PH.D. IN COMPUTER SCIENCE, UNIVERSITY OF VENICE
A PH.D. THESIS PROPOSAL

DAMIANO MACEDONIO

In present-day computing environments, a user often employs programs which are sent or fetched from different sites to achieve his/her goals, either privately or in an organization. Such programs may be run as a code to do a simple calculation task or as interactive parallel programs doing IO operations or communications between resources located almost everywhere in the world. To face up such a complex situation we need frameworks for the formalization, analysis and verification of distributed and mobile systems properties.

A process on a network can be influenced by the environment surrounding it, possibly modifying the intended behaviour of the process. Traditional correctness properties and methodologies for sequential systems are no more applicable in presence of distributed and mobile systems. Hence the necessity of designing new formal models for the description of and the reasoning on properties of distributed processes. This necessity has been recently recognized by several authors; milestones papers on this subject are [19, 7]. In particular, the π-calculus [4, 17, 19] is a process calculus where processes interact by sending communication links to each other. The basic computational step is the transfer of a communication link between two processes; the recipient can then use the link for further interaction with other parties. This makes the calculus suitable for modelling systems where the accessible resources vary over time.

Following the traditional approaches for the analysis of concurrent systems, properties of mobile processes can be expressed in terms of *observational equivalences* [20] or modal logics [8, 25]. Traditional verification methods such as type systems, model checking and control flow analysis can been applied also to distributed processes [15, 16, 22]. Although correctness and security issues motivated the studies of abstract models for distributed systems, there are still few works dealing with the verification of correctness or security properties of concurrent and mobiles systems.

We propose to study semantic characterizations of distributed systems which are suitable to analyze processes in dynamic environments. Our purpose is to specify a logical/formal tool which makes it easier to deal with concurrent or mobile systems. A logical formalism should simplify the definition of correctness and security properties for a distributed system. A logical formula defines a property and so it detects a class of processes, the processes that enjoys that property. Moreover the logical framework helps in deriving new properties as well as connections between different characterization of process properties. Our purpose is to individuate a logical language which is able to describe the the behaviour and spatial structure of concurrent systems, and thus it is a useful instrument in deriving correct systems in a compositional way. A candidate language is Spatial Logic [5, 6] which provides a powerful language to formally describe the structure of concurrent processes.

Our principal intention is to play on Spatial Logic, or logic in general, to describe process behaviour w.r.t mobility and in particular security. In fact security is a basic property for distributed systems [23]. To this aim we started by analyzing the definition of secure process in other formalisms, such as SPA [20] and π calculi.

First of all we are studying how security properties are defined in SPA [1, 2, 3, 10, 11, 12], especially noninterference [13, 14]. In particular in SPA we are extending the notion of noninterference to contexts, i.e. processes with a variable subprocess (a hole) that can be replaced by any process, in order to characterize the environments in which such properties can be guaranteed. This will lead us to a notion of "well formed" context. We believe that a "well formed" context is a context which cannot change in unpredictable ways, but follows some predetermined rules. These behavioral constraints should be reflected in the structure of the context itself.

Then we are studying π-calculus in order to make a comparison with other formalization of noninterference among processes [15, 16]. In particular we want to transfer into π-calculus the notion of noninterference properties of SPA, such as secure contexts, and try to express them within a logical framework.

To conclude we propose a possibly title for our project: "A logical framework to deal with concurrency and security properties".

This presentation is organized as follows. In Section 1 we present π-calculus as a model for Spatial Logic. In Section 2 we present some noninterference results on SPA and π-calculus. Finally, in Section 3 we present a calculus, called Basic Logic [18, 24], developed by the logic group of Padua with which I'm cooperating. Basic Logic is a weak logic which represent the core of all the logics. In fact it can be extended in order to obtain a lot of well known logics. In particular Spatial Logic should be one of its extension.

In Appendix we report all the formalisms we refer to during the presentation.

1. π-Calculus and Spatial Logic

During the last years a lot of computational formalisms have been proposed to describe the behaviour of concurrent and mobile systems. Almost of them have a common core which is asynchronous π-calculus [4, 17] (henceforth simply π-calculus). The π-calculus is then a starting point for studying mobile systems and experimenting with primitives for communication among processes.

π-calculus is a *nominal calculus*, namely a computational formalism that includes a set of pure names and allows the dynamic generation of fresh, unguessable names. The computations consist of collections of parallel processes equipped with asynchronous communication channels. To see the complete definition of processes we refer to Definition 1 in Appendix. If P and Q are processes, process $P|Q$ consists of P and Q running in parallel, possibly communicating with each other or with the outside world along communication channels. Channels are represented by names. Names are the only data that may be communicated on channels, hence a process may communicate a pure value or even a channel.

Although based on a handful of primitives, the π-calculus is very expressive. Functional programming may be reduced to the π-calculus, since the λ-calculus can be encoded. Similarly, a variety of formulations of imperative, object oriented and concurrent programming may be reduced to the π-calculus. All these encodings depend on the presence of pure names and generation.

A restriction in the π-calculus is a process of the form $(\nu n)P$, which means "make a fresh unguessable name n, and then run P". Restriction represent name generation in the π-calculus. The name n in the restriction is a *bound identifier*, whose scope is P. Restricted names are unguessable, namely one can learn a restricted name only if someone tells it.

