Mathematical Models of Computer Security

Peter Y. A. Ryan

The Software Engineering Institute, Carnegie Mellon University
Pittsburgh, PA 15213
pryan@cert.org

Abstract. In this chapter I present a process algebraic approach to the modelling of security properties and policies. I will concentrate on the concept of *secrecy*, also known as *confidentiality*, and in particular on the notion of non-interference. Non-interference seeks to characterise the absence of information flows through a system and, as such, is a fundamental concept in information security.
A central thesis of these lectures is that, viewed from a process algebraic point of view, the problem of characterising non-interference is essentially equivalent to that of characterising the equivalence of processes. The latter is itself a fundamental and delicate question at the heart of process algebra and indeed theoretical computer science: the semantics of a process is intimately linked to the question of which processes should be regarded as equivalent.
We start, by way of motivation and to set the context, with a brief historical background. A much fuller exposition of security policies in the wider sense, embracing properties other than secrecy, can be found in the chapter by Pierangela Samarati in this volume. We then cover some elements of process algebra, in particular CSP (Communicating Sequential Processes), that we need and present a formulation of non-interference, along with some more operational presentations of process algebra, including the idea of bi-simulation. I argue that the classical notion of *unwinding* found in the security literature is really just bi-simulation in another guise.
Finally, I propose some generalisations of the process algebraic formulations designed to encompass a richer class of policies and examples.

1 Background

This chapter presents a process algebra based framework in which we can express and analyze security requirements at an abstract level. I hope that the reader will come away with the impression that such an approach is well suited to formulating security properties. Many issues that have proved problematic in the formulation of, for example, secrecy in information processing systems, become much clearer when viewed in a process algebraic style. Many insights and results from the process algebra community turn out to be highly relevant in the context of information security. On the other hand, information security presents a number of challenges to current theory and so should help stimulate advances in theory.

R. Focardi and R. Gorrieri (Eds.): FOSAD 2000, LNCS 2171, pp. 1–62, 2001.

The term *security* is often used to cover a multitude of requirements, in particular:

- Secrecy (confidentiality)
- Integrity
- Availability (e.g., resilience to denial-of-service attacks).

By *secrecy* or *confidentiality* I mean, informally, that information can only be acquired by agents or processes entitled to such access. By and large I will regard the task of a policy to be to define when access should be allowed or denied. Integrity, roughly speaking, will mean that the *correctness* of data is ensured: i.e., it can only be established or modified by agents or processes entitled to influence the values of the data. Availability typically means that access to information and services to agents with the right to them is maintained in a timely and dependable manner.

Pierangela has given the background to these concepts in her chapter of this volume so we will not dwell on the various flavours that exist in the security literature. For the most part I will concentrate on secrecy for these lectures but I will touch on the other requirements. Indeed, to some extent at least, other requirements can be captured in a rather similar framework as variants of non-interference.

There has been much debate in the security literature as to what exactly is meant by the terms *security model* or *security policy* and indeed what, if any, is the distinction. I do not propose to enter such debate in these lectures, but refer the interested reader to the excellent and lucid writings of McLean, for example [57], on this and the subject area in general. For the purposes of these lectures I will take the attitude that the purpose of a *policy* is to state what information flows are to be allowed and which are to be prevented. More generally a policy will state what privileges are accorded to which agents. I will regard a *model* as being a mathematical framework in which we can precisely characterise the properties of interest, in particular that of secrecy, i.e., the absence of certain information flows.

Another much debated question is that of whether a "correct," Platonic notion of security, or at least secrecy, exists. Again I will avoid being drawn into the rather philosophical aspects of such discussions. We will see later, however, that even the apparently rather well focussed question of characterising information flows, and in particular their absence, in a system is surprisingly delicate, but for precise mathematical reasons rather than philosophical ones.

In these lectures I am principally concerned with presenting definitions security properties such as secrecy. Such definitions are of little use if we do not have ways to demonstrate that actual designs and systems meet the defintions. I will discuss some of the issues involved in going from the high-level definitions towards implementations. This turns out to be distinctly non-trivial. Step-wise development techniques are well established for so-called *safety* properties but

it is well known that security properties tend not to be preserved by such techniques. Safety properties typically amount to assertions that a system will not perform such and such an undesirable behaviour. As we will see later, security properties are far more subtle and cannot be captured by simply outlawing certain behaviours.

The next two sections provide some motivation for the use of mathematical models. Section 4 gives a brief overview of the historical development of models of computer security. This will help to put the central theme, the concept of *non-interference*, in context. Section 5 presents the Goguen-Meseguer formulation of non-interference along with some discussions of the limitations of this formulation. This is usually cited as the primary reference for non-interference although there is some prior work due to Cohen and Feiertag. This is followed by a brief introduction to process algebra, in particular CSP, which will be used to give more up-to-date formulations of non-interference. Finally I present some generalisations of these formulations and topics for further research.

2 Mathematical Models

Before we launch into descriptions of mathematical models, a few words are in order on why such models are needed at all. The purpose of (mathematical) models is to provide abstract descriptions of a more complex reality. The models will typically ignore many details of the real artifact in order to render them understandable and amenable to analysis. Care has to be taken to ensure that such abstractions are not so drastic as to remove any semblance of reality.

Usually we are only interested in certain aspects of the real artifact that are wholly or largely independent of many of the details. For example, in thermodynamics physicists study the behaviour of a gas in a box. Suppose that the gas comprises n molecules, then the state of the gas is represented by a point in a $6n$ dimensional phase space. However we are not remotely interested in the exact point in the phase space the gas actually occupies. Even if we could accurately determine this point it would actually tell us nothing of interest. It is enough for most purposes just to consider the state of the gas to be a function of three parameters: temperature, volume and pressure and to study the relationship between these. The equations of motion on the phase space lift to relations between these parameters.

Similarly in computer science we are dealing with extremely complex artifacts, many of whose details are of no interest to us or are unobservable. Again we use mathematical models which, by their very nature, abstract much of the extraneous detail or are suitably constructed to allow us to make abstractions as appropriate. In these lectures we will be concentrating on process algebras as our mathematical framework and we will see that these give rise rather naturally to a representation of systems that corresponds to what is observable by some outside observer. Process algebras also lead to a number of very natural and powerful abstraction operators. We will see, for example, how the notion of secrecy leads to equivalences on the state space of a system rather analogous to

the way a thermodynamical system is factored by equivalences of temperature, pressure and volume.

In short, the use of mathematical models allows us to *introduce simplicity*, to borrow a phrase from Abrial [76]. We can then deploy all the usual mathematician's tricks of abstraction, modularisation, symmetry, etc. to reduce the problem to mind-sized chunks. All the while we have to be careful that we have not simplified too much. This is an ever present danger, especially for security, where the old dictum: "the devil is in the detail," is particularly true.

Another beauty of mathematical models is that they are completely at your mercy: you can do what you like with them, up to requirements of consistency. You can change parameters with an ease that would be impossible with the real thing or even a physical mock up. You can perform *Gedankenexperimente*: thought experiments that would be impossible in reality, along the lines of Einstein's riding a light beam.

Another motivation for models is that they allow us to move from an abstract representation gradually, in a step-wise fashion towards an implementation. The hope is that by making the steps reasonably modest we ensure that the complexity of our manipulations and proofs is kept manageable throughout. This tends to be rather easier said than done, especially for security, but it is nonetheless a worthy and sensible goal.

We also need to bear in mind that absolute security is probably a theoretical impossibility and certainly a practical impossibility. Ultimately we need to aim to provide adequate levels of security against probable threats, enough for to ensure that the cost to the attacker of launching an attack outweighs the benefits. In the world of physical security people have long known how to rate safes in terms of the time they can resist certain types of attack. So, you choose the appropriate safe in terms of its resistance to attack and in terms of the likely value of its contents. Such ratings can be blown away by a new mode of attack and the history of safe design has been a game of cat and mouse between the safe designers and the safe crackers. When a new style of attack is invented, a few safes get blown, and new designs are deployed.

The story is rather similar in the information security world, but there are differences. The game is a far more complex one: the systems in question are vastly complex and the styles of attack vastly more diverse. Often the stakes are much higher. If we are concerned with critical infrastructures for example, we can't just insure everything with Lloyd's of London and upgrade our defences after a cyber terrorist has taken the whole system out.

3 Formal Models and Methods

A few words are also in order on the nature and utility of formal models and formal methods. The idea of formality in mathematics goes back a long way, to around the time of Hilbert. The motivation was to flush out implicit and unstated assumptions in proofs. Most rigorous, journal-style proofs will involve quite large steps whose justification involves understanding and insight into the

problem domain. Typically, the detailed justification of such steps can be readily filled in by a competent mathematician. Occasionally errors show up later or, more usually, it is found that the justification actually requires some additional assumption not explicitly stated in the original proof. The delightful book "Proofs and Refutations", [42], by Lakatos illustrates this process beautifully by tracing "proofs" of the Euler formula relating the number of edges, vertices and faces of polyhedra. Early "proofs" were published only for counter-examples to be found later. Closer examination revealed that the "proofs" actually depended on certain unstated assumptions, assumptions that were violated by the counter-examples. Thus, for example, it had unwittingly been assumed that the polyhedra were convex and had a simple topology, i.e., genus zero. The counter-examples, Kepler stars or toroidal polyhedra, etc., violated these conditions.

This is rather troubling and raises the question: when is a proof a proof? The notion of a formal system and a formal proof is an attempt to answer this question. A formal system comprises a finite set of axioms along with a finite set of inference rules. A theorem is a statement that can be reached from the axioms by finite application of the rules. No further insight or creativity is admitted other perhaps than some ingenuity in discovering the right sequence of application of rules. In principle, then, any theorem is checkable in a purely mechanical fashion and there is no way for implicit assumptions to slip into the process.

Why is all this of interest in systems engineering? Why is it not enough just to use rigorous pen-and-paper proofs, for example? There seem to be two motivations: again, the urge to flush out all unstated assumptions and, secondly, to introduce the possibility of mechanised proofs using automated tools. Are these valid motivations? The answer seems to be: *sometimes yes*. Sometimes the design of a system or component is so critical that it really is crucial to eliminate the possibility that unrecognised and flawed assumptions could creep in. Sometimes having an automated tool really does help, in the sense that it reduces the amount of work and/or increases the resulting level of assurance. But it is essential to use the right tool in the right place. The essential question is, as Bob Morris Sr., has phrased it: "Where do I put my extra five bucks of verification money?"

Unfortunately, the history of formal methods is littered with instances of a compulsion to perform a mechanised proof even where this has been vastly expensive and has added little or nothing of benefit. You really have to ask yourself why you are attempting to prove something and, in particular, why with a mechanised proof. The validity of the result has to matter. If you are after insight into why the result holds, you are probably better off with a rigorous, pen-and-paper proof. Often the kinds of theorems that crop when performing a refinement as *proof obligations* (or *opportunities* as Jim Woodcock prefers to call them) involve rather tedious and error-prone case enumeration. This is the kind of thing theorem-provers or model-checkers are actually rather good at. For the more interesting theorems the tools can all too easily get in the way. Mackenzie, for example, discusses such issues in depth [50]. Often the mere

process of thinking about how to cast a design in a form ripe for analysis reveals ambiguities, flaws and insights long before any proofs are actually attempted.

We hasten to reassure the reader that none of the results presented here have been subjected to the ignominy of mechanical proof.

4 A Brief History of Security Models

I will concentrate on the concept of non-interference and related models of security, but to help put this in context I give a very swift overview of the early evolution of computer security models in this chapter.

The topic of security models goes back some 30 years, starting with papers by Lampson, for example, that presented ways of formalizing access controls [43]. These models comprised a set of subjects S, objects O and access modes A. From a security point of view each state of the system corresponds to an access control matrix M whose rows correspond to the subjects, and columns to objects. Entries in the matrix are subsets of A and represent the allowed access of the subject to the object. Various rules constrain how states can evolve, i.e., how the entries of the matrix can be updated.

Such early work was based on analogies with the existing policies and mechanisms for the pen-and-paper world. The early work was also very much inspired by (or perhaps more accurately, funded by) the military/government environment.

In the pen and paper world, we traditionally have the notions of classified files and cleared individuals. A simple access control policy is then formulated in terms of these classifications and clearances. Thus a file might be classified *secret* and so only accessible to someone with *secret* clearance or higher. The traditional mechanisms for enforcing such policy might be a trusted registrar who would check the clearance of anyone requesting a particular file and who will only hand it over after the clearance has been verified.

Classification and clearances, that we will refer to collectively as *security levels*, are typically organised in a lattice structure. A particularly simple example is a linear hierarchy:

Top secret
—
Secret
—
Confidential
—
Unclassified

Each element of this hierarchy is said to dominate those below it, as well as itself. You are allowed to view a file if your clearance dominates the classification of the file. Thus someone with *Secret* clearance is allowed to view *Secret*, *Confidential* or *Unclassified* files but not *Top secret*, for example.

More elaborate, non-linear lattices representing partial orders are possible and indeed common, at least in the military world. These correspond to the notion of compartmentalized information and the *need to know* principle. Thus a file might carry not only a classification but also a so-called *caveat*. For example, NATO-Secret, UK-Secret etc.

The lattice might then take the form:

NATO-Secret and UK-Secret are incompatible and so someone with UK-Secret would not be entitled to view a NATO-secret file and vice versa. Someone with *Top Secret* could view both. Such lattices form the basis of the so-called Multi-Level-Secure (MLS) policies that were prevalent in the early work in computer security and provide a convenient way to define which information flows are deemed acceptable and which unacceptable.

Early models of computer security sought to map such policies and mechanisms across into the information technology world. The policies map across reasonably cleanly, though even here some subtleties do arise which we will return to shortly. Mapping the mechanisms proves to be distinctly tricky. In the pen-and-paper world the means of access to files was pretty simple: you wandered over to a registry and persuaded Miss Moneypenny that you had the appropriate clearance to view a particular file. For distributed information processing systems there is a far richer set of modes and mechanisms for accessing data and the controls on such access are quite different. Also, access may not be so direct but might be mediated by various intermediate programs.

4.1 The Bell and LaPadula Model

The model proposed by Bell and LaPadula, (BLP), is one of the earliest and best known models [2]. It is embodied in the famous *Orange Book* put out by the NCSC (National Computer Security Centre), a branch of the NSA (National Security Agency) [15]. The Orange Book was the first attempt to establish systematic criteria for the evaluation of secure systems and products.

BLP is fairly straightforward and intuitive and is a rather natural analogue of MLS policies of the pen-and-paper world. I give a somewhat simplified version here; more formal and complete descriptions can be found in [29], for example.

The model comprises a set of subjects S and objects O. Subjects are thought of as active, either principals or processes acting on behalf of principals, whilst objects are passive, e.g., files. We suppose that a lattice L of security levels has

been defined along with a mapping C from $S \cup O$ into L that assigns a clearance/classification to each subject/object. For simplicity we will suppose that there are just two modes of access: *read* and *write*. The information-flow policy is now captured by enforcing two rules:

The Simple Security Property :

A subject s is allowed read access to an object o if and only if $C(s)$ dominates $C(o)$.

That is, a subject is allowed to read only objects whose classification is dominated by its clearance. In particular a subject cannot read an object of higher classification, a requirement often referred to as *no read up*.

The * Property :

A subject s is allowed write access to an object o if and only if $C(o)$ dominates $C(s)$.

That is a subject is not allowed to write to an object whose classification is lower than its clearance, i.e., *no write down*.

Assuming that read and write really are the only modes of access to objects and, furthermore, that they really are one-way information flows in the direction we would intuitively expect, then we see that together these rules ensure that information cannot flow downwards through the lattice. Information can flow upwards.

Such a policy is referred to as a *mandatory access control* (MAC) policy. Subjects are not allowed any discretion with respect to these rules or to the classifications assigned to objects. Another class of access policies, in which subjects are accorded some discretion, can be constructed. Such policies are referred to as *discretionary access control* (DAC) policies. They might permit, for example, the owner or creator of a file some discretion as to what classification to assign it or to whom he wishes to grant access. A more elaborate version of BLP can be used to formulate DAC policies but this will not concern us. See Pierangela's chapter for more on this.

In practice the MLS model above is too rigid for real environments. In the pen-and-paper world there are typically rules and mechanisms allowing exceptions to the strict MLS access rules. You might, in some operational circumstances, want to allow a certain agent access to a file to which he would not ordinarily have access according to the MLS rules. Alternatively there may be reasons to lower the classification of a file, perhaps after some period has elapsed after which its contents are no longer deemed sensitive. Such exceptions would typically be handled by a security officer.

In the BLP model the handling of such exceptions are assigned to so-called *trusted subjects*. Exactly how such trusted subjects are implemented and indeed exactly what the term *trusted* means here has been the subject of much debate. The issue will crop up again when we come discuss intransitive non-interference.

4.2 The Harrison-Ruzzo-Ullman Model

The model proposed by Harrison, Ruzzo and Ullman (HRU) is also based on Lampson's access-control-matrix framework, but allows a far richer set of primitives for updating the state than for BLP [32]. In particular, subject to preconditions on the state, entries in the matrix can be added or removed, and indeed rows and columns can be added or deleted (corresponding to the creation or deletion of subject and objects). In effect we have an elaborate, conditional rewriting system. Rather unsurprisingly then we rapidly hit against various undecidablity results: in general establishing whether a certain state can be reached, i.e., whether a certain access can be granted, is undecidable.

