Non-Interference Properties for Nondeterministic Processes

Richard Forster

Trinity College

Thesis submitted for the degree of
Doctor of Philosophy at the University of Oxford

1999

Non-Interference Properties for Nondeterministic Processes

Abstract

Non-interference properties are concerned with the security of systems; given a process that has multiple users interacting with it, we are interested in preventing unspecified information from flowing between the users. This thesis is primarily concerned with asymmetric conditions that, given a process P which has a high-level user and a low-level user interacting with it, ensure no information flows from the high-level user to the low-level user.

For processes that behave deterministically, conditions have been stated which we believe correctly identify secure processes and a variety of different authors' specifications can be shown to be equivalent. For nondeterministic processes, whilst there are many conditions, there is no such agreement. It is frequently possible to find counterexamples to published properties, several of which can be found herein.

In this thesis information flow is considered by using formal methods, specifically CSP and transition systems. The differences between deterministic and nondeterministic processes are discussed, and the corresponding problems, as they relate to non-interference, are highlighted. Several classes of security condition are then investigated, compared and dissected: those based on determinism and the paradigms of non-interference and non-deducibility. For each of the existing conditions studied, examples are given where they incorrectly identify insecure processes as secure.

The new family of *local non-interference* conditions is then introduced. These conditions require all high-level transitions to take a process to a state that is indistinguishable, to the low-level user, to the preceding state. Their success in correctly identifying previously troublesome processes is illustrated and numerous properties are shown. In particular, the interrelationships of these conditions are given as well as their relationship to the existing conditions. Additional advanced properties are shown relating to the compositional development of secure processes.

Acknowledgements

This work was supervised by Bill Roscoe and many thanks are due to him for his work. Additional thanks are due to Philippa Broadfoot, Sue Older, Bryan Scattergood and Lars Wulf. This work was funded by the EPSRC (Award: 95306887) and the US Office of Naval Research (Contract: N00014-99-1-0381/A00001).

I would like to thank the staff and fellows of Trinity College for making Oxford a very pleasant place to live and work for seven years, especially: Russell Allen, Katie Andrews, Alan Milner, Marilyn Oakey, Valerie Parslow and Chris Prior.

I would like to thank my friends from the past few years for some of life's many distractions, a very incomplete list being: Naomi Alderman, Ira Bing, Mark Braganza, David Chambers, Guy Coates, Deborah Cooper, Nina Cornish, Rosalie Eisenberg, Catherine Emerson, Dan Hepner, David Hoffman, Matt & Mel Green, Ralph Grimble, Elen Humphreys, Fran Lowery and Colin Turnbull. Apologies to everyone not mentioned — I hope it does not put you off reading the thesis.

Fascinating Aïda for the laughter.

The **British Informatics Olympiad** is the national computer programming competition for secondary school students. I have received a great deal of pleasure through the running of this competition and my involvement with the **International Olympiad in Informatics**. A great deal of thanks are due to all my friends and colleagues in these events, in particular: Antony Rix, Fieke Dekkers, Harmke de Groot, Keith & Ruth Guthrie, Rob Kolstad and Eljakim Schrijvers.

Dedicated to my family

Contents

Notation

LTS	labelled transition system	2.1.1	11
	weak autobisimulation with respect to H	2.3.2	20
	weak bisimulation	2.2.6	16
	with respect to H	2.3.1	20
$Det(P)$	deterministic	3.1.1	23
$L\text{-}Det(P)$	L-deterministic	3.1.2	25
$L\text{-}Ind(P)$	independent	3.1.5	26
$LowDet_L(P)$	low-deterministic	3.1.8	28
$\mathcal{E}$	eager abstraction	3.2.1	30
$\mathcal{I}$	interleaving abstraction	3.2.14	37
$\mathcal{L}^{\mathcal{F}}$	lazy abstraction (over $\mathcal{F}$)	3.2.7	33
$\mathcal{L}^{\mathcal{N}}$	lazy abstraction (over $\mathcal{N}$)	3.2.11	35
$\mathcal{M}$	mixed abstraction	3.2.19	41
@	projection	3.2.10	35
BNDC	bisimulation non-deducibility on compositions	4.3.2	72
BSNNI	bisimulation strong nondeterministic non-interference	4.3.6	74
	failures-invariant	4.2.12	68
LNI	local non-interference	5.2.10	97
NDC	non-deducibility on compositions	4.3.1	71
NNI	nondeterministic non-interference	4.2.3	64
	non-deducible	4.1.4	52
	non-deducible on strategies	4.1.10	58
SBSNNI	strong bisimulation strong nondeterministic non-interference	4.3.7	75
SLNI	strong local non-interference	5.2.1	92
SNNI	strong nondeterministic non-interference	4.2.7	66
	trace-invariant	4.2.10	67

Chapter 1

Introduction

Imagine the following situation:

A high-level user and a low-level user each have a file-system (F_H and F_L) on a common disk. These file-systems are independent in that they are disjoint, neither user can interact with the other user's file-system and the contents of one system cannot effect the other; for example, the ability to create a file on system F_L is only dependent on the contents of F_L and not on any property of F_H such as the names of F_H's files or their total size. Only the high-level user can access F_H and similarly for F_L. Access to the disk is through a simple 'block at a time' disk drive controller which operates on the requests for disk blocks as they come and finishes one task before moving on to the next. If there is no mechanism for communicating the data other than to the correct user, the system allocates resources 'fairly' to the users and we ignore timing, this system is secure[1]; it does not allow the low-level user to get any high-level information.

Since requests are processed linearly, if the requested blocks are scattered the system may not be overly efficient. In order to try and make the system more efficient, the disk drive controller is replaced by one that optimises its seek operations by using the *elevator algorithm*. Such a controller keeps a list of requested blocks and keeps the disk-head seeking in the same direction

[1] Examples of the development of secure multi-level file-systems can be found in [Wul97] and [RWW94].

until there are no additional requests in that direction, it then switches direction. For example, if there are requests for blocks $10, 20, 30$ and 40, and the drive-head has just read 25 and is moving 'up', the remaining blocks will be read in the order $30, 40, 20$ and 10.