The void process, $\mathbf{0}$, is a process that can do nothing. The input prefix $m(n).P$ can receive any name from the channel m, by a message $m\langle n'\rangle$, and continue the computation as P with the received name substituted for n. Finally the replication $!P$ can be thought s an infinite composition $P|P|P|\ldots$ Replication makes it possible to express infinite behaviours.

A basic point of π-calculus is its collection of equational laws for restricted names, see Definition 2. For instance, one law asserts that the processes $(\nu n)P|Q$ and $P|(\nu n)Q$ are the same, provided that the name n does not occur in process P. This law allows the scope of a restricted name to vary. Left to right, it allows scope contraction; right to left, it allows scope expansion. A series of applications of this law, together with other laws, allows the scope of a name to expand and contract as it is passed from one process to another during the computation. In this sense, the scope of a restricted name is mobile.

Caires and Cardelli's Spatial Logic [5, 6] has been proposed with the same aim of π-calculus: describing the behaviour and spatial structure of concurrent systems. Logics for concurrent systems are not new, but the intent to describe spatial properties seems to have arisen only recently. The spatial properties that this logic consider are essentially of two kinds: whether a system is composed of two or more subsystems (i.e. "Composition" of π-calculus), and whether a system restricts the use of certain resources to certain subsystems (i.e. "Restriction" of π-calculus).

A model of Spatial Logic is just π-calculus, but the idea can be easily extended to other calculi, such as ambient calculi with locations.

A formula in Spatial Logic describes a property of a particular concurrent system at a particular time; therefore it is modal both in space and in time.

The constructs of the logic are entirely reported in Definition 4. Now we give a brief overview of the interpretation of formulae. Let $\mathbb{P}$ the set of processes of π-calculus, as described in Definition 1. A *property* on $\mathbb{P}$ is a set of processes: a subset of $\mathbb{P}$. A spatial closed formula denotes a property, namely it denotes the collection of processes satisfying that formula.

The collection of all properties (which is not quite the powerset of $\mathbb{P}$, as reported in Appendix) has the structure of a Boolean Algebra under set inclusion, so we naturally get boolean connectives in the logic, such as F, $A \wedge B$ and $A \to B$.

The propositional fragment is extended to predicate logic via a universal quantifier $\forall x.A$. This quantifier has standard properties, but the bound variable x ranges always over the countable set of channel names of π-calculus.

The collection of all properties has also the structure of a quantale, induced by the parallel composition operator over processes. In the logic, this is reflected by the operators $A|B$, $\mathbf{0}$ and $A \triangleright B$. The formula $A|B$ is the parallel composition of two properties and represents the processes that have the form $P|Q$ with P satisfying A and Q satisfying B. The formula $\mathbf{0}$ denotes the collection of void processes. Finally the formula $A \triangleright B$ is the linear implication associated with $|$. It corresponds to context system specification, i.e. a process satisfies $A \triangleright B$ if it gives a process satisfying B in case it is computed in parallel with a process satisfying A.

π-calculus's process restriction induces a pair of operators $n®A$ and $A \oslash n$ called revelation and hiding, that give a basis for describing restricted processes at the logical level. In particular $n®A$ identifies all the processes that are of the form $(\nu n)Q$ with Q satisfying A. The operator ® is called revelation because it reveals the properties satisfied by the processes before the hiding of name n. On the other hand $A \oslash n$ identifies all the processes P such that $(\nu n)P$ satisfies A. The operator $\oslash$ is called hiding because it hides the properties satisfied by the process P before the hiding of name n.

The notion of *fresh name* is introduced by a quantifier $\not|x.A$. This means that x denotes a name that is fresh with respect to the names used either in A or in processes satisfying A. A process satisfies $\not|x.A$ if for some fresh names n (fresh in the process and in the formula) it satisfies $A[n/x]$. This quantifier exhibits the universal/existential ambivalence typical of freshness: a property holding of some fresh names should also hold of any other fresh name.

The logical operator $n\langle m\rangle$ asserts that a message m is present over channel n, so it gives some minimal power to observe the behaviour of processes.

The "next step" temporal operator $\Diamond A$ allows us to talk about the behaviour of a process after a single reduction step.

In Definition 6 we report some derived connectives of basic interest. Standard operation of the classical predicate calculus, namely $\neg A$ (Negation), $\exists x.A$ (Existential quantification), $A \vee B$ (Disjunction) and T (True), are defined as expected. An interesting connectives is $A||B$, De Morgan dual of composition $A|B$, which supports the definition of a form of a spatial quantification. A process P satisfies $A||B$ if and only if every component of P, with respect the parallel composition, satisfies either A or B. We also have the modality $\Box A$, which is the dual of $\Diamond A$. A process P satisfies $\Box A$ if and only if all processes to which it reduces in one step satisfy A. Finally the free name predicate $©\eta$ holds of all processes with a free occurrence of name η.

Moreover when combined with revelation, the fresh name quantifier gives rise to a natural operation of quantification over hidden (restricted) names in a process. Intuitively, a name is revealed under a fresh identity, and then a property of the restricted processes is asserted. One can define $\nvdash x.A \stackrel{\text{def}}{=} \not|x.x®A$. According to the semantics given in Definition 5, we get the following direct semantic characterization

$$[\![\nvdash x.A]\!] = \{P : P \equiv (\nu n)Q \text{ with } Q \in [\![A[n/x]]\!] \text{ and } n \text{ not free both in } P \text{ and } A\}$$

A formula $\nvdash x.A$ reads "there is a restricted name x such that A holds for the process under the restriction". The hidden name quantifier makes it possible to express properties of processes

that depend on some secret name. For a quite simple example, consider the closed formula $\exists y.\vdash\!\!| x.(y\langle x\rangle|T)$. This formula holds precisely of those processes which are ready to send a secret name over a public channel.