BLP sacrifices flexibity and genericity for simplicity and decidability. Questions of whether a flow from a particular object to a particular subject is permitted can be immediately answered by simple comparison of their classification and clearance, ignoring for the moment the actions of trusted subjects.

4.3 Chinese Walls

Chinese walls policies arise naturally in a commercial setting, for example, in a consultancy firm. The firm will consult to various clients. We want to ensure that any given consultant, C say, does not gain access to sensitive information of two competing clients, A and B say. This is enforced by stipulating that should C get access to $A's$ files he should subsequently be barred from access to $B's$ files and vice versa. In effect we need to enforce appropriate mutual exclusion rules for clients with potentially conflicting interests.

Such policies can be formulated in the HRU model and can be modelled in the MLS framework as long as we allow for dynamic clearances, see [22,81]. Here accesses evolve: initially C has access rights to A or B but as soon as he exercises his right to $A's$ files, say, then his right to $B's$ is deleted, or vice versa. Notice, however, that rights are monotonically non-increasing.

Brewer and Nash proposed introducing such policies and models into computer security [5]. They are an instance of a more general class of policies known as *dynamic separation of duties* policies. Here a principal can take on certain roles, but there are certain mutual exclusions between roles. Thus if he takes on a role as a bank teller he cannot also countersign cheques, for example. The purpose of such rules are to try to prevent abuse of privilege by ensuring that a misdemeanor cannot be performed by a single individual and would require collusion.

4.4 The Clark Wilson Model

Clark and Wilson propose a significantly more elaborate framework designed to capture policies of relevance in a commercial environment [9]. This embraces not only confidentiality and integrity requirements but also notions of separation of duties and well-formed transactions. Rather than the access-control lists or matrices of the previous models they use *access-triples*. These define the programs

(well-formed transactions) that a user may invoke to access certain data objects. Good references for further reading are: [46,86,23].

4.5 The Biba Model

Thus far we have only considered secrecy, or if you prefer, confidentiality. In some applications, ensuring the *integrity* of data is also often of concern. A simple form of integrity policy was proposed by Biba which is really just a dual of BLP [4]. The elements of the model are identical to BLP except that we invert the *simple* and * properties and think of the lattice as representing integrity levels rather than security levels. The effect then is to ensure that information cannot flow up in terms of the integrity lattice. In other words a low-integrity subject cannot alter an object whose integrity level is higher.

4.6 Drawbacks of BLP

BLP has the advantage of being intuitively appealing and simple to understand. The fact that it is so simple and intuitive, however, conceals a number of subtleties and problems.

Firstly it relies heavily on our intuition as to the meaning of the terms *read* and *write*. These are not given precise, let alone formal, definitions but it is clear that it is being assumed that they constitute one-way flows of information. Thus if Anne reads a file X we are assuming that information only flows from X to Anne and that there is no flow from Anne to X. Similarly if Anne writes to X we are assuming that there is no flow from X to Anne. This sounds plausible and in line with our intuitive undestanding of the terms. However, interactions are rarely one-way, particularly in a distributed context. Communication usually involves protocols with messages passing back and forth to establish and maintain channels.

Let us take an extreme example: devices like the CESG One-Way-Regulator or the NRL Pump, [40], were developed to enforce one way flow of information, from L to H, say. Even these devices require some regulatory signals flowing from H to L to avoid buffer overflows and other problems. Thus, even here, there is two way flow of information, albeit of a very low bandwidth in one direction.

Our intuitions can be deceptive. Consider an even more extreme example. We could completely invert the usual semantics of *read* and *write*. This would give an entirely consistent BLP-style model but systems that satisfied it would be the antithesis of what we had in mind for a system that maintains secrecy. In fact, essentially this occurs when we try to define the notion of integrity as in the Biba model above. The saying "people in glass houses shouldn't throw stones" makes good sense, but then so does: "people who live in stone houses shouldn't throw glasses."

A further difficulty stems from the fact that when we come to map the model down onto a real system we have to try to identify *all* the channels by which subjects can access objects. In a complex system it is clearly difficult to

be sure that all such channels have been identified. This, along with the issue raised earlier-that even supposing that we have identified all the channels we may still have failed to model them accurately-makes analysis very difficult. Thus there may easily be information flows in the system that we fail to identify and that might then allow illegal information flows to go undetected. Arguably things are better in an object-oriented framework in that the methods should constitute the full set of access modes to an object. In fact, closer examination of implementations reveals other access channels not explicitly represented at the object-oriented level of abstraction.

Consider a further example: suppose that a user with a low clearance requests to create a file of a given name and suppose that a highly classified file of the same name already exists. The system might well reply to the request with a "request denied" or, arguably worse still: "request denied, file of that name already exists." This results in an information flow from H to L. It may not be a particularly useful flow as it stands but it does represent a channel that could potentially be exploited by some malicious code executing at the high level to signal to a low process. Such malicious code is referred to as a *trojan horse*.

Such channels are well known and standard solutions to this particular problem have been proposed-for example, using poly-instantiation: allowing a file of a given name to have instantiations at several security levels. However, it is also clear that the system may well harbour many other, possibly far more subtle flows of this kind. Such channels are known as covert channels and they typically arise from the sharing of resources.

The traditional reaction to such problems is to perform *covert channel analysis*. Conceptually, the known channels are severed and the system is then studied to see what channels across security-relevant boundaries remain, [41,54,55]. These can either be eliminated or, if they are difficult or impossible to eliminate, then their channel capacity can be limited to an acceptable level. Thus, for example, random delays could be introduced on response signals to introduce noise and so lower the channel capacity, see [65] for example. This tends to have a tradeoff in terms of performance but such tradeoffs are all too familiar in security in any case.

Further objections to BLP have been raised, for example McLean's *System Z* but these need not concern us here [58]. BLP stands as an important and seminal work in the area of security models and indeed continues to play a useful role in the design and evaluation of secure systems. In the next section we will present a radically different approach to modelling computer security that attempts to address some of these objections: the notion of non-interference.

5 Non-interference

5.1 Goguen Meseguer

In response to the concerns raised about BLP and access control style models in general, Goguen and Meseguer in, based on some earlier work of Feiertag and

Cohen, proposed the idea of *non-interference* [26,27,18,11]. It can be thought of as an attempt to get to the essence of what constitutes an information flow in a system and, more to the point, how to characterise the absence of any flow. In this sense it resides at a more abstract level than the access-control models and can be thought of as providing a formal semantics to the one-way-flow intuition behind terms like *read* and *write*. In particular, it abstracts completely from the inner workings of the system in question and formulates the property purely in terms of user interactions with the system interfaces.

The underlying idea will be familiar to anyone with a mathematics background as it is really a reworking in the context of simple model of computation of the standard device for characterising a function's independence with respect to a particular variable.

The model of computation assumed, at least in the original formulation, is a rather simple one. In particular it assumes that all computations are deterministic. In particular the outputs are a simple function of the state and the input. That is, given a state along with an input the resulting output is unique and well defined. We will come to non-deterministic systems a little later. Non-interference seeks to formalise the intuition that the interaction of high-level users with a system S should not influence the interactions of low-level users. We will paraphrase the original formulation to make the transition to later material a little smoother.

Histories of user interactions with the system will be recorded as traces, i.e., sequences of actions. We suppose that the set of users of the system is partitioned into two sets: high users and low users. We suppose further that the high and low users interact via separate interfaces and, in particular, low users cannot directly observe high actions. The whole point is to examine how much a Low user can infer about High actions purely from his (low's) interactions with the system. Clearly if he can directly observe the High actions the whole exercise becomes rather futile.

Notice that we will be assuming that we know the exact design of the system and, perhaps more importantly, that any hostile agents have a complete understanding of the system design. In practice neither we nor the hostile agents will have such a full understanding but it is a safe assumption to make: the less hostile agents know about the system design the less precise the inferences they can make. As with cryptography, we should not seek security through obscurity.

In accordance with the original formulation we will also assume that actions are partitioned into *inputs* and *outputs*. For the moment we will put aside the question of what the semantics behind this distinction might be. Intuitively we are thinking of inputs as being wholly under the control of the user, and outputs as wholly under the control of the system. This is similar to the BLP use of *read* and *write*.

We now think of the system as a finite state machine described by a function that, given a state and an input, returns a transition to a new state and an output. Given that we are assuming the system to be deterministic this really is a function, i.e., the state transition and output associated with an initial state

and input are uniquely defined. Furthermore, assuming a unique starting state of the system, we have a one-to-one correspondence between traces and states and so we can identify states with traces.

We now restrict the range of the output function to just outputs visible to Low. Thus:

$$Output_L(S, tr, i)$$

gives the Low output from system S when input i is applied to the state corresponding to trace tr.

The final piece of plumbing that we need is the notion of the *purge* of a trace. Informally, $purge_{HI}$ takes a trace and returns a trace with all High inputs, denoted HI, removed.

A system S is said to be non-interfering from High to Low iff:

$$\forall\, tr \in I^*, c \in I \bullet Output_L(S, tr, c) = Output_L(S, purge_{HI}(tr), c) \qquad (1)$$

In other words, for any possible history of inputs to the system and next input, the output visible to Low will be the same as if we had stripped out all the High inputs. Thus, changing High inputs leaves Low's view of the system unchanged. This is analogous to the standard way of defining a function's independence with respect to one of its variables.

It might seem that we have lost generality by assuming that the alphabet of the system is partitioned into High and Low. In fact we can deal with more general MLS-style policy with a lattice of classifications by a set of non-interference constraints corresponding to the various lattice points. For each lattice point l we define High to be the union of the interfaces of agents whose clearance dominates that of l. Low will be the complement, i.e., the union of the interfaces of all agents whose clearance does not dominate that of l. Notice also that we are assuming that we can clump all the high-level users together and similarly all the low-level users. There is nothing to stop all the low users from colluding. Similarly any high-level user potentially has access to the inputs of all other high users. We are thus again making a worst-case assumption.

Non-interference takes a very abstract view of the system, effectively treating it as a black box and formulating the property purely in terms of its external behaviours. This has advantages and disadvantages. It means that the definition is wholly independent of the details of the system, so, for example, we don't have to worry about what exactly are the internal information-flow channels. On the other hand its abstractness means that it is very remote from real implementations.

A further aspect of reality that has been abstracted away is time. Time simply does not appear in the definition or the underlying model of computation. We are, in effect, assuming that any adversaries have no way of observing the timing of actions. Similarly we have abstracted away from any issues of probability and are really just thinking in terms of what events are possible. We are thus working in what is sometimes referred to as a *possibilistic* framework. This is, of course, wholly unrealistic, but we have to start somewhere. We will discuss later how

one might extend the work presented here to address questions of time and probability.

Notice that non-interference is asymmetrical: we are saying nothing about how Low events might influence High outputs. Thus information is allowed to flow from Low to High. This is typically what we want for an MLS style policy.

Non-interference side steps the problems that BLP and other models have with covert channels, but does so at the expense of working at a very high level of abstraction. Leaving aside questions about input/output distinctions, non-interference will capture covert channels. Indeed the motivation for the approach was in large part to address the covert channel problem. In particular it addresses the possibility that a high user or process may deliberately try to signal to a low user.

Note that non-interference is really about characterizing the absence of causal flows rather than information flows. Absence of causal flow implies absence of information flow but there may be situations in which we have a causal flow without an associated flow of information. The canonical example of this is an encrypted channel. Here a secret plaintext will influence a ciphertext that is communicated over open channels: altering the plaintext input will alter the corresponding ciphertext output and so, naivly at least, non-interference is violated. However if the encryption algorithm is sufficiently strong, keys are not compromised etc, then we can think of this as not representing any information flow from classified to unclassified. We are ignoring for the moment questions of traffic analysis. Being sure that such a causal flow doesn't harbour some subtle information flow is very delicate and brings in considerations of cryptanalysis amongst other things. There are a number of delicate issues here and we will return to this point later.

5.2 Unwinding

Goguen and Meseguer also introduce the notion of *unwinding* a non-interference property. As it stands, non-interference is not very useful from a verification point of view as it involves quantification over all possible histories of the system. It is defined purely in terms of the system's interactions with the environment and without any reference to its internal construction etc. It thus constitutes an elegant and abstract definition is not easy to verify directly. Given a black box system, to determine if it obeys the non-interference property we would be reduced to attempting exhaustive testing, clearly an impossibility. In order to render the verification tractable we are forced to assume some (minimal) structure on the state of the system.

The idea of unwinding is to replace the original formulation with conditions on state transitions. It is analogous to the standard technique of defining an invariant that can then be used to prove a property of all reachable states via a structural-induction-style argument.

A few remarks are in order: it is necessary to introduce an equivalence relation over the states of the system. This is the relation induced by the purge function along with the correspondence between traces and states. Two traces

are regarded as equivalent if they have the same purges, i.e., are identical in their low-level actions and high outputs.

Unwinding is now stated as a pair of rules. A system S is non-interfering if:

$$\forall S_1, S_2 \in traces(S), a_1, a_2 \in A$$
$$\bullet\ purge_{HI}(a_1) = purge_{HI}(a_2) \wedge S_1 \approx S_2 \Rightarrow S_1' \approx S_2' \tag{2}$$

where S_1' denotes a state reached from S_1 after the action a_1 and similarly for S_2'. A denotes the full alphabet of S, i.e., inputs and outputs. Thus $A = I \cup O$ and:

$$\forall S_1 \approx S_2 and \forall a \in I \bullet Output_L(S_1, a) = Output_L(S_2, a) \tag{3}$$

The effect of the first rule is to ensure that the equivalence on the state space of S is exactly that induced by the purge of the traces. A simple induction on the length of traces along with the one-to-one correspondence between traces and states (given that S is deterministic) establishes this. That these together imply the original property now follows from this observation along with the second rule. In this simple context of a deterministic system it is also straightforward to show that these rules are necessary. Things become much more interesting when we extend such results to the non-deterministic context later.

5.3 Non-interference Is Not a Trace Property

An important observation is that non-interference is not a trace property, that is, it cannot be framed simply as a predicate on traces. More precisely, in order to determine whether a system satisfies non-interference we cannot examine it trace by trace to determine whether each trace satisfies a certain predicate. Many useful system properties can be stated as such predicates, so-called *safety* properties. These often arise in specifying safety-critical systems where we are asserting that certain undesirable behaviours should not be allowed, in which case we can formulate a predicate whose characteristic class is equal to, or perhaps is a subset of, the set of acceptable behaviours.

Non-interference is a property of the space of behaviours. It is really asserting that if certain behaviours can occur then it must be the case that other, related behaviours must also be possible. Take a trivial example. Suppose that an allowed behaviour is h followed by l, with h a High action, l a Low action. If the system is to be non-interfering then it must also be possible for just the l to occur without the prior h event. Whereas a trace property can be expressed as a set of *acceptable* behaviours, non-interference must be expressed as a set of sets of behaviours.

Conventional refinement techniques are designed to handle safety-style properties. A system Q is said to refine P, roughly speaking, if the allowed behaviours of Q are contained in P. A consequence of this observation is that non-interference tends not to be preserved by (conventional) refinement. This will be discussed more fully later.

5.4 Relationship to Bell LaPadula

It has been claimed that BLP and non-interference are equivalent, [31] for example. The proof depends on some rather strong assumptions about the system's commands and their correspondence with the access modes of the model. In general it is difficult to establish such a firm correspondence. On the other hand there is a sense in which BLP can be viewed as a mapping of a non-interference property to an implementation architecure. This mapping is necessarily rather informal, given the lack of precise semantics of BLP and the difficulty in identifying all internal channels of information flow in the actual architecture.

There are some interesting issues here: in moving from a non-interference formulation to an access-control formulation like Bell LaPadula we are somehow moving from a non-trace property to a trace property. The step involves introducing assumptions about the architecture and the access modes, etc. In effect we are mapping the abstract state space and equivalence relations onto the architecture in question. In particular, equivalences at the abstract level will typically correspond to states of the system that are indistinguishable under certain projections, for example, in certain pieces of memory. This is discussed in some detail by Rushby [74]. Certain issues remain however. For example in [21] it is shown that an access monitor that obeys the BLP rules can still leak information as a result of deadlocks. We will not discuss these issues further here as they would take us too far from the main thrust of the lectures.

5.5 Generalisations to Non-deterministic Systems

The fact that the original Goguen Meseguer formulation was restricted to deterministic systems is a serious limitation. Most systems of interest will manifest non-determinism, either because it is deliberately introduced for, say, cryptographic reasons, or because it arises as a result of abstracting internal details of the system. The first person to investigate how the Goguen and Meseguer formulation could be extended to deal with non-deterministic systems was McCullough [52,53]. He proposed a generalized version of non-interference but found that this, at least with respect to the definition of composition he proposed, failed to be compositional. Compositionality is widely regarded as a desirable feature of a security property: i.e., given two systems that each satisfy non-interference, then some suitable composition of them should also automatically satisfy it. Whether it is reasonable to assume that a *valid* definition of non-interference should be compositional is still a matter for debate. We will return to the question of compositionality later.

Problems of compositionality, non-determinism and the semantics of inputs and outputs have prompted a proliferation of variations on the original non-intereference formulation. These include: generalised non-interference [52], non-deducibility [88], non-inference [66], restrictiveness [53], forward correctability [39], non-deducibility on strategies [90], trace closure properties [45] and McLean's selective interleavings [59]. We will not go into the details of all of these as this would involve presenting a whole raft of models of computation

and would take us off the main thrust of these lectures. We will see a little later how several of these can be related when viewed in a process algebraic framework.