The system modified to use the elevator algorithm is no longer secure. Suppose F_L statically covers blocks 16–34 and F_H statically covers blocks 5–15 and 35–45[2]. A co-operating high-level user and low-level user could adopt the following strategy:

1. Both users let any outstanding blocks be processed.
2. The high-level user puts in a request for either 10 or 40 (for communicating a 0 or 1 bit respectively).
3. The low-level user puts in a request for blocks 20 and 30.

If the high-level user requested block 10 then the low-level user will receive block 20 before block 30; conversely if the request was for block 40. Only one bit of information has been transmitted, but this protocol could be repeated many times. The potential for covert channels from disk controller optimisation is discussed further in [Wra91] and [KW91]; they cite [Sch77] as first identifying the problem. The bandwidth of this particular channel, with 1991 technology, was estimated [KW91] as being around 100bps.

It is not easy to prevent covert channels from existing. Looking back at the description of the system in the first paragraph, there was a complicated framework in place to prevent any information flow. If some of those restrictions are removed covert channels reappear. For example, if sizes of F_H and F_L can grow without limit the high-level user might fill up the disk and thus prevent the low-level user from creating a new file; each time the low-level user tries to create a file, one bit of information (disk full / not full) can be transmitted.

The given problem arose because one of the components of the system, the disk controller, was not secure[3]. It is not too surprising that such insecure components can lead to the overall system being insecure. Such

[2]If the idea of a file-system being disjoint across the disk is problematic we could build a similar example, taking a single, secure file-system with high-level and low-level files. In this case the high-level user has files on either side of the low-level user's files.

problems might be avoided by adding additional machinery to the component to make it secure or only using secure components to begin with. In the latter case it is necessary to show how and if such pieces can be fitted together to make a secure system. Alternatively, we might be able to show that the problems in one part of the system cannot affect the overall system security. For all of these cases we need to be able to show that such a system is secure. In order to do that it is first necessary to decide what we mean by 'secure'.

1.1 Problem Statement

The problem we are interested in is one of *information flow*, or rather preventing the unauthorised disclosure of information. Stated informally

> Given a system which has multiple users interacting with it, we want to ensure that no information can flow between certain sets of users.

As well as preventing the users from directly communicating through *overt channels*, such as sending messages, or deliberate indirect communication, such as creating a file containing a message, we need to prevent information flowing through *covert channels*. A covert channel is when some part of the system is used in an unexpected manner to allow the transfer of information. These channels of communication may be accidental as well as non-obvious. They may arise from an apparently unrelated sequence of events or a contention for resources, and they might manifest themselves long after the events that caused them to occur.[4]

There are several issues hidden within the basic statement of the problem. The first of these is fundamental to the meaning of the problem. There is more freedom in the remaining issues; alternate answers and interpretations can lead to other legitimate approaches.

[3] This particular example can be a genuine issue since optimisations in disk controllers are often embedded and beyond the control of the operating system.

[4] Traditionally such channels have been split into *storage* and *timing* channels. The distinction can be hard to make (especially when formally modelling systems with process algebras or LTSs) and channels can have both characteristics. This is true in both timed and untimed algebras.

1. What do we mean by information flow?
2. How do the users observe the system?
3. What is a system and how are we modelling it?
4. What properties does the information flow relationship have?
5. What type of users are we concerned about? (e.g. is there any co-operation?)
6. How many users are interacting with the system?

The most difficult is probably what do we mean by 'information flow'? It is said that the difficulty in making something foolproof is that fools are so ingenious. The difficulty in defining information flow is that new examples keep appearing which current definitions say have no undesirable information flow, yet convincing arguments can be given for why there is such a flow of information. Such flows of information can be extremely subtle.

The two main paradigms for studying information flow, as distinct from security policies to prevent such flows, are *non-interference* ([GM82] and [GM84]) and *non-deducibility* ([Sut86] and [WJ90]); these are discussed in detail in Section 4.2 and Section 4.1 respectively. Suppose we are trying to prevent any information from flowing between a high-level user and a low-level user. If the actions of the high-level user do not affect what the low-level user can see the non-interference condition will be satisfied. If the low-level user is unable to deduce anything about the actions of the high-level user the system will satisfy the non-deducibility condition. The difficulty in expressing these notions formally, especially for nondeterministic systems, has led to a variety of conditions; some of these are discussed in the relevant sections.

The essential ideal of non-interference is what we are aiming for since we are trying to prevent the low-level user from gaining any high-level information. When the high-level user interacts with the system the low-level user, whilst unable to observe the high-level action, may be able to observe the effect of such an action; for example, different events may become available to the low-level user, either immediately or at a later stage. The question becomes how do we distinguish between the various characterisations of the system and when do we consider them equivalent? As pertinent are the questions: which characterisations do we need to check as being equivalent and what characterisations are we considering? For deterministic systems many different solutions appear acceptable; equivalences collapse for such processes and many of the conditions which answer these questions differently, classify the same processes as secure.

In a nondeterministic system these questions become increasingly difficult, and this is reflected by the large number of different conditions (which no longer coincide). The effects of a high-level action might equally be brought about as the result of internal nondeterministic choices. Has the high-level user affected what the low-level user can see if the same result might have come from an internal nondeterminism? Some non-interference conditions say no and do not differentiate fully between these internal choices and the high-level events. Nevertheless it turns out that it is not necessarily sufficient to equate two systems just because the results of high-level actions might have come from internal choices. The important point is that it is not always necessary for the low-level user to be able to make any deductions about the high-level user's actions in order for the high-level user to pass information to the low-level user. (This will been demonstrated later in several cases such as Example 4.1.8, Theorem 4.3.11 and Example 5.1.3.)

One issue that we are not concerning ourselves with is the bandwidth of any communication channels. We have assumed that any information flow is undesirable and a system that might exhibit any should be classified as insecure. Certain levels of information flow might be acceptable for certain systems. Alternatively, under certain modelling assumptions which are not dealt with in this thesis, such as probabilistic nondeterminism, the flow of information through a covert channel might be judged as being insufficiently likely. Discussions on the capacity of covert channels can be found, for example, in [Mill87] and [TG88].