In Spatial Logic we have a primitive formula to observe messages, $n\langle m\rangle$, corresponding to the asynchronous π-calculus. We do not have a corresponding input formula, but it can be express by using the guarantee and next step operators. We can take the following definition of input:

$$n(x).A \stackrel{\text{def}}{=} \forall x.n\langle x\rangle \triangleright \Diamond A$$

The definition says literally, that an input process is one that, in presence of any output message y over the given channel n, at the next step (after input) it behaves according to A. It is easy to se that input and output interact as expected in π calculus communication, that is $n\langle m\rangle|n(x).A$ entails $\Diamond A[m/x]$.

The rules of spatial logic are sound w.r.t. the model built with π-calculus' processes and a cut-elimination theorem is proved for the first-order fragment.

Some properties of the logic are very sensitive of the formulation of structural congruence (in fact, Sangiorgi has shown that the process equivalence induced by a similar logic is essentially structural congruence [25]). An open problem is to determine what is the processes equivalence induced by Spatial Logic.

The general structure of Spatial Logic's definition can be easily adapted to various process calculi, and it is also largely independent from the details of the operational semantics. Hence Spatial Logic may give a model in which we can abstract various process calculi in order to compare their expressiveness.

Although the semantics considered is based on unlabelled transition systems, it could be extended in a natural way to labelled transition systems by introducing new modalities into the logic.

In conclusion, Spatial Logic is a very intensional logic , that talks about fine details of process structure. This is what is required if we want meaningfully describe the distribution of processes and the use of resources over a network.

2. Security as noninterference

Protecting the confidentiality of information manipulated by concurrent systems is an important problem. There is little assurance that current systems protect data confidentiality and integrity. Analysing the confidentiality properties of a distributed system is difficult even when insecurity arises only from unintentional errors in the design or implementation. Additionally modern systems commonly incorporate untrusted, possibly malicious host or code, making assurance of confidentiality more difficult.

The standard way to protect confidential data is *access control*: some privileges, which we will call *high actions*, are required in order to access files or objects containing the confidential data. Access control checks place restrictions on the *release* of information but not its *propagation*. Once information is released from its container, the accessing program may, through error or malice, improperly transmit the information in some form. To ensure that information is used only in accordance with the confidentiality policies, it is necessary to analyse how information flows within the using program. The analysis must show that information controlled by a confidentiality policy cannot flow to a location where that policy is violated.

If a user want to keep some data confidential, he might choose a policy stating that no data visible to other users is affected by confidential data. This policy allows programs to manipulate and modify private data, so long as visible outputs of those programs do no improperly reveal information about the data. This is called *noninterference policy* [13], because it state that confidential data may not interfere with public data.

An attacker (or unauthorized user) is assumed to be allowed to view information that is not confidential (i.e. public). The usual method for showing that noninterference holds is to demonstrate that the attacker cannot observe any difference between two executions that differ only in their confidential input.

Noninterference policy has been implemented in various formalisms such as *Security Process Algebra* (SPA for short), or π-calculus, or *ambient calculus*. We first studied noninterference in SPA, then we moved to π-calculus.

SPA is a process algebra that express the interactions (*actions*) among processes. In SPA there is not value passing. The set of visible actions is partitioned into high level actions (H) and low level ones (L), in order to specify multilevel systems. A special action τ models internal computations, namely actions that are not visible outside the system. Actions may be inputs ,α, or output, $\bar{\alpha}$. Input and output on the same channel may synchronize and produce a τ action.

The operational semantics of SPA is given in terms of *Labelled Transition Systems*, see Subsection 3.3 in appendix. A *Labelled Transition System* (LTS) is a triple $(S, A, \rightarrow)$ where S is a set of states, A is a set of labels (actions), $\rightarrow \subseteq S \times A \times S$ is a set of labelled transitions. The notation $S_1 \xrightarrow{a} S_2$ means that the system can move from the state S_1 to the state S_2 by performing the action a.

The concept of *observational equivalence* between two processes is based on the idea that two systems have the same semantics if and only if they cannot be distinguished by an external observer. This is obtained by defining an equivalence relation over the set of processes, equating two processes when they are indistinguishable. Two particular observation equivalences are *strong bisimulation* and *weak bisimulation* [20]. Intuitively strong bisimulation equates two processes if they are able to mutually simulate their behaviour step by step. A weak bisimulation does not care about internal τ actions, that corresponds to synchronization inside the system. So, when a process simulates an action, it can also execute some τ actions before or after that action.

Noninterference among processes has been formalized in [10, 11]with the definition of *Bisimulation-based Non Deducibility on Compositions* (BNDC). The BNDC security property aims at guaranteeing that no information flow from the high to the low level is possible, even in the presence of malicious processes. The main motivation is to protect a system also from internal attacks, which could be performed by the so called *Trojan Horse* programs. Property BNDC is based on the idea of checking the system against all high level potential interactions, representing every possible high level malicious program. In particular, a system E is BNDC if for every high level process Π a low level user cannot distinguish E from $E|\Pi$.

BNDC property is not strong enough to analyse systems in dynamic execution environments. For instance, if code mobility is allowed, a program could migrate to different host in the middle of its computation. In this setting we have to guarantee that every reachable state of the process is secure. Another interesting example is the execution of an applet on a Java Card, where an attacker could try to bring the card in an unstable (insecure) state by powering off the card in the middle of applet computation. To deal with these situations it has been introduced the security property named P_BNDC that requires that every state reachable by the process has to be secure.