6 The Process Algebraic Approach

A central message that we want to convey in these lectures is that process algebras provide very effective frameworks in which to specify and analyse security properties, especially of distributed systems. There are a number of reasons for this. Process algebras are specifically designed for reasoning about systems interacting via the exchange of messages. They deal carefully with issues that we will see shortly are crucial to the study of security: non-determinism, composition, abstraction and the equivalence of systems. Added to these theoretical considerations is the fact that over the past decade or so the tool support for process algebras has improved dramatically: both theorem-proving and model-checking verification tools are now available.

6.1 Introduction to CSP and Process Algebra

We will base most of our discussion around the process algebra CSP. However, we will find that concepts from other algebras such as CCS and the pi-calculus are also useful, [61] and [63]. We start with a simple introduction to those aspects of CSP that we require.

Communicating Sequential Processes (CSP) was originally developed by Hoare to reason about concurrent systems interacting via hand-shake communications [36]. This was developed further by Roscoe and Brookes [7], and others. Timed CSP was originally proposed by Reed and Roscoe in [70] and further developed by Davies and Schneider [13]. For more up-to-date expositions, Roscoe [73] or, with more about Timed CSP, Schneider [82].

The interface of a process P is represented by its alphabet, denoted by αP, which is a collection of externally visible events (or actions) through which it can interact with the environment. Interaction takes the form of synchronisation on events. Thus, in order to interact, two processes simultaneously participate in some event, a say, and both move together onto the next state. This is an abstraction of the notion of a handshake communication and is well suited to many situations. Sometimes we really want to represent interaction in a more asynchronous fashion in which a process outputs some signal into the ether that might or might not be received by some other remote process at some later time. Typically this situation can be modelled within the CSP framework by introducing a medium with which the processes interact synchronously but which may delay or even lose messages, thus mimicking the effect of asynchronous communication between the end-points.

6.2 CSP Syntax

The basic syntactic constructs that will be of interest to us are as follows:

Prefix:

$$a \rightarrow P$$

Prefix choice:

$$a : A \rightarrow P(a)$$

Communication (input):

$$c?x \rightarrow P(x)$$

External choice:

$$P \,\Box\, Q$$

Non-deterministic (internal) choice

$$P \sqcap Q$$

Parallel composition over the alphabet A:

$$P \,||_A\, Q$$

Interleave

$$P \,|||\, Q$$

Hide events from the set A:

$$P \setminus A$$

Rename:

$$P[a/b]$$

P after trace tr:

$$P/tr$$

Let us explain these more fully:

Prefix The process term $a \rightarrow P$ can initially participate in the action a after which it behaves as the term P.

Prefix Choice This is similar to prefix except that we provide a set of events A from which the choice of prefix event must be drawn. Note that the continuation after the event a may be dependent on a.

Communication It is sometimes convenient to think in *value-passing* terms in which values can be communicated over channels rather than simply synchronisation on events. Channels will have types assigned to them. Let us denote the type of c by $\mathcal{T}(c)$. Thus the term $c?x \rightarrow P(x)$ can accept a value, $x : \mathcal{T}(c)$, over the channel c after which it behaves as the term P with appropriate internal variables bound to the value x. It is thus very similar to prefix choice but provides a syntactic sugar. In particular we can have channels with compund types.

External Choice $P \,\Box\, Q$ represents a choice of the two processes P and Q. If the initial events of P and Q are distinct the choice can be made by the environment, hence the name. Thus suppose that:

$$P := a \rightarrow P'$$

and

$$Q := b \rightarrow Q'$$

If the environment offers a to $P \,\Box\, Q$ then a will occur and $P \,\Box\, Q$ will thence behave like P'. Similarly if the environment offers b, then b will occur and $P \,\Box\, Q$ will subsequently behave like Q'. If the environment offers both a and b the choice will be made arbitrarily. Also if the intersection of the alphabets of P and Q is non-empty and the environment offers an event in this intersection then again the choice of continuation will be arbitrary.

Internal Choice Like $P \,\Box\, Q$ the term $P \sqcap Q$ represents a choice between P and Q but this time the choice is made internally and the environment has no influence over this choice. Consider P and Q above and suppose that the environment offers the event a to $P \sqcap Q$. It may be that the internal choice goes for the right-hand branch, i.e., $b \rightarrow Q'$ and so the event a is refused. As long as the environment insists on offering a there will be deadlock.

Parallel Composition In the alphabetised parallel composition of two processes $P \,||_A\, Q$, P and Q synchronise on all events from the set A, with $A \subseteq \alpha P \cap \alpha Q$. Thus, for any event from A both P and Q must simultaneously be prepared to participate for the event to occur. When such an event does occur both P and Q move together to their next states. Any events outside the set A can occur quite independently in either P or Q.

Interleave In the interleaved composition of P and Q, $P \,|||\, Q$, both processes can make progress entirely independently of the other. There is no synchronisation and hence no interaction between them. In fact we have:

$$P \,|||\, Q = P \,||_{\{\}}\, Q$$

i.e., interleave can be thought of as parallel composition over the empty alphabet.

Hiding Hiding over a set C simply removes events from C from view of the environment. Such hidden events are internalised: the environment cannot (directly) see or influence their occurrence. It is usual to refer to such internal, hidden events as τ events.

Renaming Renaming alters the identity of events. In general we can perform a renaming with respect to a relation on the events. More typically we will rename with respect to a one-to-one function. Sometimes also we will find it useful to rename several distinct names to a single name. We will refer to this last as *projection*.

Renaming is useful when writing CSP specifications as an alternative to parametrised specifications where, for example, the specification involves replicated components. In the context of security we will see that it is a rather useful abstraction operator that allows us to neatly capture a number of requirements.

After P/tr, where P is a process term and tr a trace, denotes the process P after it has performed the trace tr. For a non-deterministic system, P/tr will correspond to a set of states reachable by the trace tr. We will explain this more fully when we introduce the notion of a Labelled Transition System (LTS).

Constructs also exist for (mutual) recursive definitions of processes but these will not concern us.

6.3 Semantics

The semantics of CSP processes is traditionally presented in a denotational style, that is, a denotational space is constructed along with a mapping from the language to this space. In fact a number of denotation spaces, or models, have been constructed for CSP and which is appropriate for a given application depends on the kinds of property of interest and the kinds of distinctions between processes that are relevant. The simplest model is the traces model. A trace is simply a sequence of (visible) events representing a possible behaviour. In this model a process is mapped into the set of traces that correspond to its possible behaviours. Such a trace set must always contain the empty trace: for any process not having done anything must be a possible behaviour. Furthermore such a set must be prefix closed: if a certain behaviour is possible for a process S then any behaviour leading to that behaviour must also be possible:

$$\langle\rangle \in traces(S)$$

and

$$s \frown t \in traces(S) \Rightarrow s \in traces(S)$$

Consider a simple example:

$$P := a \rightarrow b \rightarrow c \rightarrow STOP$$

Where $STOP$ is the process that does nothing (so $traces(STOP) = \{\langle\rangle\}$). The traces of P are:

$$\{\langle\rangle, \langle a\rangle\langle a, b\rangle, \langle a, b, c\rangle\}$$

We can calculate the traces set for a process term P, denoted $traces(P)$, by structural induction on the syntax. Details can be found in [73] or [82]. Trace sets are thus much like the acceptance languages associated with finite automata. The traces model is well suited to reasoning about *safety* properties, that is, where we want to specify and check that certain undesirable behaviours are not possible. For deterministic processes the traces model is sufficiently rich to fully characterise processes. A process P is said to be deterministic if, for any possible trace, the set of events it is next prepared to participate in is well defined. Thus there is no trace tr after which in one run event a is offered by S whilst in another run which is identical as far as the visible trace tr is concerned, a is refused by S. Branching can occur, but it is controlled by the environment.

Where we are concerned with non-deterministic behaviour the trace model is not rich enough, indeed it is not rich enough to characterize when a system is deterministic. A simple example illustrates this. Consider the two processes P and Q (different to those earlier) defined by:

$$P = a \rightarrow (b \rightarrow STOP \,\Box\, c \rightarrow STOP)$$

$$Q = a \rightarrow (b \rightarrow STOP \sqcap c \rightarrow STOP)$$

These have the same trace sets:

$$\{\langle\rangle, \langle a\rangle, \langle a, b\rangle, \langle a, c\rangle\}$$

But a system trying to interact with them will typically be able to distinguish between them: in particular $Q \,||_c\, STOP$ could deadlock after a (if Q internally decides on the right-hand branch of the choice), whilst $P \,||_c\, STOP$ can't, it must continue to offer b. Thus, although the space of potential behaviours is identical a user's experience of interacting with these two processes could be completely different. If the user is set on Q doing b, say, but Q has chosen the RHS of the

$\sqcap$ and so is only prepared to offer c, then the user will probably end up kicking Q in frustration. This could not happen with P.

To reason about, and indeed distinguish, non-deterministic behaviours and liveness we need a richer model: we need to look at what a process may choose to refuse (or conversely, accept) as its behaviours unfold. To achieve this we now introduce the notion of a *refusal set*.

Suppose that the environment E initially offers the process P a set of events X, if $P \parallel E$ can deadlock immediately then X is said to be a *refusal* of P. Thus $\{a\}$ is a refusal of $a \rightarrow STOP \sqcap b \rightarrow STOP$. So is $\{b\}$ but $\{a, b\}$ isn't. Note that if X is a refusal for P then any subset of X will also be a refusal. The set of such refusal sets is denoted by $refusals(P)$.

This gives us information about what P may choose to refuse at the outset. We now extend this idea to give us refusal information as the behaviours of P unfold by introducing the *failures* of P.

A *failure* is a trace along with a refusal set. Thus:

$$failures(P) = \{(tr, X) \mid tr \in traces(P) \wedge X \in refusals(P/tr)\}$$

Consider a simple example:

$$P = a \rightarrow STOP \,\Box\, b \rightarrow STOP$$

$$Q = a \rightarrow STOP \sqcap b \rightarrow STOP$$

Thus

$$failures(P) = \{(\langle\rangle, \{\}), (\langle a\rangle, \{a, b\}), (\langle b\rangle, \{a, b\})\}$$

Whilst:

$$failures(Q) = \{(\langle\rangle, \{a\}), (\langle\rangle, \{b\}), (\langle a\rangle, \{a, b\}), (\langle b\rangle, \{a, b\})\}$$

And so we see that the failures sets for P and Q are distinct in the failures model. Here for brevity we have just given the maximal refusals. The sets should be filled out with the subset closures of the refusal sets. We find that the failures of Q include the elements:

$$(\langle\rangle, \{a\}) and (\langle\rangle, \{b\})$$

These are absent in the failures of P. This precisely reflects the fact that Q could, at the outset, decide to refuse a or to refuse b. P by contrast cannot refuse either.

Given the failures model we can state formally what it means for a process to be deterministic:

Definition S is deterministic iff:

$$\forall\, s \in traces(S) \wedge a \in \alpha S \neg\, (s \frown a \in traces(S) \wedge (s, \{a\}) \in failures(S))$$

In other words, we should not be able to find a trace after which some event might, in one run, be accepted, whilst in another, be refused by S, i.e., a behaviour after which the process can internally choose either to accept or refuse a.

An important point to note is that refusals, and hence failures, are defined on stable states. Stable states are ones in which no internal progress is possible. A stable state is one from which there are no outgoing τ transitions. The reason for this is that it is difficult to meaningfully assign refusals to unstable states as any internal transitions (invisible to the environment) may change these. Refusals are thought of as sets of events that, if offered to the process will never be accepted, at least before any external process has occurred. For unstable states such a definition is inappropriate.

Non-determinism can arise in three ways: explicitly from the internal choice operator, from hiding, or from ambiguous external events, e.g.:

$$(a \rightarrow b \rightarrow STOP \,\Box\, a \rightarrow c \rightarrow STOP)$$

Other semantic models exist, for example, the failures/divergence model designed to reason about internal thrashing, i.e., situations in which infinite sequences of internal events are possible without any external progress taking place. We will not need these for what follows.

Refinement Refinement, denoted $\sqsubseteq$ with a subscript to indicate in which model it is defined, is defined as set containment in the appropriate denotational model.

In the traces model:

$$P \sqsubseteq_T Q \Rightarrow traces(Q) \subseteq traces(P)$$

In the failures model:

$$P \sqsubseteq_F Q \Rightarrow failures(Q) \subseteq failures(P)$$

Refinement can be thought of as making processes more predictable. In the traces model we are simply asserting that a refinement cannot introduce new behaviours. In the failures model we are asserting this but also asserting that the refined process will never refuse events that were not refusable by the specification. This allows us to make assertions about the liveness or availability of the system to the environment (e.g., the users of the system). In particular, refinement should not introduce new ways for the system to deadlock. Refinement is monotonic with respect to the CSP operators, e.g.:

$$P \sqsubseteq_T P' \wedge Q \sqsubseteq_T Q' \Rightarrow P \,||\, Q \sqsubseteq_T P' \,||\, Q'$$

$$P \sqsubseteq_T P' \Rightarrow P \setminus C \sqsubseteq_T P' \setminus C$$

Some Useful Processes A few useful processes that we will need include:

$STOP$, that refuses to do anything:

$$traces(STOP) = \{\langle\rangle\}$$

RUN_A, always prepared to do any event drawn from A. Thus:

$$RUN_A = x \in A \rightarrow RUN_A$$

and

$$traces(RUN_A) = A^*$$

Where A^* is the set of all finite sequences with elements drawn from A.

The process $CHAOS_A$ may at any time choose to accept or reject any event from A. Thus:

$$CHAOS_A = STOP \sqcap ((x \in A \rightarrow CHAOS_A)$$

and

$$failures(CHAOS_A) = \{(tr, X) \mid tr \in A^* \wedge X \subseteq A\}$$

6.4 Labelled Transition Systems

In what follows it will often be useful to think in terms of an underlying labelled transition system (LTS). This is somewhat alien to the usual spirit of CSP, which is to concentrate on the external observations that can be performed on the system and abstract from all internal details. However, in the context of security, these internal details will often include the high-level user and so we have to be careful how we treat them. In particular we can't simply abstract them away.

In an LTS representation a process is thought of a collection of nodes, some of which are linked by labelled, directed arcs. A pair of nodes that are linked by a directed arc with label μ will represent the possibility of a transition between them associated with the event μ in the direction of the arc, where μ can be an internal τ event.

In general, a process term will correspond to a set of nodes of the underlying LTS. For example, P/tr will in general correspond to a set of (stable) states reachable by P executing the visible trace tr.

6.5 Acceptances and Ready Sets

The standard way to capture non-determinism in CSP is to use refusals. At first glance this may seem a little counter-intuitive and begs the question: why not think in terms of what the process will accept rather than what it will refuse? There are good technical reasons to use refusals for CSP rather than acceptances

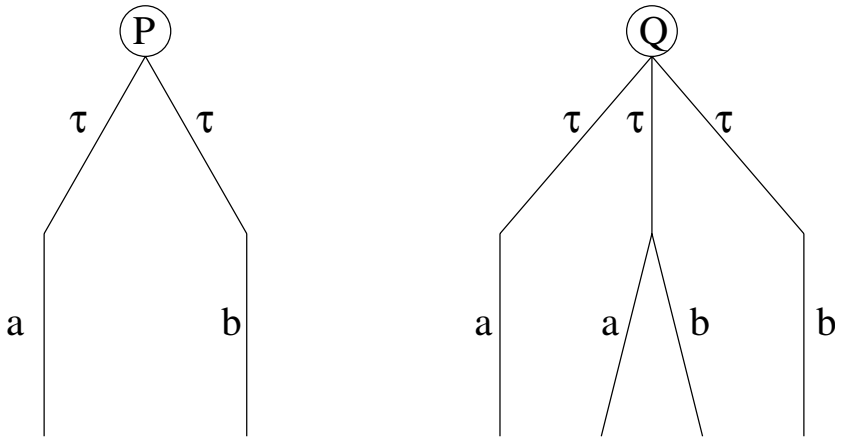

Fig. 1. Refusals vs ReadySets

as explained in [73]. For our purposes it is more intuitive to think in terms of what the system will accept. In particular this sits more comfortably with the bi-simulation approach to defining process equivalence that we will see shortly.

Acceptance sets are defined in a fashion dual to the defintion of refusal sets: X is an acceptance set of P if, when the environment offers the set X to P, an event in X will be accepted. Acceptance sets are defined to be superset closed, where closure is taken with respect to the universal alphabet Σ. The idea is that if an element of a set A will be accepted then if a larger set is offered then something from this larger set should again be accepted.

We will also need to define the idea of a *ready set*. This is defined in terms of the underlying LTS. Each node of the LTS has associated with it a ready set: the set of events that the system offers to the environment when in this state. It is thus the set of labels on the outgoing arcs from the node.

The distinction between acceptances and ready sets is that in the case of ready sets we do not take superset closure. Ready sets allow us to draw finer distinctions between processes than is possible with either acceptances or refusals. The subset or superset closure associated with the acceptances wipes out certain distinctions that are preserved when working purely with the ready sets. Figure 1 serves to illustrate this: the refusals of P and Q are identical, i.e., $\{\{a\},\{b\},\{\}\}$, whilst the ready sets of P are $\{\{a\},\{b\}\}$ and for Q they are $\{\{a\},\{b\}\},\{a,b\}\}$.

In the context of security the ready sets model seems the most appropriate. It is slightly more discriminating than either the failures or acceptances, i.e., it draws finer distinctions between processes and so allows hostile agents to draw more inferences about the state of the system. Thus, from a security point of view, it is a safer model to work with.