The information flow we are trying to prevent may be symmetric or asymmetric. In other words, we might want no information to flow in either direction between two specified users, or only require information flow to be prevented in one direction. In a general multi-user system we usually consider users as having a given security level[5]. If a higher security level indicates access to information that is more confidential, we are interested in preventing this information from reaching a user with a lower security level. We are not concerned with low-level information being available to a high-level user. In other words, we are concerned with the asymmetric flow of information from high to low.

The advantage of an asymmetric flow condition is that it is more general than the corresponding symmetric flow condition. If we require no information to flow in either direction we can check that our asymmetric security condition holds in both directions by reversing the roles of high-level user and low-level user. Not all of the security conditions in the literature are asymmetric, for example some formulations of non-deducibility such as those in Definition 4.1.4 and Definition 4.1.10, are symmetric. Asymmetric versions of such conditions have generally been given at a later date by different authors.

Given an information flow relationship between users we can ask whether it is transitive or not. This seems a reasonable assumption to make; if A can communicate with B and B can communicate with C then A might pass information on to C via B. Nevertheless it can be reasonable to allow for an intransitive interference relationship if, for example, there are users who can downgrade information or be trusted not to pass on undesirable information. Certainly many of the original security conditions allowed intransitive relationships. One of the problems with allowing such relationships is that they can obfuscate some covert channels that we are trying to prevent, for example we might wish to prevent a direct channel from A to C. Further details on intransitive relationships and their potential problems can be found in [Pin95] and [RG99].

Given a transitive information flow relationship we can simplify the problem to one that only includes two users. Systems that have a larger number of users can be dealt with by considering alternate partitionings of the users. In other words, we can group users together into two meta-users and ensure our security condition holds for these users. For each user U in a hierarchy of users we will test our condition; the partitioning used is U together with the other users U is permitted communication with, against the users with whom U is not allowed communication. We can hence use a condition that solves the problem for two users to solve the information flow problem with more than two users. From hereon in we will assume we are dealing with two users: a high-level user and a low-level user.

[5] Often referred to as multi-level security (MLS).

Information flow might occur because the two users are manipulating the system in an attempt to communicate; the high-level user acting as a 'mole' to deliberately pass information to the low-level user. Alternatively it might also occur because the low-level user is acting as a 'spy' and trying to gain information about/from the high-level user without their assistance. We need to prevent both types of communication. These can be treated as the same problem — that no undesirable information flow can ever arise from any operation of the system. If the high-level user cannot influence the low-level user neither type of covert channel can exist. In a system which does not satisfy this condition it may be desirable to distinguish between the two cases, however the objective of the information flow properties considered in this thesis is to prove that no such influence exists.

It was mentioned earlier that non-interference depends on distinguishing between different system characterisations. The characterisations that we need to check are dependent on the two unanswered issues: how do the users observe the system, and what is a system and how are we modelling it? A variety of formal notations and algebras have been considered in the literature; in particular there are those based on behavioural models and transition systems.

What the user observes about a system is dependent on how it is being modelled, similarly for the equivalence being used. For example, in a behavioural model such as CSP we are usually concerned with the denotational semantics, whereas a transition system approach has an operational basis. The choice of model has varied widely; some of the pros and cons are discussed later and, for example, in [Ros95a] and [RRF99]. The modelling choice does not just vary the type of characterisations, it can also effect the amount of information available in them. For example, whether there is a distinction between inputs and outputs, the interpretation of any nondeterminism in the model and whether there is any timing information. These choices are discussed in Section 3.3.

In this thesis there are two aspects of the system that might provide information to the users: the states the system can be in and the available events. If we have a process P that we want to test for security (i.e. without undesirable information flow), we can partition its alphabet into two disjoint subsets H and L. These subsets represent the events, both inputs and outputs, which are high-level and low-level respectively. They are disjoint because we are trying to prevent a user who can only access low-level information from observing any high-level events. The only other events are internal system events which are treated in a uniform way as the event τ.

We want to give the low-level user as great an opportunity as possible to detect any covert channels in the system. The low-level user is able to communicate and observe the system through the events in L, the actual mechanism being dependent on the model. The events in H are clearly to be excluded because this is the information we are trying to prevent being observed. In models with state information the low-level user must be unaware of the states the system is passing through because a high-level event may alter the current state.

The situation for the high-level user is different; we want to provide as much opportunity as possible for creating any covert channels. The user can communicate and observe the system through the events in H. Since we are interested in an asymmetric condition the high-level user should be allowed to observe any of the events in L. We might also choose to allow the high-level user power beyond any system equivalence, for example by allowing the observation of the current system state and how internal nondeterministic choices are resolved.

It is also reasonable to assume that both users have a full description of the system available. This might be the transition system or knowledge of all the potential behaviours of the system. Such information gives both users added opportunities for exploiting covert channels.

We can combine several of the previous points to give a new informal problem statement:

> Given a system P which has two users interacting with it, a high-level user and a low-level user, we want to ensure that no information can flow from the high-level user to the low-level user.

We will always assume that the alphabet of this process has been partitioned into the disjoint subsets H and L as described above. In other words $\Sigma = H \cup L$. The availability of an additional internal system event is denoted by $\Sigma^\tau = \Sigma \cup \{\tau\}$.

1.2 Thesis Overview

There are three parts to this thesis which are all interwoven. Firstly there are the problems with judging information flow in nondeterministic systems, secondly some of the existing security conditions are discussed and contrasted, and finally a new family of conditions is introduced and a variety of properties proved. It is the difficulties with nondeterministic systems which form the motivating factor behind the work. Determinism based conditions will be introduced which we believe correctly identify secure processes for a large class of processes but are not always sufficient when the low-level view of the system is nondeterministic. A number of well known conditions are intended to correctly identify such processes and so they are introduced and their development and failings discussed. New conditions are then proposed to overcome some of these difficulties.

Formal methods are used throughout the work for studying security conditions and processes. Chapter 2 presents some of the machinery necessary for the rest of the thesis. Labelled transition systems, which are the primary form for processes in this work, are introduced along with an associated equivalence relationship — weak bisimulation. Some of the important characteristics of CSP used herein are also discussed.