However, this property still requires a universal quantification over all the possible reachable states from the initial process. This can be avoided by including the idea of "being secure in every state" inside the bisimulation equivalence notion. This is done by defining an equivalence notion which just focus on observable actions not belonging to H. To do this, it is used a transition relation ($\approx_{\setminus H}$) which does not take care of both internal and high level actions. By using this equivalence, one can show that a process E is P_BNDC if and only if $E \approx_{\setminus H} E \setminus H$.

Noninterference is given treatment also in π-calculus [15]. Here the problem of quantification has been solved by using types. Numerous typing systems have been developed for this language. Most are based on judgments of the form $\Gamma \vdash P$, indicating that the process P is well-typed with respect to the security policy Γ, which associates capabilities with the free channel names of P. Usually these capabilities are of the form:

$\mathtt{r}\langle T\rangle$: the ability to read values of type T form a channel (*read capability*);
$\mathtt{w}\langle T\rangle$: the ability to write values of type T to a channel (*write capability*).

In addition, a complete lattice SL of security levels is associated to obtain *security types*. By varying the precise definition of a security type we can either implement resource access control methodologies, or ensure forms of noninterference.

Also in this case a notion of process behaviour, deriving from the definition of an observational equivalence, is required to introduce a noninterference condition. In particular the *may equivalence* is defined by introducing an *observer* process. An observer T is a process with an occurrence of a new reserved resource name ω, used to report success. When placed in parallel with a process P, an observer may interact with P, producing an output on ω whenever some desired behaviour of P has been observed, in this case we say that $P\,\mathbf{may}\,T$.

We say that two processes are observational equivalent, in according to may equivalence ($\simeq_{may}$), and we write $\Gamma \rhd_\sigma P \simeq_{may} Q$, if for every finite observer T, such that it is typed by Γ with security level σ, we have $P\,\mathbf{may}\,T$ if and only if $Q\,\mathbf{may}\,T$. The definition and the typing rules are enough to ensure that

$$\Gamma \rhd_\sigma P \simeq_{may} Q \text{ implies } \Gamma \rhd_\sigma P|H \simeq_{may} Q|K$$

for all H, K processes that Γ types with security level σ.

This is quite a general non interference result. For instance in the case where Q is P and K is the void process $\mathbf{0}$ we obtain

$$\Gamma \rhd_\sigma P \simeq_{may} P|H$$

indicating that, under previous conditions, the process H can not interfere with the behaviour of P.

Our aim is to translate the noninterference notion of SPA into π-calculus in order to study the relations with the noninterference definitions of π-calculus.

3. Basic Logic

Up to the beginning of last century, there was only one logic, Aristotele's Classical Logic, which was conceived as a metaphysical absolute. Starting with Brouwer's revolution, which introduced Intuitionistic Logic, number of different new logics have been developed. Each of them aimed to capture some of the distinctions which can be observed in a specific field of interpretation, but which are ignored by Classical Logic. Excluding intensional logics (which considers modalities), all such logics can be grouped under three main headings: intuitionistic logic (absence of principle of double negation), quantum logic (absence of distributivity between conjunction and disjunction), and relevance and linear logic (finer control of structural rules).

Although all of these logics are derived from Classical Logic, they have been considered as mutually incompatible. Basic Logic [24] was developed in order to provide a common foundation and to show that they share a common structure.

Basic Logic is introduced following the idea that connectives of logical language *reflect* a link between assertions at metalanguage. The common explanation of the truth of a compound proposition like $A\&B$ is that $A\&B$ is true if and only if A is true *and* B is true. In other terms, a connective $\circ$ between propositions, like & above, reflects at the level of object language a *link* between assertions in the metalanguage, like *and* above. The semantical equivalence

$$A\&B \textit{ true} \text{ if and oly if } A \textit{ true link } B \textit{ true} \tag{1}$$

which we call *definitional equation* for &, gives all we need to know about it. $A\&B$ is semantically *defined* as that proposition which, when asserted true, behaves exactly as the compound assertion A *true link* B *true*. The inference rules for & are easily obtained by *solving* the definitional equation, and they provide an explicit definition. We the say that & is introduced according to the *principle of reflection*. All the connectives of Basic Logic are introduced according this principle.

We need only two metalinguistic *links* to solve equations and introduce all the rules of the calculus: *and* and *entails*. Usually a sequent $\Gamma \vdash \Delta$ denotes $C_1, \ldots, C_m \vdash D_1, \ldots, D_n$, which in turn denotes (C_1 *True and* ... *and* C_m *True*) *entails* (D_1 *True and* ... *and* D_n *True*). A rule such as

$$\frac{\Gamma \vdash \Delta}{\Gamma' \vdash \Delta'}$$

denotes ($\Gamma \vdash \Delta$) entails ($\Gamma' \vdash \Delta'$). This vision can be extended to rules with two preconditions, in such a case the empty space denotes *and*.