Usually refusals, acceptances and ready sets are defined only over stable states. This corresponds to assuming that internal events occur *eagerly* as soon

as they are available. However for our purposes we will not want to treat all internal events as eager as we will want to think of some of them as under the control of the High user. Consequently we regard ready sets as being defined on unstable states as well as stable. In some contexts we want to include τ's in the ready sets.

Where we need to consider a process term corresponding to sets of nodes of the LTS we need to consider the corresponding sets of ready sets. Let $Nodes(P)$ denote the set of nodes, both stable and unstable, corresponding to the term P. Then

$$ReadySets(P) = \{Ready(p) \mid p \in Nodes(P)\}$$

Often we will want to restrict the ready sets to some subset of the alphabet and we use a subscript to indicate this. Thus $Ready_L$ denotes the acceptance set restricted to L. $Ready_{L\tau}$ will denote the ready set restricted to $L \cup \{\tau\}$.

One final piece of notation we will need is that of *initials*. The initials of a process term P are the events P might be prepared to participate in next, ignoring non-determinism:

$$initials(P) = \{a \mid \langle a \rangle \in traces(P)\}$$

We have now set up all the necessary machinery to introduce various process algebraic definitions of non-interference.

7 CSP Formulations of Non-interference

We now present a formulation of non-interference in CSP, originally presented in [75], that stays close to the spirit of the original Goguen-Meseguer formulation but takes account of non-determinism and dispenses with any distinction between inputs and outputs.

$$\forall\, tr \in traces(S) \bullet ReadySets_L(S/tr) = ReadySets_L(S/(tr \upharpoonright L)) \tag{4}$$

$tr \upharpoonright L$ projects the trace tr down onto the event set L. It thus has much the same effect as $purge_{HI}(tr)$ except that here we do not draw any input/output distinction and so we in effect "purge" all high events. The ready sets projected onto Low's alphabet encode the non-determinism visible to Low. This formulation therefore seems to be the natural way to extend the Goguen-Meseguer formulation into a CSP framework in a way that accounts for non-determinism.

The same reference also gives a more symmetric formulation:

$$\forall\, tr, tr' \in traces(S) \bullet tr \approx tr' \Rightarrow ReadySets_L(S/tr) = ReadySets_L(S/tr')) \tag{5}$$

Where

$$tr \approx tr' \Leftrightarrow tr \upharpoonright L = tr' \upharpoonright L$$

These are closely related but differ in some subtle ways that we will discuss shortly. Where the system S is known to be deterministic we can get away with just using *initials* in place of *ReadySets*.

Both of these look a little inelegant as they involve quantification over traces and we would like to give an algebraic formulation. An obvious formulation to try is:

$$S \setminus H =_{ReadySets} (S \,||_H\, STOP_H) \setminus H \tag{6}$$

The idea here is that composing S in parallel with $STOP$ over the H alphabet has the effect of preventing the occurrence of any H events in the RHS of the equality. The RHS thus represents the system with all H events prevented. On the LHS of the equality S's interactions with H can proceed unhindered. At first glance this seems as though it should give us what we want: that the original system from Low's point of view is indistinguishable from the system with no High activity.

It actually turns out to be a weaker property for rather subtle reasons to do with the standard semantics of CSP. As remarked earlier, the standard failures semantics of CSP only applies to stable states, i.e., states from which no τ transitions are possible. The H's have been hidden and so are abstracted eagerly and so Equation 6 only constrains the ready sets on stable states. The quantification over all traces in Equation 4, on the other hand, ensures that this definition also constrains states from which High can perform an action. Clearly the latter is appropriate: such H actions could potentially influence the events available to Low.

One way to rectify this, proposed by Roscoe [72], is by using the $CHAOS_H$ process in place of $STOP_H$:

$$S \setminus H =_{ReadySets} (S \,||_H\, CHAOS_H) \setminus H \tag{7}$$

$CHAOS_H$ can choose to deadlock on an event from H at any time, including at any state from which S would accept an H event, and so this formulation, in effect, gives quantification over all traces.

Alternatively we can give a formulation that mirrors the symmetric formulation of Equation 5 with:

$$\forall\, U, U' \in Process_H \bullet (S \,||_H\, U) =_{ReadySets} (S \,||_H\, U') \tag{8}$$

This is actually a very intuitively appealing formulation as it is really saying that Low has no way to distinguish between users who are interacting with the system through the high-level interface.

These formulations raise a number of points that deserve further discussion. Firstly, CSP draws no distinction between inputs and outputs. We deal simply with events. The act of interaction between processes is thought of as a symmetric, co-operative activity. Both processes must agree to participate in an event

for it to occur. If either refuses the event then it will not occur. As soon as both agree, it can occur. There is no concept of one of the processes causing or controlling the occurrence of an event. We will discuss later some elaborations of the usual CSP framework to allow us to draw causal distinctions if and when appropriate.

Not drawing any input/output distinctions is a safe option: we won't make errors as a result of incorrectly categorising events. On the other hand, it may be overly strong in some cases and we may find ourselves rejecting systems that would seem to be secure.

A related point is that the formulations of Equations 4 and 7 imply that the purge of any trace is itself a valid trace. At first glance this would seem right: we are, in effect, saying that to Low the system always looks as though High has done nothing. However, we have to be careful. Consider a system in which an activity of Low triggers an alert message on High's screen and let us suppose that High cannot prevent or delay this message. Here Low will know when such an alert event will occur at the High level (because he caused it) but we would not usually regard this as representing a flow from High to Low. The point is that the occurrence of the alert event cannot be influenced by High and therefore cannot be used by him to signal to Low.

To illustrate this further consider the simple process:

$$S = l \rightarrow h \rightarrow l_2 \rightarrow STOP$$

Low can deduce that h has occurred when he sees l_2. The purge formulation would reject S as insecure. If h is a signal event (e.g., an alarm) over which High has no control we really should regard this process as secure.

If h is not refusable or delayable by High, i.e., High can only passively observe its occurrence, then, for the purge formulation, we should use the process S':

$$S' = l \rightarrow ((h \rightarrow l_2 \rightarrow STOP) \,\Box\, (l_2 \rightarrow STOP))$$

H refusing h does not deadlock S. Or, equivalently, we could use the original S with the symmetric formulation. Of course, if the occurrence of h can be influenced by High then we would be right to regard S as insecure. Thus, by suitable formulation of our models, we can capture distinctions between signal and refusable events. We will discuss later how, using the framework of testing equivalence, we can do this in a more systematic and general way.

Another difficulty with the formulations above is that they suffer from what is sometimes referred to as the *refinement paradox*. That is, we can define a process that satisfies them but for which a conventional refinement is insecure. Consider the process defined by:

$$S = (h_1 \rightarrow (l_1 \rightarrow STOP \sqcap l_2 \rightarrow STOP)) \,\Box\, (h_2 \rightarrow (l_1 \rightarrow STOP \sqcap l_2 \rightarrow STOP))$$

S satisfies our definitions of non-interference but is refined by:

$$S = (h_1 \rightarrow l_1 \rightarrow STOP) \,\Box\, (h_2 \rightarrow l_2 \rightarrow STOP)$$

This clearly leaks information. We will discuss this problem in more detail later. It is not unique to the formulations given above. The refinement problem was noted long ago by McLean [59] and, in a different setting, by Jacob [37]. Intuitively the problem arises from the fact that conventional refinements reduce non-determinism, i.e., make the system more predictable. This is entirely appropriate for safety-critical systems but can be disastrous for information security: making a system more predictable potentially allows hostile agents to make more precise inferences on the basis of limited information. A stream cipher with predictable output is not one in which we should have much confidence, to paraphrase Tom Stoppard in"Arcadia."

We should also comment on our use of ready sets rather than refusal sets. By using ready sets we are again making a worst-case assumption: that Low may be able to directly observe exactly which events are on offer. Whether this is appropriate really depends on the system in question. If, for example, the system has a display of lights that indicate at each point which events it will accept then the ready sets formulation is appropriate.

For other systems it may be appropriate to think in terms of Low performing experiments by offering an event or set of events to the system and seeing if anything is accepted. If it is accepted then the system moves on to the next state and, having moved on, Low can obtain no further information about what other events might have been accepted in the previous state. Thus typically the information available to Low is much less in this model, hence his inability to make such fine distinctions between processes. The subset closure of the refusal sets (or alternatively the superset closure of the acceptance sets) encodes this: certain distinctions that could be made working only with ready sets are masked by the closure. There is a sort of *Heisenberg uncertainty principle* at play here: observing certain parameters of the system tends to disturb it and so disrupt the observation of other parameters.

On the other hand, if the environment can back-track to the initial state after an event has been accepted and continue testing then it will be able to identify the exact ready set for that state.

8 Abstraction

Process algebras provide elegant ways of abstracting away details and encoding different views of a system. The most obvious way of abstracting is to hide a set of events, C say:

$$P \setminus C$$

However, as previously remarked, the standard CSP semantics assumes that hidden, internal events occur eagerly. As a result this form of abstraction works fine for signal events, e.g., messages to the screen. In some situations, notably security, we want to abstract certain events but not force them to be eager, they may be under the control of High and so refusable or delayable. Here it is appropriate to use *lazy* abstraction:

$$Lazy_C(S) := (S \,||_C\, CHAOS_C) \setminus C$$

Another, closely related, way to abstract events but without hiding them is to camouflage them:

$$Cmflg_C(S) := (S \,|||\, RUN_C)$$

Here the environment can still see the events from C but cannot tell if they are associated with S or are just spurious C events from RUN_C.

Variants of these are possible, reflecting certain subtleties of the semantics of CSP. A full discussion can be found in chapter 12 of [73].

Renaming can also be a useful abstraction operator. We can use it in two principal ways: permuting a set of event names or renaming a set of events to a single name. We can think of the latter as a sort of projection: Low knows an event from the set in question has occurred but not which.

There are two main ways of using the permutation of a set of events: applying the same permutation throughout an execution or using a fresh permutation at each step of the execution. The latter really gives the same effect as the projection. The former is potentially more interesting as it allows for Low to correlate events. We will see later how these are useful for coding anonymity and encrypted channels.

Mixes of these are also possible and, using certain coding tricks, it is, to some extent at least, possible in CSP to allow for the abstractions to vary dynamically. This kind of modelling works rather better in a process algebra like Milner's π-calculus, which is specifically designed to handle dynamic networks of processes.

9 Unwinding the CSP Formulation

The formulations above all involved quantifications over traces or processes. We would prefer a more convenient definition for verification. In [27] the idea of *unwinding* is introduced. The idea is to replace the original definitions by conditions on individual transitions. Assuming that the class of possible transitions is essentially finite this should give a more tractable formulation set to check. [75] gives such conditions for the formulation of Equations 4 and 5. Let us consider the latter, more symmetric version, as it will prove more suitable for what follows.

Unwinding (Symmetric Version)

– Rule 1:

$$\forall\, Y_i, Y_j \in States(S) \bullet Y_i \approx Y_j \Rightarrow ReadySets_L(Y_i) = ReadySets_L(Y_j)$$

– Rule 2:

$$\forall\, Y_i, Y_j \in States(S) \bullet Y_i \approx Y_j \Rightarrow \forall\, e \in Initials(Y_i), \exists\, tr \in traces(Y_j) \bullet \\ e \upharpoonright L = tr \upharpoonright L \wedge Y_i/e \approx Y_j/tr$$

Note that we have introduced an (abstract) state space and an equivalence relation $\approx$ on it. Rule 1 has the effect of ensuring that equivalent states give rise to the same *ready sets* when restricted to Low's interface. Rule 2 ensures that $\approx$ is exactly that induced by the purge of H events. That is, states reached by traces with the same purge are regarded as equivalent. A straightforward induction argument on the length of traces establishes this correspondence. With this we can readily proof that:

$$Rule1 \wedge Rule2 \Rightarrow NI_{CSP}$$

The implication in the other direction is more delicate however, i.e., to show that the rules are necessary as well as sufficient. [75] gives a rather clumsy proof of this by arguing that any process that satisfies the non-interference property can be implemented as a machine that satisfies the rules.

In fact a far more elegant and insightful proof is possible when one observes that the unwinding rules actually bear a striking resemblance to the notion of bi-simulation, allowing us to borrow some results from the process algebra literature. First we need to introduce a few ideas from the operation style of process semantics.

10 Operational Semantics

An operational semantics is typically presented in the form of transition rules. Thus $P \xrightarrow{\mu} P'$ indicates that the process term P can make a transition labelled μ to the process term P'. The semantics can then be presented in terms of transition rules corresponding to the various syntactic operators. Simple examples, with empty antecedents include:

$$(a \rightarrow P) \xrightarrow{a} P$$

or

$$(a \rightarrow P \,\Box\, b \rightarrow Q) \xrightarrow{a} P$$

and

$$(a \rightarrow P \,\Box\, b \rightarrow Q) \xrightarrow{b} Q$$

The first simply asserts that a process term $a \rightarrow P$ can first perform the action a and then behaves like the process P. The latter two simply assert that the term $a \rightarrow P \,\Box\, b \rightarrow Q$ can perform an a in which case it subsequently behaves as P or a b in which case it then behaves as Q.

Notice that there are two different kinds of arrows here: inside the brackets the arrows are CSP prefix arrows. The longer, labelled arrows denote labelled transitions between process terms and are not part of the CSP notation but are part of the notation needed to present the operational semantics.

10.1 Strong Bi-simulation

Given a denotational semantics, equivalence is simply that induced by the mapping into the denotational space: two processes are deemed equal if they map to the same object in the space. In an operational style of semantics this device is not available to us and we need alternative ways to characterise the equality of terms of the algebra. The semantics is given purely in terms of what actions a process term can perform, hence we need to define equality in these terms. The usual approach is to use the notion of bi-simulation; intuitively that two terms are equal if each is able to match the others actions.

More formally: processes P and Q are strongly bi-similar if $\exists$ a symmetric relation R on the space of process terms such that:

$$PRQ \wedge P \xrightarrow{\mu} P' \Rightarrow \exists\, Q' \bullet (Q \xrightarrow{\mu} Q' \wedge P'RQ')$$

Here μ is any transition label drawn from the set of actions of P, including τ's.

In other words, assuming that they start off in equivalent states, if one can perform an action μ then the other must also be able to perform the μ action and, furthermore, after these actions they can end up in equivalent states. The latter clause ensures that we can unfold this condition to ensure that they can continue to simulate each other indefinitely. Thus each can mimic the other. Notice that they will not necessarily end up in equivalent states. The condition only requires that it is *possible* for them to reach equivalent states. Thus in general there may be other transitions from Q also labelled μ but that end up in states that are not related to P'.

10.2 Weak Bi-simulation

Strong bi-simulation insists that the processes stay in step on all actions including the hidden τ actions. Given that the τ actions are not observable by the environment this tends to be too strong a criterion. We can have processes that are observationally indistinguishable and yet are not strongly bi-similar. Strong bi-similarity is thus drawing distinctions dependent on internal, unobservable differences between implementations. This is not really in the spirit of the abstract, implementation-independent, viewpoint of process algebra. The natural weakening of this condition is to relax the requirement that the processes stay in step on the τ's.

Weak bi-simlarity is defined in a similar way except that for visible events we allow for arbitrary interleavings of τ events with the matching event to reach equivalent states. A *tau* transition in one process can be matched by an arbitrary

sequence of *tau*'s, possibly of length zero. Where μ is a visible event we take $P \stackrel{\hat{\mu}}{\Longrightarrow} P'$ to indicate that P can transition to P' with a visible event μ interleaved with arbitrary τ's, that is, an element of $\hat{\mu} \in \tau^*.\,\mu\,.\tau^*$, or where μ is a *tau* event $\hat{\tau}$ is taken to denote a sequence τ^n of *tau*'s, where n could equal 0. We can now define the weak-bisimulation relation on process terms.

Two process terms P and Q are weakly bi-similar if $\exists$ a symmetric relation R such that:

$$PRQ \wedge P \stackrel{\mu}{\longrightarrow} P' \Rightarrow \exists\, Q' \bullet Q \stackrel{\hat{\mu}}{\Longrightarrow} Q' \wedge P'RQ'$$

Thus, if one process performs a visible event, we require that the other process be able to match this but allow for interleaving with τ events to reach equivalent states. If one process performs a hidden τ event, we require that the other be able to perform some sequence of τ's, possibly none, to arrive at an equivalent state.

10.3 Unwinding and Bi-simulation

There is a clear similarity between unwinding rules and (weak) bi-simulation. Both introduce an equivalence on states and both require the possibility that matching transitions from equivalent states lead to equivalent states.

The point about this analogy is that it potentially gives us a way of viewing unwinding results in a process algebraic context. If we can frame the unwinding rules in a bi-simulation style it also potentially gives us a more elegant and insightful proof of completeness. Furthermore it could give us access to established results and algorithms for establishing bi-simulation relations and for verifying equivalences of processes.

There are, however, some differences: bi-simulation does not have an immediate analogy of rule 1 of the unwinding conditions, i.e., the matching of the ready sets of equivalent states. In fact we will see how this follows from the requirement for a bi-simulation that the processes be able to match visible transitions.

Another difference is that the unwinding rules work with ready sets. It is well known that weak bi-similarity is stronger than failures equivalence and, as discussed earlier, ready sets equivalence is stronger than failures.