Chapter 3 discusses determinism and more especially low determinism, where the system appears deterministic to the low-level user. Process abstractions and the determinism based conditions are then introduced, in particular the lazy security condition. It is argued that this is the correct definition for deterministic processes but not for nondeterministic ones. Some of the difficulties for nondeterministic processes are then looked at and their absence for deterministic processes highlighted.

Chapter 4 introduces several conditions for nondeterministic processes. Non-deducibility requires the low-level user to be unable to make deductions about the high-level user's behaviour. Non-interference requires the high-level user is unable to affect the low-level user's environment. Each of these conditions has been extended to overcome various problems that have been identified and, for non-interference, to cover nondeterministic systems. Some of these extensions are considered and further conditions, that are based on extensions to both non-deducibility and non-interference, are then discussed. The contribution of this thesis is to their classification and comparison; in particular a number of examples are given which illustrate that the published conditions classify some (at best) questionable processes as secure.

Chapters 5 and 6 introduce a new family of security conditions and discuss various results. Local non-interference requires H labelled transitions to take a process to a state that is indistinguishable, to the low-level user, from the preceding state. Some of the more basic results are given in Chapter 5 where the conditions' interrelationships are shown. Chapter 6 demonstrates that, for the processes we believe the lazy security condition correctly identifies, the new conditions are in agreement. Several laws for constructing secure processes are then given and the implementation of the conditions is discussed.

Finally, Chapter 7 presents some basic conclusions.

Notes

The symbol § can be found before some of the theorems, propositions and corollaries in the following chapters. This indicates that the following item has been taken directly from the literature.

When CSP or CCS is used to describe processes it should be considered as a literal translation of the transition system, i.e. we are using their operational semantics.

Chapter 2

Machinery

2.1 Transition Systems

Processes in this thesis are usually modelled as *labelled transition systems* (LTS). An introduction to these can be found in [Arn94] and [Ros98, Section 7.2]. A transition system consists of a set of possible states which a process can be in and the potential transitions between these states. The transitions within the system are labelled by the action/event which triggers the transition. In other words, an LTS is a directed graph where the nodes are process states, the edges are transitions and the edges are labelled. More formally:

Definition 2.1.1 A *labelled transition system* (LTS), over an alphabet Σ, is a 5-tuple $\mathcal{A} = \langle S, T, \alpha, \beta, \lambda \rangle$ where

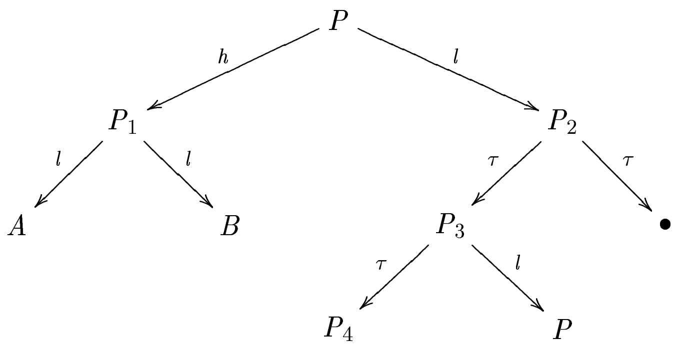
- S is a finite or infinite set of *states*,
- T is a finite or infinite set of *transitions*,
- α and β are two mappings from T to S which take each transition $t \in T$ to the two states $\alpha(t)$ and $\beta(t)$, respectively the *source* and *target* of the transition t,
- λ is a mapping from T to Σ^τ taking each transition t to its label $\lambda(t)$.

It often makes sense to consider the initial state of a transition system; all LTSs that represent processes have an initial state. If P is a state in a transition system we can refer to the process P, meaning the transition system where P is the initial state.

Operationally an LTS is always in a single state $\sigma \in S$ and it can move to a new state $\sigma' \in S$ if there exists a $t \in T$ such that $\alpha(t) = \sigma$ and $\beta(t) = \sigma'$. Since the LTSs represent processes, changing states by transition t means the event $\lambda(t)$ is taking place. If $\lambda(t) = \tau$ this event is an internal event, otherwise it is an external, visible event. States with τ labelled transitions are *unstable* and some event (not necessarily a τ labelled one) must occur within a short time. States without τ labelled transitions are *stable* states. If a state has visible events the environment has the choice of which of the labels to choose; if there are several states with the same label the environment does not choose which of them is selected.

Labelled transition systems will often be illustrated in the text by diagrams. States are either shown as a $\bullet$ or by an identifier for specifically named states. Two states with the same (non-blank) identifier indicate the same state, whereas all unidentified states should be treated as distinct. Transitions between states are shown as lines. Arrows on the transition lines point from the source (α mapping) to the target (β mapping) of the transition. Transitions are also marked by the label (λ) of their transition since event names are important.

Example 2.1.2 Consider the following LTS with initial state P:



There are eight states in this LTS. In state P the environment has the choice of an h and an l event and nothing will occur until one of these is chosen; an h event will move the process to state P_1 and similarly an l event will move it to state P_2. State P_1 has two identically labelled l events and once the environment has chosen to communicate an l, a nondeterministic choice is made as to whether the process enters state A or B. State P_2 is unstable since it has two τ events and so the process will enter either P_3 or the unidentified state (nondeterministically) without the environment. State P_3 is also unstable and may give the environment the opportunity to communicate an l and let the system return to the initial state, or may choose to enter state P_4. State P_4 and the unidentified state have no transitions and so the process cannot change state again.