All the assumptions we need are only:

$$\text{identity} \qquad A \vdash A$$

$$\text{composition} \quad \frac{\Gamma \vdash A \quad A, \Gamma' \vdash \Delta}{\Gamma, \Gamma' \vdash \Delta}\ \text{cutL} \qquad \frac{\Gamma \vdash \Delta, A \quad A \vdash \Delta'}{\Gamma \vdash \Delta, \Delta'}\ \text{cutR}$$

Now we can see how to obtain the inference rules for connective & starting from equation (1) and using only "identity" and "composition". The direction from right to left of (1) gives a good rule:

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A\&B} \tag{2}$$

The direction from right to left gives

$$\frac{\Gamma \vdash A\&B}{\Gamma \vdash A} \qquad \text{and} \qquad \frac{\Gamma \vdash A\&B}{\Gamma \vdash B} \tag{3}$$

But these are not good rules because the connective we have to define compares into the premisses. We can solve the equation by founding an equivalent admissible rule. If we take $\Gamma = A\&B$ we obtain a trivial premiss ($A\&B \vdash A\&B$) that gives us two axioms $A\&B \vdash A$ and $A\&B \vdash B$. Finally, by using composition with the axiom $A\&B \vdash A$ and the premiss $A \vdash \Delta$ we obtain a good rule:

$$\frac{A \vdash \Delta}{A\&B \vdash \Delta} \tag{4}$$

Now we can observe that we can proceed in the opposite way to obtain (3) from (4). This proves that all the rules are equivalent. We have obtained the rules (2) and (4) that are equivalent to the definitional equation (1).

The main discovering is that all the connectives of well-known logics could be introduced according to the principle of reflection, moreover the solution of the definitional equation follows always the same schema.

Basic Logic provides a simple structure, in which all the usual logics are founded. We can obtain all the extensions of our logic **B** by operating all the combinations of actions **L** (introduction of contexts on the left), **R** (introduction of the contexts on the right), and **S** (adding structural rules of weakening and contraction). In particular Girard's Linear Logic is **BLR**, Intuitionistic Logic is **BLS**, and Classical Logic is **BLRS**.

Finally Basic Logic admits a mathematical model, just a monoid equipped with a binary relation, that we call *relational monoid* [18]. This model can be extended by adding properties that involves the monoidal operation and the binary relation, in such a way we can obtain well known models for the extensions of Basic Logic, for instance Girard's phase space for Linear Logic. In particular, expressing Spatial Logic as an extension of Basic Logic could lead to a mathematical model for Spatial Logic.

References

[1] A. Bossi, R.Focardi, C. Piazza, S. Rossi, *A Proof System for Information Flow Security*. LOPSTR'02. To appear.

[2] A. Bossi, R. Focardi, C. Piazza, S. Rossi, *Bisimulation and Unwinding for Verifying Possibilistic Security Properties*. Proc. Fourth International Conference on Verification, Model Checking and Abstract Interpretation, VMCAI 2003, to appear.

[3] A. Bossi, R.Focardi, C. Piazza, S. Rossi, *Transforming processes to check and ensure Information Flow Security*. AMAST'02. To appear.

[4] G. Boudol, Asynchrony and the π-calculus. Technical Report 1702, INRIA-Sophia Atipolis, 1992.

[5] L. Caires, L. Cardelli, *A Spatial Logic for Concurrency (Part I)*. Naoki Kobayashi and Benjamin C. Pierce (Eds.). Theoretical Aspects of Computer Software; 4th International Symposium, TACS 2001, Sendai, Japan, October 2001, Proceedings. Lecture Notes in Computer Science 2215. Springer, 2001. ISBN 3 540 42736 8. pp 1-37.

[6] L. Caires, L. Cardelli, *A Spatial Logic for Concurrency (Part II)*. To appear in CONCUR'02.

[7] L. Cardelli, A.D. Gordon, *Mobile Ambients*. In Proceedings of Foundations of Software Science and Computation Structures (FoSSaCS), 1998, volume 1378, LNCS, M. Nivat (Ed.), Springer-Verlag.

[8] L. Cardelli, A.D. Gordon, *Anytime, Anywhere. Modal Logics for Mobile Ambients*. Proceedings of the 27th ACM Symposium on Principles of Programming Languages, 2000. pp 365-377.