Bi-simulation differs from failures equivalence in that it makes distinctions according to where non-determinism is resolved, i.e., at what point internal choices are made. Consider the following simple example that illustrates this:

$$P = (a \rightarrow b \rightarrow STOP) \,\Box\, (a \rightarrow c \rightarrow STOP)$$

$$Q = a \rightarrow (b \rightarrow STOP \sqcap c \rightarrow STOP)$$

In the first the branching occurs before the visible a event whilst in the second it occurs after the a. They can easily be shown to be failures and testing equivalent. These two processes are actually also ready sets equivalent as it

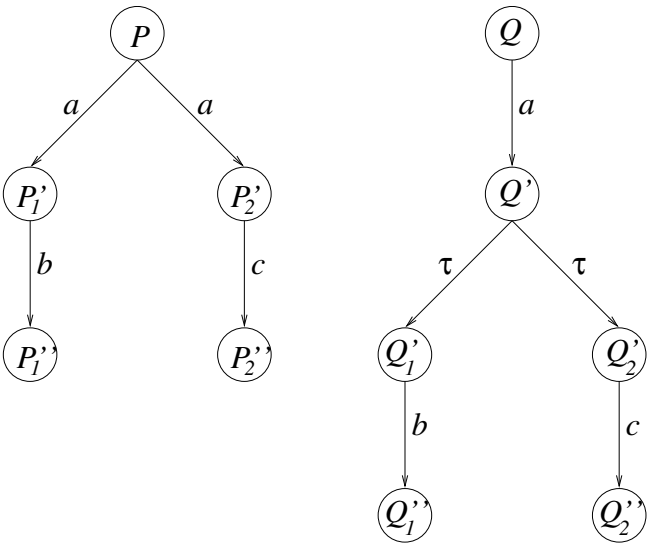

Fig. 2. Processes P and Q

happens. Any process interacting with them but with no visibility of the internal mechanisms would be unable to distinguish them. It can easily be shown, however, that no bi-simulation relation between them can be constructed.

At this point we could adopt the position that because bi-simulation gives a stronger and therefore, arguably, a safer definition of secrecy and we could simply adopt a bi-simulation definition of security. This would give a formulation of secrecy along the lines of those proposed by Gorrieri et al, [19] and described in the chapter by R.Focardi, R.Gorrieri of this volume.

In fact bi-simulation-style definitions of process equivalences have been formulated that are precisely equivalent to the failures equivalence, and indeed which, with a minor change, can also capture ready sets equivalence. We can therefore utilise these to produce a bi-simulation-style formulation that is exactly equivalent to our unwinding rules.

In the theory of automata it has long been known that a non-deterministic finite state automaton can be converted into one that is deterministic and equivalent with respect to the acceptance language model. The construction involves lifting the model to the power set of the state space and working with sets of nodes and transitions rather than single nodes. This is not quite what we want as the accepting language model ignores non-determinism. It does, however, give an indication of how to proceed.

Gardiner introduces a formulation of simulation that is equivalent to failures refinement of CSP [24]. In a more recent paper this is extended to provide a construction of bi-simulation that corresponds to failures equivalence and indeed, by a slight tweak of the construction, also provides a construction that corre-

sponds to the ready sets equivalence [25]. The details are quite elaborate and beyond the scope of these lectures. We will try to give the intuition and provide a construction for processes that satisfy our definitions of non-interference.

10.4 Power Bi-simulation

The problem with the example earlier was the insistence on relating individual states. If instead we think in terms of sets of states we can abstract away from the details of where exactly internal choices are made. The key idea then is to construct a bi-simulation relation on the power set of the state space: $\mathbb{P}\, States$.

For a process S, set of states ϕ and visible event a define the *spray* of ϕ, $\langle [\phi] \rangle_a$, by:

$$\langle [\phi] \rangle_a = \{\phi' \in States(\phi) \mid \exists\, \tilde{a} \in \tau^*.a.\tau^* \bullet \phi \stackrel{\tilde{a}}{\Longrightarrow} \phi'\} \tag{9}$$

That is, $\langle [\phi] \rangle_a$ gives all states reachable starting from a state in ϕ with a visible a, possibly interleaved with τ's.

A relation $\approx$ on $\mathbb{P}(S) \times \mathbb{P}(S)$ is a power bi-simulation relation for an LTS of S if:

$$S_1 \approx S_2 \Rightarrow \langle [S_1] \rangle_a \approx \langle [S_2] \rangle_a \tag{10}$$

This in itself is not so useful: trivial solutions for $\approx$ exist. In particular the top relation that relates everything to everything satisfies this so the usual device of taking the largest relation satisfying the property is singularly uninteresting in this case. To get interesting solutions we need to impose additional constraints on $\approx$.

The intuition behind Gardiner's construction is to find the largest symmetric equivalence relation $\approx$ satisfying 10 and such that for any pair of related elements (sets of states) ϕ, ϕ' and any subset $s \in \phi$ we can find a subset s' of ϕ' such that:

$$initials(s) \subseteq initials(s')$$

Thus, refering to figure 2 again, the node Q' must be matched up with the pair of nodes $\{P_1', P_2'\}$, whilst P_1' can be matched with just Q'.

This gives a characterisation of equivalence that is exactly as discriminating as failures equivalence. Alternatively we can obtain ready sets equivalence using the slightly stronger requirement of $=$ of initials rather than $\subseteq$. It also turns out that for the ready sets model the bound has to act on stable states rather than arbitrary states as was the case for the failures equivalence. Gardiner shows that such constructions give bi-simulation characterisations of equivalence that correspond exactly to traces, failures or ready sets (depending on the choice of bound).

The construction is related to the normalisation performed by FDR, the model-checker for CSP. FDR stands for Failures, Divergences and Refinement. The tool, marketed by Formal Systems Europe Ltd performs refinement check between pairs of CSP specifications [17]. The tool creates an LTS representation of a CSP specification. During normalisation it converts the original LTS to a Generalised LTS (GLTS), whose nodes correspond to sets of nodes of the original with annotations (minimal acceptances) to carry the non-determinism information.

10.5 Loose Bi-simulation

We now introduce a construction for the power bi-simulation relation for systems satisfying the definitions of non-interference given by Equation 5, or equivalently Equation 8.

Once again it is helpful to think in terms of an underlying Labelled State Transition System (LTS) with internal transitions exposed, i.e., with labels drawn from $H \cup L \cup \{\tau\}$.

Define the relation $\sim_S$ on states of S by:

$$\sim_S = \{(S/tr, S/tr') \mid tr, tr' \in traces(S) \wedge tr \upharpoonright L_\tau = tr' \upharpoonright L_\tau\} \tag{11}$$

where $L_\tau = L \cup \{\tau\}$

In effect this is the equivalence relation induced by *purging* the $H's$ but keeping the L's and τ's visible.

We now define a variant of weak bi-simulation that we call loose bi-simulation:

P and Q are loosely bi-similar if $\exists$ a symmetric relation $\sim_S$ such that:

$$P \sim_S Q \wedge P \xrightarrow{\mu} P' \Rightarrow \exists\, Q' \bullet Q \stackrel{\overline{\mu}}{\Longrightarrow} Q' \wedge P' \sim_S Q' \tag{12}$$

Where, for $\mu \in L \cup \{\tau\}$, $\overline{\mu}$ denotes a sequence of events in $H^*.\mu.H^*$, the set of all finite sequences of H actions interleaved with a single μ. For $\mu \in H$ we take $\overline{\mu}$ to denote a sequence of H events, possibly of length zero. It is thus analogous to weak bi-simulation but with τ and L events kept visible whilst the $H's$ are hidden. Thus the $H's$ playing the role of τ's.

10.6 Power Bi-simulation for Non-interference

We now define a power bi-simulation relation on $\mathbb{P}\, States(S) \times \mathbb{P}\, States(S)$:

$$\approx_S := \{(\{S \mid S \stackrel{tr}{\Longrightarrow} S'\}, \{S \mid S \stackrel{tr'}{\Longrightarrow} S'\}) \mid tr, tr' \in traces(S) \wedge tr \upharpoonright L = tr' \upharpoonright L\} \tag{13}$$

i.e., abstracting the τ's and $H's$. Note that this relation acts on the power set of the state space and so relates sets of states to sets of states.

Lemma 1. *If S satisfies the loose bi-simulation w.r.t. $\sim_S$ then it satisfies a power bi-simulation w.r.t. $\approx_S$.*

The proof follows straightforwardly from the definitions.

Lemma 2. *If $S_1 \approx_S S_2$ then for all $s_1 \in S_1$ there exists $s_2 \in S_2$ such that $s_1 \sim_S s_2$, and vice versa.*

Again the proof is a straightforward, if slightly messy, induction argument.

Lemma 3.

$$s_1 \sim_S s_2 \Rightarrow Ready_L(s_1) = Ready_L(s_2)$$

This follows immediately from the definition of loose bi-simulation: states related by $\sim_S$ must be able to match transitions and so they must match ready sets.

Lemmata 2 and 3 together immediately imply:

Lemma 4.

$$S_1 \approx_S S_2 \Rightarrow ReadySets_L(S_1) = ReadySets_L(S_2)$$

Figure 3 may help illustrate what is going on. Here p_1 and p_2 are related by $\sim$ and so have matching ready sets on $L \cup \{\tau\}$. p_1 is an element of the set of nodes Ψ_1 and p_2 is an element of Ψ_2 with $\Psi_1 \approx \Psi_2$. The diagram shows both p_1 and p_2 making transitions labelled *mu* to p_1' and p_2' respectively and, if the system satisfies loose bi-simulation with respect to $\sim$, we have $p_1' \sim p_2'$. Furthermore p_1' and p_2' will be elements of Ψ_1' and Ψ_2' respectively, where $\Psi_1' = \langle [\Phi_1] \rangle_\mu$ and $\Psi_2' = \langle [\Phi_2] \rangle_\mu$, with $\Psi_1' \approx \Psi_2'$. Finally note that $Ready_{L\tau}(p_1') = Ready_{L\tau}(p_2')$.

We have thus established that S satisfies loose bi-simulation w.r.t. $\sim_S$ implies S satisfies power bi-simulation w.r.t. $\approx_S$ and the ready sets bound which in turn is equivalent to S satisfying the non-interference property of Equation 5.

In fact we see that Lemma 4 is really just a restatement of unwinding rule 1 and Lemma 1 is a restatement of rule 2.

We have thus constructed a power bi-simulation formulation equivalent to the original non-interference property. Completeness, i.e., that this formulation is both necessary and sufficient, now follows from Gardiner's results that show that the largest equivalence over the LTS of the CSP language that satisfies the equality of initials bound on stable states gives the same equality of terms as the ready sets model.

It is worth examining this construction more carefully. Although originally designed as a rather formal exercise in recasting certain results in a process algebraic framework and to give a more elegant and insightful proof of the completeness of an unwinding result, it actually has some useful spin-offs. Firstly notice that we have, in effect, divided the hidden events into $H's$ and τ's and are thinking of these as the potential source of any non-determinism that may be observable by Low.

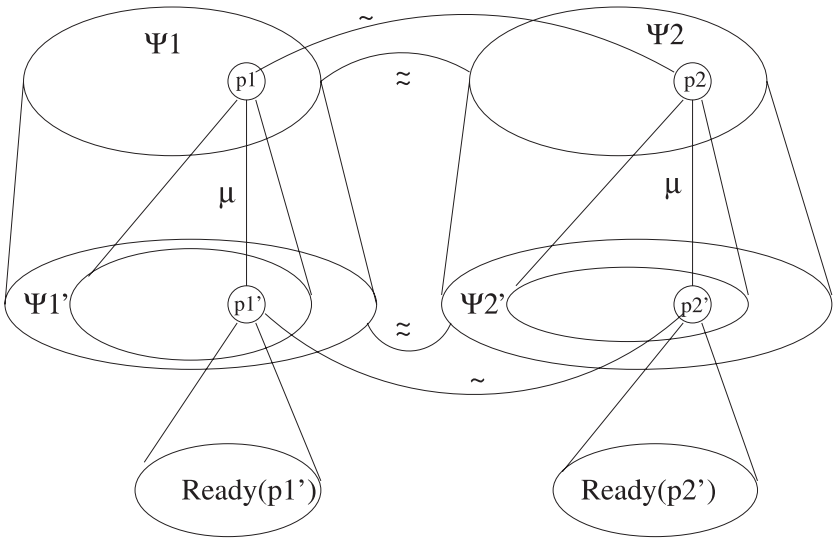

Fig. 3. Loose bi-simulation

Examining the loose bi-simulation property we see that it is asserting that the occurrence of $H's$ cannot influence the availability of τ's. We can think of the τ's as representing internal, system sources of entropy giving rise to the non-determinism visible to L, for example, due to a scheduler or to the output of a stream cipher. For a system satisfying non-interference in this framework the τ's are the sole source of any non-determinism visible to Low. Different τ behaviours can resolve the non-determinism in different ways but differing H behaviours cannot. The fact that $H's$ do not interfere with the τ's ensures that there cannot be an indirect channel from High to Low.

It may be appropriate to disambiguate the τ's. Usually, in CCS for example, τ's are invisible and so can reasonably be regarded as indistinguishable. In our construction above, however, we have, at some stages of the analysis at least, exposed the τ's. Suitable disambiguation of the τ's could allow a closer correspondence to be established between differing τ behaviours and differing non-deterministic behaviours manifest to Low. In effect the τ's are doing the book keeping that ensures that the non-determinism manifest to Low in states equivalent under $\approx$ is consistent.

A pleasing feature of the power-bi-simulation approach is that it connects the denotational, operational and testing formulations of process equivalence, and hence of non-interference. In [33] it is argued that bridging different presentations of semantics should be a major goal of theoretical computer science.

Unwinding results can be interpreted as special forms of bi-simulation and so existing algorithms for constructing bi-simulations may be useful for establishing unwinding rules. Pinsky provides a construction for equivalence classes

in a somewhat different, predicate based framework [68]. These results can be mapped across to Gardiner's predicate transformer approach.

10.7 Composability

Reasoning about the composability of various formulations of non-interference with respect to various forms of composition becomes much easier in a process algebraic framework. Furthermore, having a bisimulation formulation appears to help significantly. To illustrate we present a proof of composition for processes satisfying our loose bisimulation formulation with respect to parallel composition. Parallel composition seems to be one of the more interesting operators to consider, interleaving tends to be rather trivial for example. Note that for this to make sense, we assume that High channels are linked to High channels and Low to Low. Without this assumption non-interference would be trivially violated.

Suppose the S and T both satisfy loose bi-simulation w.r.t $\sim_H$ induced by $purge_H$.

Thus

$$S_1 \sim_H S_2 \wedge S_1 \xrightarrow{\mu} S_1' \Rightarrow \exists\, S_2' \bullet S_2 \stackrel{\overline{\mu}}{\Longrightarrow} S_2' \wedge S_1' \sim_H S_2'$$

Similarly for T.

Now consider $S \,||_\Sigma\, T$ where $\Sigma = H \cup L \cup \{\tau\}$. The Σ subscript on $||$ will be taken as implicit from now on.

and define $\sim_H^*$ on $S \,||\, T$ by:

$$S_1 \,||\, T_1 \sim_H^* S_2 \,||\, T_2 \Leftrightarrow S_1 \sim_H S_2 \wedge T_1 \sim_H T_2$$

Lemma 5.

$$S_1 \,||\, T_1 \sim_H^* S_2 \,||\, T_2 \Leftrightarrow \exists\, s_1, s_2 \in traces(S) \bullet S_1 = S/s_1 \wedge S_2 = S/s_2$$

where:

$$purge_H(s_1) = purge_H(s_2)$$

And similarly for T.

That is the $\sim_H^*$ equivalence is the same as would have been induced by purging the traces of the composed process.

Now we need to show:

$$S_1 \,||\, T_1 \sim_H^* S_2 \,||\, T_2 \wedge S_1 \,||\, T_1 \xrightarrow{\mu} S_1' \,||\, T_1'$$

implies $\exists\, S_2' \,||\, T_2'$ such that

$$S_2 \,||\, T_2 \stackrel{\overline{\mu}}{\Longrightarrow} S_2' \,||\, T_2' \wedge S_1' \,||\, T_1' \sim_H^* S_2' \,||\, T_2'$$

Now

$$S_1 \,||\, T_1 \xrightarrow{\mu} S_1' \,||\, T_1' \Rightarrow S_1 \xrightarrow{\mu} S_1' \wedge T_1 \xrightarrow{\mu} T_1'$$

so by Loose bisimulation of S and T we must have S_2' and T_2' such that:

$$S_2 \stackrel{\overline{\mu}}{\Longrightarrow} S_2' \wedge T_2 \stackrel{\overline{\mu}}{\Longrightarrow} T_2'$$

so

$$S_2 \,||\, T_2 \stackrel{\overline{\mu}}{\Longrightarrow} S_2' \,||\, T_2'$$

but we also have:

$$S_1' \sim_H S_2' \wedge T_1' \sim_H T_2'$$

so indeed:

$$S_1' \,||\, T_1' \sim_H^* S_2' \,||\, T_2'$$

as required.

The structure of this proof follows largely from the nature of bi-simulation. The only tricky part is in showing that the equivalence used in defining loose bi-simulation for the composition of the processes is the *same* as that used for the processes separately. In this case we had to show that the equivalence $\sim_H^*$ defined above is the same as had we defined it directly from the purge of traces of the composed process. In other words, the key step is to show that the equivalence used in defining the loose bi-simulation *lifts* through composition in the appropriate way. In this case the proof is straightforward but for the subtler forms of non-interference we will meet later it may not necessarily be so straightforward (or even necessarily true). Schneider discusses a variety of compositionality results with respect to a number of styles of composition results in the context of a framework based on testing equivalence [84].