(End of Example)

The notion of a transition between states can be extended to that of a *path*:

Definition 2.1.3 A *path* (of length n) in an LTS is a sequence of transitions $t_1, \ldots, t_n$ such that

$$\forall\, i : 1 \leqslant i < n \bullet \beta(t_i) = \alpha(t_{i+1})$$

There are several shorthand notations for denoting transitions. $P \xrightarrow{\mu} P'$ means we can get from state P to state P' by a transition labelled with μ. $P \overset{\mu}{\Longrightarrow} P'$ is a shorthand notation for $P\ (\xrightarrow{\tau})^* P_1 \xrightarrow{\mu} P_2\ (\xrightarrow{\tau})^* P'$, where $(\xrightarrow{\tau})^*$ stands for a path of τ labelled transitions which may be empty. The notation $P \overset{\mu}{\not\Longrightarrow}$ means $\nexists\, P'$ such that $P \overset{\mu}{\Longrightarrow} P'$, similarly for $P \overset{\mu}{\not\longrightarrow}$. $P \overset{\hat{\alpha}}{\Longrightarrow} P'$ stands for $P \overset{\alpha}{\Longrightarrow} P'$ if $\alpha \neq \tau$ and $P\ (\xrightarrow{\tau})^* P'$ if $\alpha = \tau$.

The $\Longrightarrow$ notation is also extended to paths; if $\delta = \delta_1 \delta_2 \ldots \delta_n$ then $P \overset{\delta}{\Longrightarrow} P'$ means $\exists\, P_1, \ldots, P_{n-1}$ such that $P \overset{\delta_1}{\Longrightarrow} P_1 \overset{\delta_2}{\Longrightarrow} \cdots \overset{\delta_{n-1}}{\Longrightarrow} P_{n-1} \overset{\delta_n}{\Longrightarrow} P'$. It is distinguished by context.

2.2 Bisimulation

Bisimulation is a fundamental equivalence for transition systems. It is due to Milner [Miln80] and Park [Par81], and it is the equivalence used in CCS

[Miln89]. A *strong bisimulation* relationship between two systems is one which identifies states in the two systems that cannot be differentiated by an experimenter who can only distinguish between states by the set of transition labellings leading from them and deductions based on such information.

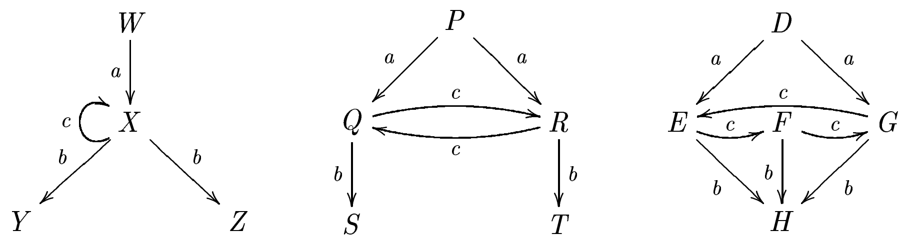
Definition 2.2.1 Let $\mathcal{A}_1 = \langle S_1, T_1, \alpha_1, \beta_1, \lambda_1 \rangle$ and $\mathcal{A}_2 = \langle S_2, T_2, \alpha_2, \beta_2, \lambda_2 \rangle$ be two labelled transition systems over the same alphabet. A binary relation $R \subseteq S_1 \times S_2$ is a *strong bisimulation* if $(P, Q) \in R$ implies, for all $\gamma \in \Sigma^\tau$,

(i) Whenever $P \xrightarrow{\gamma} P'$ then, for some Q', $Q \xrightarrow{\gamma} Q'$ and $(P', Q') \in R$
(ii) Whenever $Q \xrightarrow{\gamma} Q'$ then, for some P', $P \xrightarrow{\gamma} P'$ and $(P', Q') \in R$

There may be many relationships which satisfy this condition. For example, the empty relationship $R = \emptyset$ always does. We are really interested in whether two systems can be distinguished from each other by such an experimenter. Two systems are *strongly bisimilar* if there is a strong bisimulation between them in which their initial states are related.

Definition 2.2.2 Let $\mathcal{A}_1 = \langle S_1, T_1, \alpha_1, \beta_1, \lambda_1 \rangle$ and $\mathcal{A}_2 = \langle S_2, T_2, \alpha_2, \beta_2, \lambda_2 \rangle$ be two labelled transition systems over the same alphabet. $\mathcal{A}_1$ and $\mathcal{A}_2$ are *strongly bisimilar*, written $\mathcal{A}_1 \approx \mathcal{A}_2$, if R is a strong bisimulation and $(s_1, s_2) \in R$ where s_1 and s_2 are the initial states of $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively.

Example 2.2.3 Consider the following LTSs:



All three transition systems are strongly bisimilar. The following strong bisimulation relationships exist:

$$
\begin{aligned}
R_1 &= \{(W,P),(X,Q),(X,R),(Y,S),(Y,T),(Z,S),(Z,T)\} \\
R_2 &= \{(P,D),(Q,E),(Q,F),(Q,G),(R,E),(R,F),(R,G),(S,H),(T,H)\}
\end{aligned}
$$

In all three transition systems W, P and D, a single a event can be communicated, followed by an arbitrary number of c events, followed by a single b event. No other events are available and so the systems cannot be distinguished.

If, for example, the transition $F \xrightarrow{b} H$ was removed, D would no longer be strongly bisimilar to W or P. Without this transition, D can get to a state where an additional c event is required before a b event can be communicated.

(End of Example)

It can be easily shown that the arbitrary union of strong bisimulation relationships between $\mathcal{A}_1$ and $\mathcal{A}_2$ is itself an bisimulation relationship. It therefore follows that the there exists a greatest bisimulation between any two transition systems.

§Theorem 2.2.4 Let $\mathcal{A}_1 = \langle S_1, T_1, \alpha_1, \beta_1, \lambda_1 \rangle$ and $\mathcal{A}_2 = \langle S_2, T_2, \alpha_2, \beta_2, \lambda_2 \rangle$ be two labelled transition systems over the same alphabet. Let

$$R = \bigcup \{S \mid S \text{ is a strong bisimulation}\}$$

R is the *greatest* (or *maximum*) strong bisimulation between $\mathcal{A}_1$ and $\mathcal{A}_2$. Furthermore, R is an equivalence relation.

A bisimulation relationship given between a labelled transition system $\mathcal{A}_1$ and itself is called an *autobisimulation*. Since this is a bisimulation relationship there exists a greatest autobisimulation R. This is of particular interest since another LTS can be built, with one state for each equivalence class in R, which is the smallest LTS to be bisimilar to $\mathcal{A}_1$.

