the language or calculus under study and define the shape of solutions of the analysis. Define a Flow Logic for when solutions are acceptable; this takes the form of defining a number of clauses, much in the style of Type Systems.

- Prove that all solutions are semantically correct, e.g. in the form of a subject reduction result, and prove that least solutions exist, i.e. establish a model intersection property.
• If possible, derive from the clauses a constructive procedure that builds solutions, and evaluate its complexity. (Web-based tools have been constructed for the analyses in [7, 8, 27] and are available from the web-page http://www.daimi.au.dk/~fn/FlowLogic.html under the entries for [7, 8, 27].)

• Select a specific dynamic security property and define a static check on solutions that implies the dynamic condition. This may require to add some components to solutions, but usually these extensions do not affect the results established in the previous steps.

While more complex flow analyses can be devised, these have not been necessary for the applications to security surveyed here. If needed, our approach can be strengthened using state-of-the-art techniques from Data Flow Analysis and Abstract Interpretation because of the strong ties between these approaches and our formulation using Flow Logics.

A more widely used alternative for calculi of computation and security, is based on Type Systems: security requirements are seen as static information about the objects of a system (see, among the long list of references, [1, 35, 36, 37, 38, 20, 32, 33, 12, 11]). Type Systems also allow a clear statement of semantic correctness (sometimes called type soundness) and allow to study whether or not best solutions exist (in the form of principal types). The interplay between Type Systems and Control Flow Analysis is not yet fully understood, but simple Type Systems and simple Control Flow Analyses seem to be equally expressive (see the example on Abadi’s notion of non-interference mentioned above). While both Type Systems and Control Flow Analyses can be proved semantically sound using a subject reduction result, it would seem that only approaches based on Control Flow Analysis admit best analyses for all systems. Indeed, sometimes Type Systems lack the corresponding notion of principal type, thereby making them harder to use in practice.

In our view, the main reason that Control Flow Analysis has not previously been used to any great extent for calculi of computation is that earlier formulations have lacked the elegance of Type Systems; we hope that our approach based on Flow Logics will change that. Another apparent difference is that Type Systems normally are prescriptive, i.e. they infer types and
impose the well-formedness conditions at the same time, whereas static analyses are normally descriptive, i.e. they merely infer the information and then leave it to a separate step to actually impose demands on when programs are well-formed. However, Type Systems lacking the prescriptive elements have been studied under names such as “soft typing” and the prescriptive element can be added to our Flow Logic formulation of static analysis (although the model intersection property is then likely to fail). Anyway, the large commonality between the two approaches should be exploited for a suitable cross-fertilization; not only do they have much the same aims, but also the semantic tools used to prove correctness turn out to be very similar (e.g. bisimulations versus Partial Equivalence Relations).

References