[9] L. Cardelli, A.D. Gordon, *Mobile Ambients*. Theoretical Computer Science, Special Issue on Coordination, D. Le Mtayer Editor. Vol 240/1, June 2000. pp 177-213.
[10] R. Focardi, *Analysis and Automatic Detection of Information Flows in Systems and Networks*. PhD Thesis, UBLCS-99-16, University of Bologna, July 1999.
[11] R.Focardi, R. Gorrieri, *Classification of security propeties (Part I: Information Flow)*. In R. Focardi and R. Gorrieri, editors, Foundations of Security Analysis And Design, volume 2171 of LNCS. Springer, 2001.
[12] R. Focardi, C. Piazza, S. Rossi, *Proofs Methods for Bisimulation based Information Flow Security*. Proc. Third International Workshop on Verification, Model Checking and Abstract Interpretation, VMCAI 2002, LNCS 2294, pag. 16-31, Springer-Verlag, 2002.
[13] J.A. Goguen, J. Meseguer, *Security Policiy and Security Models*. Proc. of the 1982 Symposium on Security and Privacy, pages 11-20. IEEE Computer Society Press, 1982.
[14] J.A. Goguen, J. Meseguer, *Inference Control and Unwinding*. Proceedings of the 1984 Symposium on Security and Privacy, pp 75-86. IEE Computer Society 1984.
[15] M. Hennessy, *The security picalculus and non-interference*. Computer Science Report n. 05/2000. COGS, University of Sussex.
[16] M. Hennessy, J. Riely, Information Flow vs. Resource Access in the Asynchronous Pi-Calculus. In Proceedings of the International Colloquium on Automata, Languages and Programming, vol. 1853 of Lecture Notes in Computer Science, pp. 415-427, Springer-Verlag, 2000.
[17] K. Honda, M.Tokoro, *On asynchronous communication semantics*. In P. Wegner, M Tokoro, O. Nierstrasz, editor, Proceedings of the ECOOP '91 Workshop on Object-Based Concurrent Computing, volume 612 of LNCS 612 Springer-Verlag, 1992.
[18] D. Macedonio, G.Sambin, *Relational Semantics for Basic Logic*. To appear in The Journal of Symbolic Logic.
[19] R. Milner, J. Parrow, D. Walker, *A Calculus of Mobile Processes, parts I and II*. Information and Computation, 100, pp 1-77, 1992.
[20] R. Milner, *Communication and Concurrency*. Prentice-Hall, 1989.
[21] R. Milner, *Communicating and Mobile Systems: the π-calculus*. Cambridge University Press 1999.
[22] F. Pottier, *A Simple View of Type-Secure Information Flow in the π-Calculus*. In Proceedings of the 15th IEEE Computer Security Foundations Workshop, pages 320-330, 2002.
[23] A. Sabelfeld and A. C. Myers, *Language-Based Information-Flow Security*. IEEE J. Selected Areas in Communication", 2002. To appear.
[24] G. Sambin, G. Battilotti, C. Faggian, *Basic logic: reflection, symmetry, visibility*. J. Symbolic Logic 65 (2000), pp. 979-1013.
[25] D. Sangiorgi, *Extensionality and Intensionality of the Ambient Logics*. In 28th Annual Symposium on Principles of Prograsmming Languages, pp.4-13 ACM, 2001.

Appendix

3.1. **Syntax and operational semantics of the asynchronous π-Calculus.** We report the syntax and operational semantics of the asynchronous π-calculus [4, 17] using the notation of [21].

Definition 1 (**Processes**). *Given a countable set Λ of names, The set $\mathbb{P}$ of processes is given by the following abstract syntax*

$$
\begin{array}{lll}
m, n, p & \in \Lambda & \textit{(Names)} \\
 & & \\
P, Q, R & ::= & \\
 & \mathbf{0} & \textit{(Void)} \\
 & P|Q & \textit{(Parallel)} \\
 & (\nu n)P & \textit{(Restriction)} \\
 & m\langle n\rangle & \textit{(Message)} \\
 & m(n).P & \textit{(Input)} \\
 & !P & \textit{(Replication)}
\end{array}
$$

By following the standard notation we say that in Restriction $(\nu n)P$ and Input $m(n)$, the distinguished occurrence of the name n is *bound*, with scope the process P. We say that n is *free* in a process P if there is an occurrence of n in P which is not bound. We identify processes up to α-congruence ($\equiv_\alpha$), i.e. we do not distinguish processes up to the safe renaming of bound names. Moreover with $P[m/n]$ we denote the replacement of the free (possibly) occurrence of name n in P by the name m (use the same idea to treat the formula $A[m/n]$).

Here we consider an operational semantics based on syntactic congruence and unlabelled transition systems.

Definition 2 (**Structural congruence**). *Structural congruence, noted $\equiv$, is the least congruence on processes such that*

$$
\begin{array}{lll}
P \equiv_\alpha Q \implies P \equiv Q & P|\mathbf{0} \equiv P & P|Q \equiv Q|P \\
P(Q|R) \equiv (P|Q)|R & !\mathbf{0} \equiv \mathbf{0} & !P \equiv !P|P \\
!(P|Q) \equiv !P|!Q & !!P|!P & (\nu n)\mathbf{0} \equiv \mathbf{0} \\
(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P & & \\
n \text{free in } P \implies P|(\nu n)Q \equiv (\nu n)P|Q & & \\
n \neq p, n \neq m \implies (\nu n)p(n).P \equiv p(m).(\nu n)P & &
\end{array}
$$

Definition 3 (**Reduction**). *Reduction is the least binary relation $\to$ on processes inductively defined as follows*

$$
\begin{array}{ll}
m\langle n\rangle|m(p).P \to P[n/p] & Q \to Q' \implies P|Q \to P|Q' \\
P \to Q \implies (\nu n)P \to (\nu m)Q & P \equiv P', P' \to Q', Q' \equiv Q \implies P \to Q
\end{array}
$$

3.2. **Syntax and semantics of Spatial Logic.** We here consider only propositional Spatial Logic, i.e. a first order logic without quantification on formulae. Basic constructs of the Spatial Logic include logical, spatial and temporal operation.

Definition 4. *Given an infinite set $\Lambda = \{m, n, p \ldots\}$ of names and a countable set $\mathcal{V} = \{x, y, z \ldots\}$ of name variables, formulae are defined as follows (let $\eta \in \Lambda \cup \mathcal{V}$):*

$$
\begin{array}{lll}
A, B, C & ::= & \\
 & F & \textit{(False)} \\
 & A \wedge B & \textit{(Conjunction)} \\
 & A \Rightarrow B & \textit{(Implication)} \\
 & \mathbf{0} & \textit{(Void)} \\
 & A|B & \textit{(Composition)} \\
 & A \triangleright B & \textit{(Guarantee)} \\
 & \eta \circledR A & \textit{(Revelation)} \\
 & A \oslash \eta & \textit{(Hiding)} \\
 & \eta\langle\eta'\rangle & \textit{(Message)} \\
 & \forall x.A & \textit{(Universal quantification)} \\
 & \not|x.A & \textit{(Frresh name quantification)} \\
 & \Diamond A & \textit{(Next step)}
\end{array}
$$

Next we define the semantics of the Spatial Logic, without entering into details. The basic idea consists in assigning to each formula A a set of processes $[\![A]\!]$, namely the set of all processes that satisfy the property denoted by A.