Example 2.2.5 Let W, P and D be the labelled transition systems given in Example 2.2.3. Consider the LTS:

$$\begin{tikzcd} J \arrow[r, "a"] & K \arrow[r, "b"] \arrow[loop below, "c"] & L \end{tikzcd}$$

This transition system is formed from the equivalence classes in the greatest autobisimulation of W; alternatively, since P and D are strongly bisimilar to W, it is also the LTS formed from their greatest autobisimulations.

(End of Example)

Strong bisimulation requires every action, including τs, to be matched between LTSs. Since τ events are unobservable, extensions of bisimulation have been defined where they play a less prominent role. If two states are linked by a path containing at most a single non-τ event, i.e. $\sigma \stackrel{\hat{\alpha}}{\Longrightarrow} \sigma'$, an experimenter who only observes transitions will not distinguish this from a $\sigma \xrightarrow{\alpha} \sigma'$ transition. *Weak bisimulation* or *observational equivalence* requires all $\xrightarrow{\alpha}$ transitions to be matched by $\stackrel{\hat{\alpha}}{\Longrightarrow}$ paths.

Definition 2.2.6 Let $\mathcal{A}_1 = \langle S_1, T_1, \alpha_1, \beta_1, \lambda_1 \rangle$ and $\mathcal{A}_2 = \langle S_2, T_2, \alpha_2, \beta_2, \lambda_2 \rangle$ be two labelled transition systems over the same alphabet. A binary relation $R \subseteq S_1 \times S_2$ is a *weak bisimulation* if $(P, Q) \in R$ implies, for all $\gamma \in \Sigma^\tau$,

(i) Whenever $P \xrightarrow{\gamma} P'$ then, for some Q', $Q \stackrel{\hat{\gamma}}{\Longrightarrow} Q'$ and $(P', Q') \in R$

(ii) Whenever $Q \xrightarrow{\gamma} Q'$ then, for some P', $P \stackrel{\hat{\gamma}}{\Longrightarrow} P'$ and $(P', Q') \in R$

Versions of Definition 2.2.2 and Theorem 2.2.4 can be demonstrated for weak bisimulation.

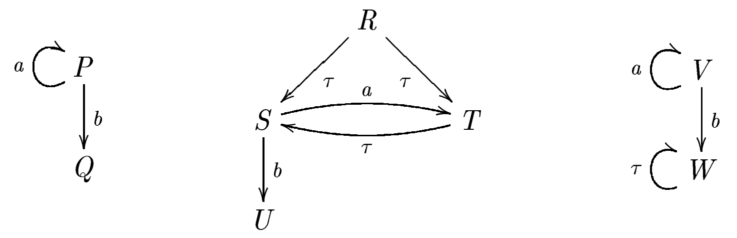
Definition 2.2.7 Let $\mathcal{A}_1 = \langle S_1, T_1, \alpha_1, \beta_1, \lambda_1 \rangle$ and $\mathcal{A}_2 = \langle S_2, T_2, \alpha_2, \beta_2, \lambda_2 \rangle$ be two labelled transition systems over the same alphabet. $\mathcal{A}_1$ and $\mathcal{A}_2$ are *weakly bisimilar* (or *observationally equivalent*), written $\mathcal{A}_1 \approx_B \mathcal{A}_2$, if R is a weak bisimulation and $(s_1, s_2) \in R$ where s_1 and s_2 are the initial states of $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively.

§Theorem 2.2.8 Let $\mathcal{A}_1 = \langle S_1, T_1, \alpha_1, \beta_1, \lambda_1 \rangle$ and $\mathcal{A}_2 = \langle S_2, T_2, \alpha_2, \beta_2, \lambda_2 \rangle$ be two labelled transition systems over the same alphabet. Let

$$R = \bigcup \{ S \mid S \text{ is a weak bisimulation} \}$$

R is the *greatest* (or *maximum*) weak bisimulation between $\mathcal{A}_1$ and $\mathcal{A}_2$. Furthermore R is an equivalence relation.

Example 2.2.9 Consider the following LTSs



All three transition systems are weakly bisimilar, but not strongly bisimilar. The following weak bisimulation relationships exist:

$$\begin{aligned} R_1 &= \{(P,R),(P,S),(P,T),(Q,U)\} \\ R_2 &= \{(P,V),(Q,W)\} \end{aligned}$$

(End of Example)

The two bisimulations are related. Strong bisimulation implies weak bisimulation, and weak bisimulation is the same as strong bisimulation when paths have been replaced by transitions.

§Theorem 2.2.10 Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two labelled transition systems over the same alphabet. Let $\mathcal{A}_1'$ be made by adding to $\mathcal{A}_1$ a transition for every $\stackrel{\hat{\mu}}{\Longrightarrow}$ path, labelled by μ; similarly for $\mathcal{A}_2'$. $\mathcal{A}_1$ and $\mathcal{A}_2$ are weakly bisimilar if and only if $\mathcal{A}_1'$ and $\mathcal{A}_2'$ are strongly bisimilar.

NB: The bisimulation used throughout the rest of this thesis is weak bisimulation, although the word 'weak' will be dropped.

2.3 CSP

Communicating Sequential Processes (CSP) is a process algebra due to C. A. R. Hoare [Hoa85], which provides a mathematical basis for modelling and analysing concurrent processes. CSP is used in several ways throughout this thesis: it is used to express many of the conditions and processes, and its notions of traces, failures and divergences are used throughout. The following is a brief overview of the more relevant items and the reader is expected to be familiar with the algebra. The best resource for the language is [Ros98].

Failures/Divergences Model

There are several denotational semantics for CSP based on different sets of behaviours. The standard model, failures/divergences, models the process P by the pair $(\mathit{failures}\,[\![P]\!], \mathit{divergences}\,[\![P]\!])$ where

1. $\mathit{failures}\,[\![P]\!] \subseteq \Sigma^* \times \mathbb{P}\,\Sigma$ are the stable failures (s, X) together with (t, X) where $t \in \mathit{divergences}\,[\![P]\!]$. A stable failure (s, X) consists of a *trace* s followed by a set X of actions that can be refused in some stable state after s.
2. $\mathit{divergences}\,[\![P]\!] \subseteq \Sigma^*$ is the extension-closed set of traces s after which the process P can perform an infinite unbroken sequence of τ events.