10.8 The Roscoe-Woodcock-Wulf Approach

The fact that the formulations of non-interference given above fail to be preserved under the usual refinement of CSP, or indeed most other styles of refinement, is troubling. Various responses are possible to this: we could conclude that such a formulation is flawed. We could conclude that this is just a fact of life and security is not a property preserved under refinement. This would be a distinctly depressing conclusion as it would deny us the possibility of step-wise development and verification. Another is to accept that conventional refinements will not preserve security and if we want security to be automatically preserved we will need a new formulation of refinement. Yet another response is to conclude that maybe it is unreasonable to expect security to be preserved automatically and that we have no choice but to do further verification as we move down towards the implementation during the design process: i.e., to generate and discharge proof obligations at each refinement step.

We will discuss the latter reactions a little later. First we introduce an alternative CSP formulation of non-interference that is preserved under CSP refinement. The approach is due to Roscoe, Woodcock and Wulf [71].

The key idea here is to assert that a suitable abstraction of the system that represents Low's view be deterministic. In our earlier formulations we sought

ways to allow some non-determinism at Low's level but denying High any way to influence the way this is resolved. In the this approach there simply isn't any non-determinism in Low's viewpoint. Such a system is fully under the control of the environment (as far as it's externally observable behaviour is concerned) and so it cannot be the source of any information or entropy. There may be entropy being created and sloshing around internally but it never gets to leak outside.

Definition. A system S satisfies RWW non-interference, denoted $NI_{RWW}\,S$, iff $Abstract_H(S)$ is deterministic.

Where $Abstract_H$ denotes an appropriate abstraction of the H events giving Low's view of the system.

As remarked earlier, non-determinism is never increased by CSP refinement. Consequently, any system that is deterministic to start will remain deterministic under refinement. Thus any refinement of a system that satisfies the RWW property of non-interference will necessarily also satisfy it. A further advantage is that it can be automatically checked, indeed FRD has a button for this.

This makes the approach very attractive. It is stronger than the previous formulations we have discussed: any system satisfying NI_{RWW} will also satisfy all of the formulations given previously. It also side-steps many of the rather delicate issues that dog other formulations: in particular, what is the right notion of equivalence to use? How can we be sure that there is not some subtle way for High to influence non-determinism that is missed by our models. The latter problem is another manifestation of the refinement problem.

For a system whose security can be formulated as NI_{RWW} it is clearly sensible to use this formulation for the above reasons. The drawback is that it would appear that there is a large class of systems of interest for which this formulation is not appropriate, i.e., for which some residual non-determinism at Low is unavoidable. The classic example is the encrypted channel that we will discuss in greater detail later. Suppose that we consider a stream cipher (essentially a one-time-pad). This is a source of pure entropy and it does leak to the outside. However, a well designed cipher box properly incorporated in a secure architecture should still be secure. In essence the entropy generated by the box cannot be influenced or predicted by High.

Such an example cannot, in any obvious way at least, be formulated in the NI_{RWW} style. Actually it is not trivial to formulate in other weaker formulations either and we will return to this point later.

Another appealing feature is that the definition of determinism is fairly uncontroversial and coincides for most process algebras.

It is possible to combine this approach with the loose bi-simulation approach described earlier: consider an abstraction of the system with the H's abstracted but the τ's kept visible and indeed disambiguated. We now require that this abstraction be deterministic. This is stronger than the loose bi-simulation that we introduced earlier and indeed implies it. The *real* system again has the τ's abstracted and so can manifest non-determinism at Low's interface. Again we

know that any non-determinism visible to Low will be entirely due to differing τ behaviours and furthermore we know that High's activity cannot influence the τ's. We thus get the best of both worlds: a simple and easily verifiable and refinable property that still allows for some non-determinism at Low's level.

For this to work we certainly need to disambiguate the τ's as we would otherwise have non-determinism due to ambiguous branching on τ's. There seems no reason not to do this and indeed the identification of all internal events with a single event could itself be regarded as a modelling device, an entirely sensible one for hidden events.

We also need to avoid non-determinism arising from ambiguous Low events. In fact we can always transform a system with ambiguous visible events to one in which such branching is replaced by τ branching.

10.9 The Jacob Security Ordering

A drastically different approach to formalising security was taken by Jacob, [37,38]. Jacob suggests that there is, in fact, no way of saying categorically whether or not a system is secure. In this approach security is considered to be a purely relative property: at best all we can do is establish that one system is at least as secure as another. To this end he introduces a security ordering based on the notion of *infer* functions. Given a Low level projection of a trace the infer function returns the set of system behaviours consistent with this observation. Formally:

Given a trace $tr \in trace(S)$

$$infer_S(tr) := \{s \mid s \in traces(S) \wedge s \upharpoonright L = tr \upharpoonright L\} \tag{14}$$

The larger the cardinality of the set returned by *infer* the greater the uncertainty about the system behaviour associated with that observation. The idea now is, for a pair of systems P and Q, to compare the size of the *infer* function for all possible Low observations. We will consider P to be at least as secure as Q if:

$$\forall\, tr \in traces(Q) \bullet infer_Q(tr) \subseteq infer_P(tr) \tag{15}$$

In other words, for any given Low observation, the set resulting from the infer function applied to P is always a superset of the corresponding infer function for Q. Thus the uncertainty resulting from observations of P is always greater than would be the case for Q.

The idea is very interesting, but it has failed to catch on, presumably because people have tended to feel uncomfortable with not being able to characterise a given system as either secure or insecure. Given that security is never absolute this is really not a valid objection. To return to the safe analogy: it is not meaningful to rate a safe as "secure," but it may be meaningful to claim that one safe is more secure than another, i.e., for all known forms of attack it will

withstand for a longer period. The time may be ripe to take a fresh look at this approach.

10.10 Testing Equivalence

Another way of characterising process equivalence is the notion of testing. The idea is highly intuitive: if no experiment that the environment can perform on a pair of systems P and Q can distinguish between them then they are deemed to be equivalent. This is really very compelling in the context of security as we can think of these experiments as representing the efforts by Low to infer something about High's activity.

Schneider shows that several of the existing formulations of non-interference style properties can be cast rather naturally as flavours of testing equivalence [84]. Indeed it seems that in several cases the authors of these were in effect independently reinventing certain flavours of testing equivalence. We will show, for example, how *non-deducibility* and *non-deducibility on strategies* emerge naturally from thinking in terms of testing equivalences. First we give a formal definition of testing equivalence.

A test is a process T with a distinguished *success* event ω. We allow the system P to run in parallel with T and observe whether the resulting process can reach a state in which the ω event is possible. We can then characterise processes in terms of the set of tests they pass, which provides us with an alternative way to assign meaning to process terms. In particular, we regard two processes that, for all possible tests, pass the same set of tests as equivalent. We have to make precise what we mean by *all possible tests*. Typically this will simply be the space of tests that can be expressed in the process algebra. In some circumstances we may want to constrain the space of possible tests to reflect limitations on the capabilities of the environment (or hostile agents in the case of security).

To formally define the notion of a test we need to introduce a special *SUCCESS* process which satisfies the transition rule: $SUCCESS \xrightarrow{\omega} STOP$. Thus, *SUCCESS* is a simple process that performs a success event ω and then stops. ω is a special event that is introduced purely as a device to define the success of a test and will not be part of the universal alphabet Σ.

Success of a test is now characterised by whether the process:

$$(P \,||_{\Sigma}\, T) \setminus \Sigma$$

can perform ω. Where T is constructed from the CSP operators along with the *SUCCESS* process. Thus the occurrence of the ω event signals that P has agreed to perform some behaviour offered by T. You can think of T as representing some abstract test harness that is attached to P.

10.11 May Testing Equivalence

There are three possible outcomes of such a test when applied to a particular process: it might always succeed, always fail or sometimes succeed.

If $(P \mid\mid_\Sigma T) \setminus \Sigma$ can perform ω then we will say that P *may pass* T, written $PmayT$. That is, $\exists$ an execution of $(P \mid\mid_\Sigma T) \setminus \Sigma$ that results in ω. It is possible that there are executions of P that do not result in ω so success is not guaranteed, hence the name *may testing*.

P and Q are now deemed to be may testing equivalent, written $P =_{may} Q$, iff:

$$\forall\, T \bullet PmayT \Leftrightarrow QmayT$$

In other words the set of tests that P may pass is exactly the same as the set that Q may pass. May testing equivalence is known to give the same equivalence as the traces model [10].

An analogous notion of *must testing* can be defined by requiring that all (maximal) executions of $(P \mid\mid_\Sigma T) \setminus \Sigma$ result in an ω. This gives an equivalence that corresponds to the failures model of CSP. The condition of maximality is required because there will always be the possibility of short runs that have not reached the success state but would if allowed to continue and so should not be regarded as a failure of the test.

10.12 May Testing Non-interference

We can use the form of equivalence given by may testing to give another statement of non-interference.

Definition: S is *mayNI* if for all High processes H_1 and H_2, with $\alpha H_i \subseteq H$:

$$H_1 \mid\mid_H S \mid\mid_L T =_{may} H_2 \mid\mid_H S \mid\mid_L T \tag{16}$$

With $\alpha T \subseteq L$

This gives a weaker characterisation than those given above as may testing ignores distinctions that Low may draw based on observations of non-determinism. We can give a formulation similar to, say, Equation 9 by requiring must testing equivalence [84].

10.13 Non-deducibility

Equation 17 is the natural thing to write down in a *may testing* framework. However, by altering this slightly, we can mimic one of the early formulations: *non-deducibility* due to Sutherland, [88]. The essential idea is to stipulate that whatever observations Low may make of the system the space of possible High level *inputs* consistent with those observations is unchanged. Intuitively this is rather appealing and appears to address the encryption problem: whatever ciphertext Low observes he cannot reduce the space of plaintexts compatible with this ciphertext.

We need to partition the High level events into inputs and outputs. We then restrict the high-level processes in the definition to ones with an alphabet drawn

only from High *inputs* and we use this in the definition of Equation 17. This is not really a natural thing to do in a process algebraic framework and indeed it and Sutherland's original formulation are found to be flawed, as observed by Johnson and Wittbold [90].

Unsurprisingly, the problem arises from ignoring the High outputs. Wittbold et al, construct a system that satisfies non-deducibility but for which High can modify his behaviour based on observations of High outputs in such a way as to signal to Low. The construction amounts to a stream cipher that High can induce to stutter. This allows him to predict certain bits and so leak plaintext to Low. In fact, as remarked in [77], this is really conceptually this same as High somehow being able to observe the bits of a cipher stream before he submits his plaintext. If he can somehow achieve this he can now exclusive or these into the High inputs (plaintext). The self inverse property of Vernan encryption then results in raw plaintext being visible to Low.

It might seem unlikely that such a flawed installation of a cipher device would be implemented in a secure system but the point is that the non-deducibility formulation fails to detect this problem.

10.14 Non-deducibility on Strategies

The difficulty with non-deduciblity as originally presented is that it takes no account of possible malicious behavious by High, maybe in collusion with Low. Notice that the counter-example provided by Johnson and Wittbold still satisfies non-deducibility in the sense that, assuming that any cipher stream is possible, then any plaintext is consistent with the ciphertext that Low observes.

Having made this observation, Johnson and Wittbold propose a more refined version of non-deducibility: they introduce the notion of High strategies and use this to define *non-deducibility on strategies*. In effect High is allowed to make his choices at run time on the basis of observations he can make on the system. They now require of the system that it satisfy non-deduciblity, whatever strategy High may adopt.

Note that if High had to resolve all his choices ahead of time, for example, had to decide on what text to submit to the cipher box before getting a chance to observe any of its outputs, then he could not communicate with Low.

The strategies of Wittbold et al, are really just instances of High CSP processes. As a result, NDoS turns out to be equivalent to our Equation 17. Thus the CSP testing approach rather naturally shows up the flaw of non-deducibility and leads naturally to the notion of non-deducibility on strategies.

It is also interesting to consider this example in the context of our power-bisimulation framework. If we think of the τ's as encoding the output of the stream cipher, the loose bi-simulation property ensures that High cannot influence them. High's exclusive-oring of predicted bits into the plaintext is tantamount to tampering with these τ's. There are however some subtle issues here of causality. It is not clear that this has really been precisely captured in the formalism. We would like to show, for example, that predicting and compensating for the bits

is equivalent to High forcing the bits to all equal zero and just using the original plaintext. How to capture the act of prediction is rather delicate.

A formulation that is basically equivalent to Equation 17, called non-deduciblity on composition (NDC), can be found in Gorrieri et al, [19] except that they use a conventional bi-simulation, so theirs is slightly stronger. Their formulation predates ours but it is nice to see essentially the same formulation emerging from a different approach.

An elaboration of testing equivalence, due to Schneider [83], allows for non-refusable Low events. In effect we constrain the space of tests to processes that are receptive on the L non-refusable events. We could also consider constraining the space of High processes and we will return to this in the section on generalisations of CSP non-interference.

10.15 The Lowe Approach to Information Flow

Lowe points out some of the limitations of existing formulations of NI [48]. None seem to give just the right characterisation. He traces part of the problem to the way non-determinism can be resolved in CSP. Consider:

$$P = h \rightarrow STOP \;|||\; (l \rightarrow STOP \sqcap l' \rightarrow STOP)$$

Intuitively this should be secure: the intuition behind the interleave operator is that it allows both processes to execute entirely independently. However if one considers the LTS, figure 4 upper diagram, we see that there appear to be two internal decision points. If these two choices are resolved differently, lower diagram, we get an information flow.

Thus the choice has been resolved differently before and after the occurrence of the h. In fact the original CSP specification only really had one choice but the LTS representation appears to have two. These are really the same and so should really be resolved consistently. The inconsistent resolution has in, effect, introduced a spurious causal relationship between the high and the low events not intended in the original specification.

Lowe's solution is to introduce a demon that ensures that such choices are resolved in a consistent way. He introduces additional labels on the LTS to carry this syntactic information. This allows him to define the notion of consistent refinement and then defines a system to be non-interfering iff any consistent refinement satisfies non-interference in the sense of Equation 5 (except that in this case a formulation suitable for deterministic systems is appropriate and so it is enough to require that the initials match rather than that the ready sets match).

11 Generalisations

Thus far we have dealt with the problem of characterising the complete absence of information flows through a system. We have seen that even in this comparatively simple situation it is remarkably difficult to arrive at a satisfactory

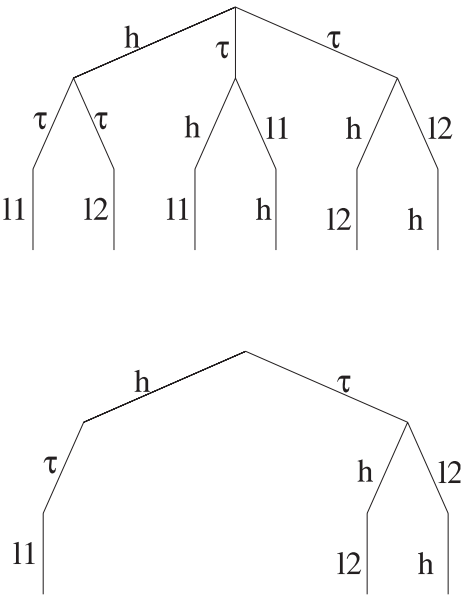

Fig. 4. The Labelled Transition System for P and its "refinement"

characterisation, even where we have abstracted from questions of time or probability. Later we will discuss ways to incorporate these aspects too but in this section we will address generalisations of what we have developed so far.

11.1 Limitations of Non-interference

Up to now we have used the idea of non-interference to characterise the complete absence of information flow or indeed stronger, the absence of any causal flow. Typically this is too strong for practical applications. We have already encountered the encrypted channel example in which we have causal flow but no (significant) information flow. Other situations also call for restricted or controlled flows. Sometimes this will be due to policy requirements. Downgraders are an example in which certain flows are allowed under certain circumstances, i.e., the classification of a previously sensitive file is reduced to unclassified by a security officer or maybe just due to the passage of time. Another example is the downgrading of statistical imformation derived from an otherwise sensitive database.

Sometimes it is more a question of functionality. Typically strict non-interference simply isn't feasible due to clashes of resource demands, etc. A simple example is the One Way Regulator or NRL pump. This is a device intended to allow information flow from Low to High but not from High to Low. In fact some flow from High to Low is necessary to regulate the flow from Low and

avoid buffer overflow. For the *NRL Pump* the bandwidth of the High to Low regulating channel is severely restricted.

More generally there is a trend away from simple MLS-style policies to richer styles of policy involving finer granularity and more subtle, dynamic (possibly history- or location-based for example) controls of accesses. A simple example is the Chinese-Walls-style policy, mentioned earlier, in which an access to one class of file denies access to files in a conflicting class.

In this section we investigate to what extent these policies and requirements can be captured in a non-interference framework. There is ,of course, a question as to whether such a framework is the appropriate. It may well be that a large class of such policies simply do not fit naturally or indeed at all into a non-interference framework. It is interesting however to investigate how far we can push things.

The following generalisation of Equation 5 suggests itself:

$$\forall\, tr, tr' \in traces(S) \bullet tr \approx tr' \Rightarrow$$
$$\mathcal{A}_H((S \mid\mid Constrain)/tr) \equiv \mathcal{A}_H((S \mid\mid Constrain)/tr') \tag{17}$$

Firstly observe that $\approx$ can be an arbitrary equivalence on traces, including but not confined to those induced by purge-like functions. We will discuss some of the possibilities that suggest thenselves shortly.