However not any set of processes can denote a property in a proper way. For instance, it is sensible to require $[\![A]\!]$ to be closed under structural congruence. Moreover suppose we have $P \in [\![A]\!]$ with n free in P, but not free in A, in this case we say that the free occurrence of n in P are *fresh* w.r.t. formula A. So the particular choice of the name n should not depend on A itself, hence it is natural to consider that all the fresh names w.r.t. A are to be treated uniformly.

Definition 5 (Semantics). *The denotational map $[\![A]\!]$ is the function that assigns a subset of $\mathbb{P}$ to each name-closed formula A. It is inductively defined as follows:*

$$
\begin{aligned}
[\![F]\!] &\stackrel{\text{def}}{=} \emptyset \\
[\![A \wedge B]\!] &\stackrel{\text{def}}{=} [\![A]\!] \cap [\![B]\!] \\
[\![A \Rightarrow B]\!] &\stackrel{\text{def}}{=} \{P : \textit{if } P \in [\![A]\!] \textit{ then } P \in [\![B]\!]\} \\
[\![\mathbf{0}]\!] &\stackrel{\text{def}}{=} \{P : P \equiv \mathbf{0}\} \\
[\![A|B]\!] &\stackrel{\text{def}}{=} \{P : P \equiv Q|R \textit{ for any } Q \in [\![A]\!] \textit{ and } R \in [\![B]\!]\} \\
[\![A \triangleright B]\!] &\stackrel{\text{def}}{=} \{P : \textit{if } Q \in [\![A]\!] \textit{ then } Q|P \in [\![B]\!]\} \\
[\![n \circledR A]\!] &\stackrel{\text{def}}{=} \{P : P \equiv (\nu n)Q \textit{ for any } Q \in [\![A]\!]\} \\
[\![A \oslash n]\!] &\stackrel{\text{def}}{=} \{P : (\nu n)P \in [\![A]\!]\} \\
[\![m\langle n\rangle]\!] &\stackrel{\text{def}}{=} \{P : P \equiv m\langle n\rangle\} \\
[\![\forall x.A]\!] &\stackrel{\text{def}}{=} \bigcap_{n \in \Lambda} [\![A[n/x]]\!] \\
[\![\not|x.A]\!] &\stackrel{\text{def}}{=} \bigcup_{n} ([\![A[n/x]]\!] \setminus \{P : n \textit{ is free in } P\}) \textit{ with } n \textit{ not free in } A \\
[\![\Diamond A]\!] &\stackrel{\text{def}}{=} \{P : P \to Q \textit{ and } Q \in [\![A]\!]\}
\end{aligned}
$$

Definition 6 (**Basic Derived Connectives**). *Some derived connectives of basic interest are defined as shown next.*

$$\begin{array}{rcll} \neg A & \stackrel{\text{def}}{=} & A \Rightarrow F & \textit{(Negation)} \\ T & \stackrel{\text{def}}{=} & \neg F & \textit{(True)} \\ A \vee B & \stackrel{\text{def}}{=} & \neg A \rightarrow B & \textit{(Disjunction)} \\ A||B & \stackrel{\text{def}}{=} & \neg(\neg A|\neg B) & \textit{(Decomposition)} \\ \exists x.A & \stackrel{\text{def}}{=} & \neg\forall x.\neg A & \textit{(Existential quantification)} \\ ©\eta & \stackrel{\text{def}}{=} & \neg\eta ® T & \textit{(Free names)} \\ \eta = \eta' & \stackrel{\text{def}}{=} & \neg(©\eta \triangleright (©\eta \oslash \eta')\,) & \textit{(Equality)} \\ \Box A & \stackrel{\text{def}}{=} & \neg\Diamond\neg A & \textit{(All next)} \end{array}$$

3.3. **Secure Process Algebra (SPA).** SPA syntax is based on a set of *visible* actions $\mathcal{L} = I \cup O$, where $I = \{a, b, \ldots\}$ is a set of *input* actions and $O = \{\bar{a}, \bar{b} \ldots\}$ is a set of *output* actions. A special action τ models internal computations, namely actions that are not visible outside the system. The complementation function $\bar{\cdot} : \mathcal{L} \rightarrow \mathcal{L}$ connects a with $\bar{a}$ for all $a \in I$, $\bar{a}$ with $\bar{\bar{a}} = a$ for all $a \in O$, and τ with $\bar{\tau} = \tau$. The set of all *action* is $Act = \mathcal{L} \cup \{\tau\}$. The set $\mathcal{L}$ of visible actions is partitioned into two sets: H and L of high and low level actions, such that $\overline{H} = H$, $\overline{L} = L$, $H \cup L = \mathcal{L}$, and $H \cap L = \emptyset$. The syntax of SPA *agents* is defined as follows:

$$E ::= \mathbf{0} \mid a.E \mid E + E \mid E|E \mid E \setminus v \mid E[f] \mid Z$$

Where $a \in Act$, $v \subseteq \mathcal{L}$, $f : Act \rightarrow Act$ is such that $f(\bar{a}) = \overline{f(a)}$ and $f(\tau) = \tau$, and Z is a constant that must be associated with a definition $Z \stackrel{\text{def}}{=} E$.