These sets satisfy the following axioms:

F1. The set of traces is non-empty and prefix-closed

$$\mathit{traces}\,[\![P]\!] = \{t \mid (t, X) \in \mathit{failures}\,[\![P]\!]\}$$

F2. Any subset of a refusal can also be refused

$$(s, X) \in \mathit{failures}\,[\![P]\!] \wedge Y \subseteq X \implies (s, Y) \in \mathit{failures}\,[\![P]\!]$$

F3. If X is a refusal after trace t and no event in Y can be performed then $X \cup Y$ is also a refusal after t

$$\begin{array}{ll} & (s, X) \in \mathit{failures}\,[\![P]\!] \wedge \forall\, a \in Y . s \frown \langle a \rangle \notin \mathit{traces}\,[\![P]\!] \\ \implies & \\ & (s, X \cup Y) \in \mathit{failures}\,[\![P]\!] \end{array}$$

D1. The set of divergences is suffix-closed

$$s \in \mathit{divergences}\,[\![P]\!] \cap \Sigma^* \wedge t \in \Sigma^* \implies s \frown t \in \mathit{divergences}\,[\![P]\!]$$

D2. Every divergent trace contributes arbitrary failures

$$s \in \mathit{divergences}\,[\![P]\!] \implies (s, X) \in \mathit{failures}\,[\![P]\!]$$

A process Q is said to refine P, written $P \sqsubseteq Q$, if the behaviours of Q are a subset of P's behaviours. More formally

$$P \sqsubseteq Q \Leftrightarrow \mathit{failures}\,[\![P]\!] \supseteq \mathit{failures}\,[\![Q]\!] \wedge \mathit{divergences}\,[\![P]\!] \supseteq \mathit{divergences}\,[\![Q]\!]$$

Two processes are equivalent, under the failures/divergences model, if they both refine the other.

Operational Semantics

The transition systems looked at in the following are specified using CSP, with a little notation borrowed from Milner's CCS [Miln89]: $a.P$ is equivalent to $a \rightarrow P$ and $+$ is the CCS choice operator. τ can be written explicitly in CCS and so this notation can be used for both internal and external choice. The standard operational semantics (see [Ros98, Chapter 7]) apply. In particular:

Timeout

The timeout operator, $P \triangleright Q$, allows the environment the opportunity to communicate the events in P, but if no such communication occurs it behaves like Q. τ events in P are promoted

$$\frac{P \xrightarrow{\tau} P'}{P \triangleright Q \xrightarrow{\tau} P' \triangleright Q}$$

Any visible event in P can be communicated

$$\frac{P \xrightarrow{a} P'}{P \triangleright Q \xrightarrow{a} P'} \quad (a \neq \tau)$$

Finally, if P does not communicate the process will become Q

$$\frac{}{P \triangleright Q \xrightarrow{\tau} Q}$$

Restriction

The restriction operator is taken from CCS although its notation has been changed to avoid conflict with the existing CSP operators. A process P with the H labelled events restricted, written $P \setminus_H$, can perform all the events of P provided they do not belong to H. It is equivalent to $P \underset{H}{\parallel} \mathit{Stop}_H$.

$$\frac{P \xrightarrow{a} P'}{P \setminus_H \xrightarrow{a} P' \setminus_H} \quad (a \notin H)$$

The restriction operator $\setminus_H$ can be extended over all the states of an LTS. In other words, we can remove all the H labelled transitions from a process, not just those that are necessary to prevent the process from communicating any H labelled events. This may create a disconnected LTS where some of the states are not reachable from the initial state.

Definition 2.3.1 Let $\mathcal{A}_1 = \langle S_1, T_1, \alpha_1, \beta_1, \lambda_1 \rangle$ be an LTS. The LTS of $\mathcal{A}_1$ *with respect to* H is $\langle S_1, T_1', \alpha_1, \beta_1, \lambda_1 \rangle$ where $t \in T_1'$ if and only if $t \in T_1$ and $\lambda_1(t) \notin H$.

Since a bisimulation relationship can exist between any LTSs there is no problem when the transition systems are disconnected. Calculating an auto-bisimulation of $\mathcal{A}_1$ with respect to H can also be done directly, without creating an intermediary transition system, by ignoring all H labelled transitions.

Definition 2.3.2 Let $\mathcal{A}_1 = \langle S_1, T_1, \alpha_1, \beta_1, \lambda_1 \rangle$ be an LTS. A binary relation $R \subseteq S_1 \times S_1$ is a *weak autobisimulation with respect to* H if $(P, Q) \in R$ implies, for all $\gamma \in \Sigma^\tau \setminus H$,

(i) Whenever $P \xrightarrow{\gamma} P'$ then, for some Q', $Q \overset{\hat{\gamma}}{\Longrightarrow} Q'$ and $(P', Q') \in R$

(ii) Whenever $Q \xrightarrow{\gamma} Q'$ then, for some P', $P \overset{\hat{\gamma}}{\Longrightarrow} P'$ and $(P', Q') \in R$

There is a corresponding link between related states in R and the related restricted states of $\mathcal{A}_1$.

Lemma 2.3.3 Let $\mathcal{A}_1 = \langle S_1, T_1, \alpha_1, \beta_1, \lambda_1 \rangle$ be an LTS.

(i) If R is a weak autobisimulation with respect to H and $(P, Q) \in R$, then $P \setminus_H \approx_B Q \setminus_H$.

(ii) If R is the maximal weak autobisimulation with respect to H and $P \setminus_H \approx_B Q \setminus_H$, then $(P, Q) \in R$.

Common Processes

There are three standard CSP processes which will we will use. *Stop* is the process that is unable to do anything. $Chaos_H$ is the most nondeterministic, non-divergence process that communicates in H. Run_H is a process that is always willing to communicate any event in H.

$$
\begin{aligned}
Run_H &= ?x : H \rightarrow Run_H \\
Chaos_H &= Stop \sqcap ?x : H \rightarrow Chaos_h
\end{aligned}
$$

Chapter 3

Deterministic Conditions

The security conditions for information flow can be split into two categories: those that restrict themselves to deterministic processes (or processes that appear deterministic to the low-level user) and those that encompass nondeterministic processes. In this chapter conditions in the former category are introduced and the differences between deterministic and nondeterministic processes are discussed.