$\mathcal{A}_H$ denotes the operator chosen to abstract parts of the interface to S: eager, lazy, projection or some mixture.

Constrain denotes the envelope of behaviours that we are concerned about (not necessarily itself a process). High behaviours outside this envelope are allowed to interfere with Low under such a definition.

$\equiv$ denotes an appropriate process equivalence.

Alternatively we could seek to generalise Equation 8:

$$U_1 \sim U_2 \Rightarrow \mathcal{A}_H(S \mid\mid U_1 \mid\mid Constrain) \equiv \mathcal{A}_H(S \mid\mid U_2 \mid\mid Constrain) \tag{18}$$

where $\sim$ denotes a suitable equivalence over High processes.

Using a testing approach allows us to delimit the space of tests, as in Schneider's work, or indeed to constrain the space of high users that we quantify over. The latter freedom renders redundant the *Constrain* term that we used above.

11.2 Encrypted Channels

As a first example of how such generalised formulations of non-interference might be applied let us return to the encrypted channel in which High feeds a classified plaintext into an encryption device and the resulting ciphertext is transmitted over some open channel c. We will suppose that the encryption algorithm is secure and any keys that might be involved are uncompromised. Indeed we could simply assume that we are dealing with a one-time-pad encryption.

Here we instantiate the equivalence relation $\approx$ in definition 18 by:
$\approx$ is defined by:

$$tr \approx tr' \Leftrightarrow \#(tr \upharpoonright H) = \#(tr' \upharpoonright H)$$

that is, plaintexts of equal length are regarded as equivalent, and define $\mathcal{A}_H$ as projection on the channel c: renaming 0's and 1's to some single symbol, $\bullet$ say.

A secure encryption channel now passes the form of non-interference defined in Equation 17 instantiated with these abstractions and equivalences. Indeed the information flow seems to have been quite accurately encoded: Low can determine the length of a High message transmitted over c but not its contents. It does, however, fail to take account of the fact that Low could detect when identical cipher-texts have been transmitted. Presumably if we really are dealing with a one-time-pad this is not relevant: the occurence of identical cipher-texts is firstly extremely unlikely and secondly signifies nothing. On the other hand, for a block cipher in, say, electronic code book mode it could be highly significant. To capture this situation we can alter the abstraction applied to the c channel. We now allow for process equivalence up to renaming on the alphabet of c. We should now think of the alphabet of c as being the bit strings of length l, where l is the length of the cipher blocks. For electronic code book mode it is appropriate to consider the renaming to be constant on the blocks. This is analogous to our encoding of pseudo-anonymity that will be discussed later.

This last example suggests that it might be useful to consider process equivalence up to isomorphism, where the isomorphism could be more general than simple renaming that we considered above.

11.3 Downgrading Statistical Information from a Sensitive Database

Another situation that can be encoded rather naturally is one in which unclassified statistical information is extracted from an otherwise classified database. Here we simply choose an equivalence over the state space of the database such that states giving rise to the same statistical values are regarded as equivalent.

11.4 File Editing

In general many different editing sequences can lead to the same final text, ignoring mark-up features. Certain edit commands will commute, some will negate others. Thus, editing sequences leading to identical final texts could be classed as equivalent. More generally we could define equivalence under arbitrary rewrite rules on traces, but this might hit decidability problems.

11.5 Operating Systems

The purpose of an operating system can be thought of as maintaining the illusion for each user that they are interacting with their own private system. Thought

of this way we see that this kind of requirement is remarkably close to the idea of non-interference: the user's experience of interacting with the system should be largely unaffected by the presence of other users. In fact it is possible to get away with a weaker requirement here as we are not really worried if the user can infer something about the existence and behaviour of other users as long as he still gets the functionality he expects. We will see shortly that this is closely related to fault-tolerance requirements. We thank John McHugh for this observation.

11.6 Intransitive Non-interference

In [74], Rushby discussed what he calls *intransitive non-interference*. The name is actually rather deceptive as it is really about intransitive information flow policies. In some situations it is desirable for information to be allowed to flow in certain circumstances from H to L via some intermediate process, D say, that controls or as least audits the flows. No direct flows from H to L are allowed. At first this seems somewhat bizarre as we are in effect stipulating that $H \Rightarrow D$ and $D \Rightarrow L$ whilst denying $H \Rightarrow L$.

Rushby gives a couple of examples where this kind of requirement arises: downgraders and crypto boxes. Thus, in the case of a down-grader, we allow certain files to flow from a high classification to a lower classification but only if this is controlled by a downgrading device.

What really seems to be asserted here is that any flow from H to L should be regulated or at least audited by D. Rushby deals with this by introducing an *ipurge* function that no longer acts point-wise on individual events but accounts for downgrade events: thus we do not purge events that have been downgraded.

In [28] Goldsmith and Roscoe critique this, pointing out the Rushby's formulation can allow undesired flows. They give an alternative formulation using their determinism approach.

We can use the equivalence induced by *ipurge* to cast Rushby's examples into our generalised formulation. It may be possible to capture more general forms of intransitive information flow policies by suitable choices of traces equivalence.

11.7 Fault-Tolerance

Fault-tolerance and masking can also be encoded as non-interference style properties: roughly speaking, faults should not interfere with users. This idea crops up in [89] and later in [73,87]. For fault-tolerance, however, a weaker version may be acceptable as we are not really concerned with the possibility of information flowing from the faults to the users. It is thus not necessary in the definitions to demand equivalence, refinement is enough: that user functionality is unaffected. We are not really concerned if the errors may be able to resolve some of the non-determinism seen by the users. Indeed we might want information about the error behaviour to flow to the users in the form of warnings, alarms, etc.

We can thus define S to be tolerant of fault behaviours within a certain envelope defined by the process $FAULTS$ by:

$$(S \;||_{faults}\; FAULTS) \setminus faults \sqsubseteq_F (S \;||_{faults}\; STOP) \setminus faults \tag{19}$$

Thus $FAULTS$ ($\alpha FAULTS = faults$) is a process that encodes the failures behaviours we hope to be able to mask or tolerate. For example, for a Byzantine agreement algorithm, $FAULTS$ might specify the number and type of failures to be tolerated. The RHS of the refinement represents the fault-free behaviour of S. We are thus saying that as long as the fault behaviours stay within the envelope defined by $FAULTS$, the system should continue to be a refinement of the fault-free system, i.e., continues to provide the same functionality. We are assuming that the fault-free system provides the required functionality though strictly speaking this would need to be seperately verified against a suitable requirements specification.

Note the use of refinement in this definition rather than equality as used in the definitions of secrecy.

Alternatively, suppose that $MISSION$ encodes the mission critical functionality, then we could require:

$$S \;||_{faults}\; FAULTS \sqsubseteq_F MISSION \tag{20}$$

Thus, even in the presence of a faults scenario within the envelope defined by $FAULTS$ the mission critical functionality should remain. We could formulate a series of such requirements specifying acceptable degradation of functionality under increasingly severe fault scenarios:

$$S \;||_{faults}\; FAULTS_i \sqsubseteq_F MISSION_i \tag{21}$$

11.8 Intrusion-Tolerance

In principle we could apply this approach to defining intrusion-tolerance with something analogous to $FAULTS$ to encode attack scenarios. It is not really clear that attack scenarios can be so readily modelled. One could, in principle, try to model hostile capabilities (cf security protocols analysis [78]) maybe even including cost factors (which, inter alia, might help reduce the search space). This remains an avenue for research.

11.9 Anonymity

Another property that can be given an elagant non-interference-style formulation is anonymity. It is a sort of converse to authentication: authentication is about a process being assured of the identity of agents or processes with which it is interacting. Anonymity is concerned with preventing identities from being revealed. As with authentication, anonymity comes in many flavours depending on the application and requirements.

It is clear that anonymity has much in common with confidentiality. The latter can be thought of as a kind of anonymity over a message space. Indeed our Equation 8 could be interpreted as a statement of anonymity: Low cannot distinguish which of two possible users are interacting with the system through the High interface. This is actually a little strong for the usual meaning of anonymity as it requires that any user be indistinguishable from no user. More typically anonymity means that you know that an action has been performed but are unable to tell who is associated with it. Process algebraic definitions of various flavours of anonymity can be found in [85].

Pseudo-anonymity can be similarly formulated using a constant permutation in the renaming abstraction rather that just projection. This allows correlation between pseudonyms across time.

11.10 Dynamic Separation of Duty

Earlier we mentioned dynamic separation of duty policies. It turns out that such policies can be remarkably simply stated in a process algebra. As a simple example, suppose that an agent A can choose between two roles, $Role_1$ and $Role_2$, but that these are declared mutually exclusive. Once we have expressed these roles as CSP processes, we can express this in the CSP notation as:

$$A = Role_1 \,\Box\, Role_2 \tag{22}$$

This gives a Chinese walls style policy for these two roles. Various generalisations to multiple roles and to situations in which the choices are constrained in certain ways are obvious.

11.11 Dealing with Probability

In principle it seems straightforward to extend the framework presented earlier to deal with probability by asserting that, for example, the probabilities of transitions for related states should be equal. In practice the formalism and verification gets cumbersome. One rather natural way to introduce probability is using the testing framework:

Let

$$S_i^* := Abstract_H(S \,||_H\, U_i)$$

Then we could assert that S is probabilistically non-interfering if:

$$\forall\, U1 \approx U2 \wedge T \in Tests \bullet$$
$$Prob(Success(S_1^* \,||_L\, T)) = Prob(Success(S_2^* \,||_L\, T)) \tag{23}$$

It is far from clear that such a property would be tracable from a verification point of view. However unwinding this definition to some appropriate

bi-simulation property might prove more tractable. Here one would be dealing with the deltas in probabilities, in particular asserting that probabilities are unchanged by the occurrence of High events, as the system evolves. Thus we might assert something along the lines of:

$$S_1 \approx S_2 \Rightarrow Prob(S_1 \xrightarrow{h} S_1') = Prob(S_2 \xrightarrow{h'} S_2') \wedge S_1' \approx S_2' \tag{24}$$

where $Prob(S_1 \xrightarrow{h} S_1')$ denoted the probability of a transition labelled h from the state S_1 to S_1'. Note that such probabililites will only really make sense where they arise due to "essential" non-determinism of the system rather than non-determinism that can be resolved by agents or processes interacting with the system. This might sidestep the need to assign absolute probabilities to transitions. We are grateful to Cathy Meadows for pointing this out.

11.12 Dealing with Time

Similarly we could move to a (discrete) timed model, using, for example, the tock dialect of CSP, with assertions that the times at which events become available to Low are independent of High activity. Again things rapidly get very complex-certainly difficult to model-check. Toy examples, for example, encrypted channels and maybe the One-Way-Regulator, can probably be pushed through but real systems seem out of reach for the moment.

12 Future Directions

In this section we outline some directions for future work.

CSP draws a distinction between internal and external non-determinism. These are similar to the *don't know*, *don't care* style of distinctions drawn in many formal methods. We have seen that these two flavours, although they are fine for investigating safety and liveness properties, are not enough to capture some aspects of information security. In particular, for security, we often need to distinguish *probabilistic* or *essential* non-determinism. We need to be very careful how we refine non-determinism: some non-determinism may be vital to security, for example, that arising from a stream cipher.

We have mentioned a number of approaches to formalising secrecy and hinted at possible connections. These need to be more fully investigated and understood. Besides these a number of other, recent proposals have appeared that appear to be related, for example: Sabelfeld and Sands [79], Mantel [51], and Pinsky [69].

We have illustrated how a few requirements and examples can be encoded in our generalised form of NI_{CSP}. It remains to try to tackle some more realistic examples in this framework: the Java security model, databases, smart cards. It would also be interesting to investigate more fault-tolerance and even intrusion-tolerance examples. It would also appear that a number of non-security applications may be usefully addressed using this kind of framework. Indeed it

seems that, although these ideas were developed in a security context, they may turn out to be at least as useful applied in other contexts.

A major challenge is to extend such techniques to address time and probability. In principle this seems straightforward. To extend the models in a way that remains tractable is far from straightforward. Even without time and probability we are straining the limits of what is tractable from a verification point of view. On the other hand some of the subtleties that arise are due to the abstraction that we make of, for example, time. Thus in some respects models that include time may be simpler, at least in a conceptual if not complexity sense.

12.1 Composition Results

We saw earlier how using a bi-simulation formulation of non-interference leads to a very simple and elegant proof of a composition result. It seems likely that using such a bi-simulation formulation of the generalised forms of non-interference could similarly lead to simple proofs of compositionality, or alternatively, where compositionality fails, shed light on exactly why it fails. Indeed it seems likely that one could give a useful characterisation of the class of equivalence relations, i.e policies, that give rise to compositionality.

12.2 Links to Cryptographic Analysis Techniques

To date the cryptographic definitions of secrecy and the formal definitions that we have presented have been developed entirely independently. This is particularly clear in the area of security protocol analysis in which we understand very well how to analyse the strength of the cryptographic algorithms and primitives on the one hand and the protocols on the other. The latter tends however to abstract away from the details of the cryptographic primitives and it is still poorly understood how to link the results of the two styles of analysis. As a result it is possible for subtle interactions between the crypto primitives and the protocol design to slip through analysis. It is quite possible to have a protocol that is perfectly secure inplemented with algorithms that in themselves are secure and yet the whole is seriously flawed. An example of this can be found in [12].

Ideally we would like to be able to tie together the two styles of analysis in a way that remains tractable. An attempt to do this is Lincoln et al [47] but the resulting framework is very elaborate and it is unclear that anything but rather simple examples can be handled. [16] applies the idea of non-interference to the analysis of cryptographic protocols.

Typically the cryptographic definitions and proofs involve reduction arguments and in some cases testing style definitions. The spy is allowed to submit an arbitrary number of plaintexts of his choice to an encryption device and to observe the resulting ciphertexts. His choices can be adaptive: they can depend on the outcomes of previous experiments. He then finally submits a pair of distinct plaintexts and gets the resulting ciphertexts back in an arbitrary order. If he is able to guess which ciphertext corresponds to which plaintext with a significantly greater than 0.5 probability the device is deemed insecure, otherwise

it is deemed secure. The details of the definition of *significantly greater than 0.5* is rather technical and need not concern us here. See for, example, [30,3].

The testing style definitions of non-interference that we have presented above may provide a point of contact bewteen the cryptographic and formal methods approaches. This is a topic for future research. One can think of non-deducibility on strategies in terms of the definition of resistance against adaptive chosen plaintext attack: Low is allowed to repeatedly input High behaviours to the system and observe the resulting behaviours through his interface. If eventually he can glean enough insight into the behaviour of the system to be able to make better than evens guesses as to which of a pair of High behaviours has occurred then the system is deemed to be flawed. The analogy is rather subtle and there are some interesting and illuminating differences.

12.3 Subliminal Channels and Information Hiding

Another topic that may be worth investigating using the techniques presented in these lectures is that of subliminal channels and information hiding. An attempt to formally define such channels is given in Desmedt [14]. It would be interesting to see if similar formalisation might be possible using one of the generalised forms of non-interference described here. The idea behind information hiding is to conceal the existence of information in a message or in data. This involves trying to make messages with different hidden information look indistinguishable to an observer lacking some appropriate key. It is clear that this has a similar feel to some of the properties and policies that we have been trying to capture: that behaviours in a given equivalence class be indistinguishable to certain observers.

12.4 Automated Support

Besides the theoretical problems that remain there is still the question of developing suitable tools and techniques to make the verification of significant applications feasible and ideally routine. Significant strides have been made with the usability of theorem provers but they still require specialist expertise to use. Similarly major strides have been made in the application of model-checking to security, see [78]. Model-checking holds out more promise as far as the degree of automation typically achievable but here too highly specialised expertise is still required to keep the state spaces of the model down to managable sizes. Important advances are being made in this area using data independence, for example Lazic et al [44], combining data independence and induction techniques, Broadfoot [6], and using *predicate abstraction*, Saidi [80].

12.5 Links to Other Process Algebras

We have seen that CSP is highly effective at capturing many of the properties of concern but also that we have found ourselves hitting the limits of the framework and having to introduce constructs usually regarded as outside conventional

CSP. CCS has also been applied with great effect to information security, see the chapter by Focardi and Gorrieri in this volume. It seems likely that no single, existing process algebra will provide us with all the machinery we need for information security applications. For example, besides the problems of distinguishing flavours of non-determinism, we need to be able to address mobility. Mobility and dynamic networking is an increasingly pervasive aspect of modern systems and the research community needs to get to grips with it. A number of process algebras have been proposed to address issues of mobility, location etc. These include the pi-calculus [64], the ambient calculus [8]. The chapter by Andy Gordon in this volume provides more on this topic. We need to see what features of these we can adapt or incorporate.

Asynchronous algebras may also prove fruitful to investigate. The asynchronous model of communication is similar to our notion of non-refusable events. We no longer assume a hand-shaking model of communication in which both sides synchronise on an action. Actions are launched into the ether and their reception is not guaranteed. This is in many respects a highly appropriate model for security applications in a distributed environment.

12.6 Static and Typing Analysis

Another rather different approach to defining secrecy is represented by the static analysis and typing analysis techniques of Volpano [91] and others, for example [35]. Here non-interference properties are cast in terms of static or typing conditions on a programming language or process algebra.

13 Conclusions

In these lectures I have sought to give the reader an overview of the evolution of mathematical formulations and frameworks for a number of security requirements and policies. We have concentrated on the notion of secrecy or confidentiality and, in particular, variants of the idea of non-interference as a way to formally characterise the absence of information flows.