Intuitively, $\mathbf{0}$ is the void process that does nothing; $a.E$ is a process that behaves as E after performing an action a; $E + E$ represents the non deterministic choice between two processes; $E|E$ represent the parallel composition, where executions are interleaved, possibly syncrhronized on complementary input/output actions, producing an internal action τ; $E \setminus v$ represent a process prevented from performing actions in v; $E[f]$ is the process E whose actions are renamed via relabelling function f. We indicate with $\mathcal{E}$ the set made up of SPA agents, and with $\mathcal{E}_H$ the set of agents that can perform only high actions.

The operational semantics of SPA is the LTS $(\mathcal{E}, Act, \rightarrow)$, where the states are the terms of the algebra and the transition relation $\rightarrow \subseteq \mathcal{E} \times Act \times \mathcal{E}$ is defined by structural induction as the least relation generated by the following axioms and inference rules:

Prefix
$$\frac{-}{a.E \stackrel{a}{\rightarrow} E}$$

Sum
$$\frac{E_1 \stackrel{a}{\rightarrow} E_1'}{E_1 + E_2 \stackrel{a}{\rightarrow} E_1'} \qquad \frac{E_2 \stackrel{a}{\rightarrow} E_2'}{E_1 + E_2 \stackrel{a}{\rightarrow} E_2'}$$

Parallel
$$\frac{E_1 \stackrel{a}{\rightarrow} E_1'}{E_1|E_2 \stackrel{a}{\rightarrow} E_1'|E_2} \qquad \frac{E_2 \stackrel{a}{\rightarrow} E_2'}{E_1|E_2 \stackrel{a}{\rightarrow} E_1|E_2'} \qquad \frac{E_1 \stackrel{a}{\rightarrow} E_1' \quad E_2 \stackrel{\bar{a}}{\rightarrow} E_2'}{E_1|E_2 \stackrel{\tau}{\rightarrow} E_1'|E_2'}$$

Restriction
$$\frac{E \stackrel{a}{\rightarrow} E'}{E \setminus v \stackrel{a}{\rightarrow} E' \setminus v} \text{ if } a \notin v$$

Relabelling
$$\frac{E \stackrel{a}{\rightarrow} E'}{E[f] \stackrel{f(a)}{\rightarrow} E'[f]}$$

Definition 7 (**Strong Bisimulation**). *A binary relation $\mathcal{R} \subseteq \mathcal{E} \times \mathcal{E}$ over agents is a strong bisimulation if $E\mathcal{R}F$ implies, for all $a \in Act$,*

- *if $E \xrightarrow{a} E'$, then there exists F' such that $F \xrightarrow{a} F'$ and $E'\mathcal{R}F'$;*
- *if $F \xrightarrow{a} F'$, then there exists E' such that $E \xrightarrow{a} E'$ and $E'\mathcal{R}F'$.*

Two agents E, F are strongly bisimilar, denoted $E \sim F$, if there exists a strong bisimulation $\mathcal{R}$ such that $E\mathcal{R}F$.

A weak bisimulation is a bisimulation which does not care about internal τ actions, that corresponds to synchronization inside the system. So, when F simulates an action of E, it can also execute some τ actions before or after that action. We consider another transition $\stackrel{\hat{a}}{\Longrightarrow} \subseteq Act \times \mathcal{L} \times Act$ such that $E \stackrel{\hat{a}}{\Longrightarrow} F$ denotes $E(\xrightarrow{\tau})^* \xrightarrow{a} (\xrightarrow{\tau})^* F$ if $a \neq \tau$, and $E(\xrightarrow{\tau})^* F$ if $a = \tau$ (where $(\xrightarrow{\tau})^*$ denotes a – possibly empty – sequence of τ labelled transitions).

Definition 8 (**Weak Bisimulation**). *A binary relation $\mathcal{R} \subseteq \mathcal{E} \times \mathcal{E}$ over agents is a weak bisimulation if $E\mathcal{R}F$ implies, for all $a \in Act$,*

- *if $E \xrightarrow{a} E'$, then there exists F' such that $F \stackrel{\hat{a}}{\Longrightarrow} F'$ and $E'\mathcal{R}F'$;*
- *if $F \xrightarrow{a} F'$, then there exists E' such that $E \stackrel{\hat{a}}{\Longrightarrow} E'$ and $E'\mathcal{R}F'$.*

Two agents E, F are weakly bisimilar, denoted $E \approx F$, if there exists a weak bisimulation $\mathcal{R}$ such that $E\mathcal{R}F$.

In [20] it is proved that $\sim$ is the largest strong bisimulation, $\approx$ is the largest weak bisimulation, and they are equivalence relations.

Next we give two definition of noninterference in SPA.

Definition 9 (**BNDC**). *Let $E \in \mathcal{E}$.*

$$E \in BNDC \quad \textit{iff} \quad \forall \Pi \in \mathcal{E}_H, E \setminus H \approx (E|\Pi) \setminus H.$$

Definition 10 (**Persistent_BNDC**). *Let $E \in \mathcal{E}$.*

$$E \in P\_BNDC \quad \textit{iff} \quad \forall E' \textit{ reachable from } E \textit{ and } \forall \Pi \in \mathcal{E}_H, \\ E' \setminus H \approx (E'|\Pi) \setminus H, \textit{ i.e. } E' \in BNDC.$$