The central thesis of these lectures is that characterising non-interference reduces ultimately to characterising the equivalence or indistinguishability of processes. Several corollaries flow from this observation:

Establishing how to characterise the equivalence of processes is itself a fundemental and delicate question. Indeed the whole question of what we mean by a process is intimately related to what processes should be regarded as equal. We should not therefore be too surprised that the problem of what formulation of non-interference is *correct* has remained controversial in the information security community for more than 20 years. Indeed it seems likely that there is no single, Platonic formulation of secrecy. There are no Maxwell's field equations for secrecy, as it were.

Which form of process equivalence is appropriate seems to depend on what model of computation we adopt and what observations and experiments we deem

the environment capable of performing on the system. In some cases it will even depend on what computational capabilites we assume of the environment. To some extent this is just the standard problem facing any exercise in mathematical modelling: any model will necessarily be an abstraction of reality. As far as possible we seek to make our models faithful, at least as far as the properties of interest are concerned. Usually in the interests of tractability, we are often forced to make sweeping assumptions and approximations.

On the more positive side, thinking in process algebraic terms and in terms of process equivalence provides many insights and ready-made results. Process algebras deal carefully with questions of non-determinism, process equivalence, composition and so on. We have seen that the idea of unwinding is closely analogous to that of bi-simulation and that many of the historical formulations of non-interference can be cast as flavours of testing equivalence.

We have seen that a process algebraic framework provides an excellent basis from which to explore various generalisations of the original, rather binary concept of non-interference. It also provides an effective framework for reasoning about compositionality.

It should also be acknowledged that we have found ourselves straining the machinery of, for example, CSP to try to capture all the subtleties that information security throws up. Indeed it would appear that no existing process algebra is entirely suited to capturing all the aspects of information security. We have also seen that questions of causality seem not adequately addressed by existing process algebras. This has its plus side: the fact that we are testing the limits of existing theory when trying to apply it to security problems provides new challenges for the theory and stimulates further research.

Significant advances have been made in recent years in the specification and verification of security requirements, protocols, etc,, [78]. I hope at least to have conveyed the point that information security raises some fascinating and fundamental challenges both at the theoretical level and at the practical level of tools and techniques for verification.

The concept of non-interference has been a major preoccupation of the information security community for more than 20 years. We have discussed at length the problems in obtaining a satisfactory definition. Besides this there remains the question of what purpose, if any, non-interference actually serves in the specification and development of secure systems. It has been pointed out that no real security policy ever mentions the notion explicitly and in any case it is, in practice, impossible to realise in any real system: contention for resources means that it can never be fully attained in practice. Add to these concerns the point that non-interference is such an abstract notion that it is generally extremely difficult to map it down to the implementation level. All this might suggest that non-interference is little more than a rather elegant, theoretical debating point.

On the other hand, information security policies are concerned with what information flows are allowed and which are illegal. Thus, information-flows and their absence are central to such policies. It would seem, therefore, that something akin to non-interference, characterising the absence of information flow,

must be a fundamental element of any such policy. If we cannot get the specification and verification of the absence of certain information flows right, then we really do not understand the foundations of our subject.

It should also be remarked that we are starting to see a number of applications of non-interference-like concepts for a far wider class of applications and requirements. Maybe the concept will actually prove to be more useful beyond the realm of information security, in which it was conceived. The proof of the usefulness of the concept will only really be established when it has found an effective role in the specification and verification of real applications.

14 Acknowledgements

The author would like to thank the following people who have contributed to the ideas described here, commented on drafts and with whom I have had many enjoyable and fruitful discussions: Sven Dietrich, Simon Foley, Paul Gardiner, Michael Goldsmith, Roberto Gorrieri, Joshua Guttman, Gavin Lowe, John McHugh, John McLean, Cathy Meadows, Sylvan Pinsky, Bill Roscoe, Pierangela Samarati. A particular thanks to goes Steve Schneider, with whom many of these ideas were developed. A special thanks also goes to Jeremy Jacob, who was instrumental in stirring my interest in applying CSP to the problems of information assurance.

Thanks also to MSR in Cambridge, UMBC Maryland and NR Oslo for hospitality during the preparation of parts of this material. Finally a thanks goes to the DERA Strategic Research Programme for support during the development of many of the ideas presented here.

References

1. Abadi, M. and Gordon, A.: A calculus for Cryptographic Protocols: the Spi Calculus, Information and Computation (1999)
2. Bell, D. E. and LaPadula, L. J.: Secure Computer System: Unified Exposition and Multics Interpretation, Tech report ESD-TR-75-306, Mitre Corp, Bedford, Ma. (1976) 7
3. Bellare, M. and Rogaway, P.: Entity Authentication and key Distribution, Advances in Cryptography- Proceedings of Crypto (1993) 55
4. Biba, K. J.: Integrity Considerations for Secure Computer Systems, US Airforce Electronic Systems Division (1977) 10
5. Brewer, D. F. C., Nash, M. J.: The Chinese Wall security policy, in Proceedings of the IEEE Symposium on Security and Privacy, (1989) 206-214 9
6. Broadfoot, P. et al: Automating Data Independence, European Symposium on Research in Computer Security, LNCS vol 1895, Springer (2000) 55
7. Brookes, S. D. and Roscoe, A. W.: An Improved Failures Model for Communicating Sequential Processes Springer Verlag, Proceedings NSF-SERC Seminar on Concurrency (1985) 17
8. Cardelli, L.: Mobility and Security, Lecture Notes for the Marktoberdorf Summer School (1999) 56

9. Clark, D. R. and Wilson, D. R.: A Comparison of commercial and military computer security policies. In Proceedings of the IEEE Symposium on Security and Privacy, (1987) 184-194 9
10. Cleaveland, R. and Hennessy, M.: Testing equivalence as a bisimulation equivalence. Formal Aspects of Computing, Volume 5, (1993) 1-20 44
11. Cohen, E.: Information Transmission in computational Systems. Sixth ACM Symp. on Operating Systems Principles, November (1977) 133-139 12
12. Coppersmith, D. et al.: Low-exponent RSA with related messages. In Advances in Cryptology - EUROCRYPT '96 (Lecture Notes in Computer Science 1070), Springer-Verlag, (1996) 1-9 54
13. Davies, J., Schneider S. A.: A Brief History of Timed CSP, Theoretical Computer Science, 138, (1995) 17
14. Desmedt, Y. and Yung, M.: Minimal cryptosystems and defining subliminal-freeness. In Proceedings 1994 IEEE International Symposium on Information Theory, p. 347, Trondheim, Norway, June 27-July 1, (1994) 55
15. US Department of Defense: DOD Trusted Computer Security System Evaluation Criteria (The Orange Book), DOD 5200.28-STD, (1985) 7
16. Durante, A. et al: A Compiler for Analysing Cryptographic Protocols using Non-Interference, ACM Trans. on Soft.Eng. and Method, 9(4) (2000) 1-9 54
17. `http://www.formal.demon.co.uk/` 36
18. Feiertag, R. J.: A technique for Proving Specifications are Multi-level Secure Technical report CSL109, CSL, SRI International (1980) 12
19. Focardi, R. and Gorrieri, R.: A Classification of Security Properties, JCS, 3(1): (1995) 5-33 34, 46
20. Focardi, R, Ghelli, A. and Gorrieri, R.: Using noninterference for the analysis of security protocols, DIMACS workshop on Design and Formal Verification of Security protocols (1997)
21. Focardi, R., Gorrieri, R.: The Compositional Security Checker: A Tool for the Verification of Information Flow Security Properties. IEEE Trans. on Soft. Eng., 23(9): (1997) 550-571 16
22. Foley, S. N.: A Taxonomy for Information Flow Policies and Models, in Proceedings of IEEE Symposium on Security and Privacy, IEEE Press (1991) 9
23. Foley, S. N.: The Specification and Implementation of Commercial Security Requirements including Dynamic Segregation of Duties, 4th ACM Conference on Computer and Communications Security, ACM Press, (1997) 10
24. Gardiner, P.: Algebraic Proofs of Consistency and Completeness. Theoretic Computer Science, (1995) 150-161 34
25. Gardiner, P.: Power simulation and its relation to traces and failures refinement, ENTCS, vol 32, URL: `http://www.elsevier.nl/locate/entcs/volume32.html` 35
26. Goguen, J. A. and Meseguer, J.: Security policies and security models, IEEE Symposium on Security and Privacy, (1982) 12
27. Goguen, J. and Meseguer, J: Inference Control and Unwinding, Proceedings of the IEEE Symposium on Research in Security and Privacy (1984) 12, 30
28. Goldsmith, M. H. and Roscoe, A. W.: What Is Intransitive Noninterference? Proceedings of the Computer Security Foundations Workshop, IEEE Press(1999) 50
29. Gollmann, D.: Computer Security, Wiley (2000) 7
30. Guttman, J. et al: The Faithfulness of Abstract Encryption, to appear 55
31. Haigh, J. T.: A Comparison of Formal Security Models, Proc 7th National Computer Security Conference, Gaithersburg, September (1984) 88-119 16

32. Harrison, M. A. et al: Protection in operating systems. Communications of the ACM, 19(8) , August (1976) 461-471 9
33. He, J. and Hoare, C. A. R.: Unified Theories of programming. Prentice Hall International, (1998) 38
34. Hennessy, M.: Algebraic Theory of Processes, MIT Press (1989)
35. Hennessy, M.: The security pi-calculus and non-interference, Computer Science Technical Report 2000:05, School of Cognitive and Computing Sciences, University of Sussex. 56
36. Hoare, C. A. R.: Communicating Sequential Processes, Prentice Hall (1985) 17
37. Jacob, J. L.: Security Specifications, Proceedings of the IEEE Symposium on Research in Security and Privacy (1988) 29, 42
38. Jacob, J. L.: Basic Theorems about Security Journal of Computer Security, Vol 1 Number 4, (1992) 385-411 42
39. Johnson, D. and Thayer, F.: Security and the Composition of Machines, In Proceedings of the Computer Security Foundations Workshop, IEEE Press, (1988) 16
40. Kang, M. H. et al: Design and Assurance Strategy for the NRL Pump, Computer, Vol. 31, No. 4, April (1998) 56-64 10
41. Kemmerer, D.: Verification Assessment Study Final Report NCSC report (1986) 11
42. Lakatos, I.: Proof and Refutations: The logic of mathematical discovery. Cambridge University Press, (1977) 5
43. Lampson B.: Protection, ACM Operating Systems Reviews, 8, (1974) 6
44. Lazic, R. and Nowak, D.: A Unifying Approach to Data-independence, In Proceedings of the 11th International Conference on Concurrency Theory (CONCUR 2000), Lecture Notes in Computer Science. Springer-Verlag, August (2000) 55
45. Lee, S. and Zakinthinos, A.: A General Theory of Security Properties, Proceedings of the IEEE Symposium on Research in Security and Privacy (1997) 16
46. Lee, T. M. P.: Using Mandatory Integrity to Enforce 'Commerical' Security, Proceedings of the IEEE Symposium on Research in Security and Privacy, (1988) 140-144 10
47. Lincoln, P. et al: Probabilistic polynomial-time equivalence and security analysis, Proceedings of FM'99 (1999) 54
48. Lowe, G.: Probabilities and Priorities in Timed CSP, D.Phil. thesis Oxford University (1993) 46
49. Lowe, G.: Defining Information Flow University of Leicester tech report (1999)
50. MacKenzie, D.: Computers and the Sociology of Mathematical Proof. Prepared for Northern Formal Methods Workshop, Ilkley, September (1998) 5
51. Mantel, H.: Unwinding Possibilistic Security Properties. In Proceedings of ESORICS (2000) 53
52. McCullough, D.: Specifications for Multi-level Security and a Hook-up Property, Proceedings of the IEEE Symposium on Research in Security and Privacy (1987) 16
53. McCullough, D.: Noninterference and the Composition of Security Properties Proceedings of the IEEE Symposium on Research in Security and Privacy (1988) 16
54. McHugh, J.: Covert Channel Analysis. A chapter in the *Handbook for the Computer Security Certification of Trusted Systems*, (An ongoing series published by the Center for High Assurance Computing Systems, Naval research Laboratory, 4555 Overlook Ave, SW, Washington, DC 20375,) November 1994 – Revised December 1995. Available at `http://chacs.nrl.navy.mil/publications/handbook/index.html` 11

55. McHugh, J.: A Formal Definition for Information Flow in the Gypsy Expression Language. In *Proceedings of The Computer Security Foundations Workshop,* Mitre Corporation, Bedford, MA (1988) 147-165 11
56. McIver, A. et al: Refinement-oriented probability for CSP, Formal Aspects of Computing 8(9) (1996)
57. McLean, J.: Security Models Encyclopedia of Software Engineering (ed. John Marciniak) Wiley & Sons, Inc., (1994) 2
58. McLean, J.: A Comment on the 'Basic Security Theorem' of Bell and LaPadula, Information Processing Letters, vol. 20, no. 2, Feb. (1985) 11
59. McLean, J.: A General Theory of Composition for Trace Sets Closed Under Selective Interleaving Functions, Proceedings of 1994 IEEE Symposium on Research in Security and Privacy, IEEE Press, (1994) 16, 29
60. Menezes, A. J. et al: Handbook of Applied Cryptography. CRC Press (1996)
61. Milner, R.: A Calculus of Communicating Systems. Springer, LNCS 92, (1980) 17
62. Milner, R.: Communication and Concurrency, Prentice-Hall (1989)
63. Milner, R. et al: A calculus of Mobile Processes, I and II. Information and Compution, 100: (1992) 1-77 17
64. Milner, R.: Communicating and Mobile Systems: the Pi-Calculus, CUP (1999) 56
65. Moskowitz, I. and Costich, O.: A classical automata approach to noninterference type problems Proceedings of the Computer Security Foundations Workshop V, (1992) 11
66. O'Halloran, C.: A Calculus of Information Flow, Proceedings of ESORICS (1990) 16
67. Pfitzmann, B. et al: Crptographic security of reactive systems, ENTCS, 32 (2000)
68. Pinsky, S.: Absorbing covers and intransitive non-interference, IEEE Symposium on Research in Security and Privacy (1995) 39
69. Pinsky, S. and Ziegler, E.: Noninterference Equations for Nondeterministic Systems, Proceedings of the Computer Security Foundations Workshop, (2001) 53
70. Reed, M. and Roscoe, A. W.: A Timed Model for Communicating Sequential Processes. In proceedings of the 13th ICALP, LNCS 226, (1986) 17
71. Roscoe, A. W. et al: Non-interference through determinism, Proceedings of ESORICS (1994) 40
72. Roscoe, A. W.: CSP and determinism in security modelling, in proceedings of the IEEE Symposium on Security and Privacy, IEEE Computer Society Press, (1995) 27
73. Roscoe, A. W.: The theory and practice of concurrency, Prentice-Hall (1997) 17, 21, 25, 30, 50
74. Rushby, J.: Noninterference, Transitiivity and Channel-Control Security Policies, SRI Tech Report (1992) 16, 50
75. Ryan, P. Y. A.: A CSP formulation of non-interference and unwinding, Presented at CSFW 1990 and published in Cipher, Winter 1990/1991 26, 30, 31
76. Ryan, P. Y. A. and Sennett C. T. eds: Formal Methods in Systems Engineering Springer Verlag (1993) 4
77. Ryan, P. Y. A. and Schneider, S. A.: Process Algebra and Non-interference, JCS Vol 9, nos 1,2, (2001) 75-103 45
78. P. Y. A. Ryan et al: Modelling and Analysis of Security Protocols, Pearson (2001) 51, 55, 57
79. Sabelfeld, A. and Sands, D.: Probabilistic Non-interference for Multi-threaded Programs. In Proceedings of the IEEE Computer Security Foundations Workshop, Cambridge, July 3-5 2000, IEEE Computer Society, (2000) 200-215 53

80. Saidi, H.: Model Checking Guided Abstraction and Analysis, Proc of the 7th International Static Analysis Symposium (2000) 55
81. Sandhu, R. S.: Lattice Based Access control Models, IEEE Computer, volume 26, number 11, November (1993) 9-19 9
82. Schneider, S. A.: Concurrent and Real time systems: the CSP approach, Wiley (1999) 17, 21
83. Schneider, S. A.: Testing and abstraction, Royal Holloway, University of London Tech Report tr-99-02 (1999) 46
84. Schneider, S. A.: May Testing, Non-interference and Compositionality, Royal Holloway Tech report CSD-TR-00-02, January 2001. 40, 43, 44
85. Schneider, S. A. and Sidiropoulos, A.: CSP and anonymity, Proceedings of ESORICS (2000) 52
86. Shockley, W. R.: Implementing the Clark Wilson Integrity Policy Using Current Technology, in Proceedings of the National Security Conference, (1988) 29-36 10
87. Simpson, A. C.: Safety Through Security, DPhil thesis, Oxford University (1996) 50
88. Sutherland, D.: A model of information, 9th National Computer Security Conference (1986) 16, 44
89. Weber, D.: Specifications for Fault Tolerance. ORA report (1988) 19-3 50
90. Wittbold, J. T. and Johnson, D. M.: Information flow in nondeterministic systems, Proceedings of the Symposium on Research on Security and Privacy (1990) 16, 45
91. Volpano, D and Smith G.: Probablilistic non-interference in a concurrent language. Journal of Computer Security, 7(2, 3): November (1999) 231-253 56