Section 3.1 introduces determinism and highlights its use for studying information flow. Determinism is extended to just cover the low-level user and several types of low determinism, including one (independence) that is appropriate as a security condition, are discussed and compared.

Section 3.2 discusses process abstractions and looks at three determinism based security conditions: eager, lazy and interleaving. It is argued that, whilst the eager condition is too weak, the lazy condition provides a natural formulation of security for deterministic processes which is reinforced by its agreement with both independence and the interleaving condition.

There are nondeterministic processes that are secure and so Section 3.3 introduces various flavours of nondeterminism. Several difficulties with nondeterministic processes are discussed such as characterising equivalence, distinguishing some secure and insecure processes, what nondeterminism actually means and the implications of choosing a model on the assumptions made about the system's users.

3.1 Determinism

In a *deterministic* process the events that are available can be fully determined by the sequence of observable events that has already been communicated. If there is any uncertainty in which events will be available, due to there being different possible sets of available events, the process is *nondeterministic*. A nondeterminism may be introduced by internal events that are not observable, or external events that can lead to multiple states.

Determinism is easiest to state under a behavioural model because the existence of multiple or τ labelled transitions in an operation model does not necessarily imply nondeterminism. In other words, just because the current state of a transition system may be nondeterministic, it may still appear deterministic to its environment; algorithms for deciding determinism for finite state LTSs can be found, for example, in [Ros98, Section 9.1].

Under the failures/divergences model a deterministic system is not allowed to diverge or accept any events it may refuse.

Definition 3.1.1 A process P is deterministic if and only if it is divergence-free and

$$\forall\, s \in \Sigma^*, a \in \Sigma \bullet \neg((s, \{a\}) \in \mathit{failures}\,[\![P]\!] \wedge (s \frown \langle a \rangle, \emptyset) \in \mathit{failures}\,[\![P]\!])$$

There are several reasons why it is useful to consider deterministic processes from a security viewpoint. Determinism provides a natural formulation of security and deterministic processes possess several desirable properties. The determinism based conditions, originating in [RWW94] and [Ros95a], are based on the idea that if a user is presented with a deterministic process, since the available events are fully determined by the previous events, there cannot be interference by another user.

If the low-level user's view[1] of the system contains any nondeterminisms that are resolved in actuality by the high-level user, there is the potential for influencing the low-level user and hence the possibility of information flow. With just a low-level view of the system we may not be able to determine whether a given nondeterminism can be resolved by the high-level user or if it is internal to the system; this possibility for influence is removed if the low-level system view is deterministic. Furthermore, if the high-level user is able to influence the low-level user, we would expect this to cause a nondeterminism to be present in the low-level user's view of system; the contrapositive implies the absence of any high-level influence.

There are additional advantages to considering deterministic systems. Firstly we do not need to make unnecessary restrictions on how the users view the system; the low-level user might only have system traces available, but they might, for example, also record failures or use refusal testing. When a process is modelled there are corresponding assumptions made about how the users view the system. For deterministic systems the different (untimed) concurrency models collapse [Ros95a]. In other words, the same processes are identified as being deterministic. A user who is able to observe the process under a different model to the one we are considering will be unable to gain any additional information.

This property extends to conditions that are not based on determinism. In general, properties consider an equivalence over some function of the system, for example, between certain pairs of states or sets of behaviours. For a sufficiently deterministic system[2], if the condition holds, the process will be secure even if the users are observing the process under different models; when we look at such processes we are proving security against users with more power than we are modelling.

Dealing with deterministic properties limits the problems with the choice of model and equivalence. It also removes the need to decide how to interpret nondeterminism, since none is present in the model. The different views of nondeterminism have a bearing on how we need to model the system, what we can do with it and how the system is observed. For example, the deterministic process are maximal under CSP refinement; in other words, any implementation of a deterministic specification is semantically the same process, and we would expect security to be preserved. For nondeterministic systems this depends on when we believe nondeterminisms can be refined. This is discussed at length in Section 3.3.

[1] The appropriate model to use, and the corresponding abstraction, will vary with how the low-level user is able to observe the system. This is discussed in Section 3.2.

[2] I.e. if the necessary states or abstractions are deterministic.

Low Determinism

The deterministic notion of security that has been outlined (and will be discussed in Section 3.2) is based on extracting the low-level user's view of the system and then looking to see if it is deterministic. An alternative approach is to look directly at the process. A process P is deterministic if it is free of divergence and never has the opportunity to accept an event that it can refuse. This notion can be extended to one of *low determinism*. This is determinism from the low-level user's viewpoint; a low-level event cannot be both accepted and refused after a given trace.

Definition 3.1.2 Process P is *L-deterministic*, written $L\text{-}Det(P)$, if and only if it is non-divergent and

$$(s, \{a\}) \in \mathit{failures}\,[\![P]\!] \quad \Leftrightarrow \quad (P/s)^0 \cap \{a\} = \emptyset$$

for all $a \in L$ and traces s of P.

A process that is L-deterministic does not have to be deterministic, since a high-level event can be both accepted and refused. This corresponds to the intuition that a process can be secure whilst exhibiting nondeterminism to the high-level user, or communicating information from low to high. The converse, that any deterministic process is L-deterministic, holds as a direct implication of the definitions.

Lemma 3.1.3 If P is a non-divergent, deterministic process then, for all L, $L\text{-}Det(P)$ holds.

An L-deterministic process cannot exhibit any nondeterminism to the low-level due to any internal choices. Nevertheless, a high-level user can still introduce observable nondeterminisms into the process, since the low-level user is unable to distinguish between traces which only differ in their H labelled events.

Example 3.1.4 Let $L = \{l_1, l_2\}$ and $H = \{h\}$. Consider the process

$$\begin{aligned} P = \quad & l_1 \to \mathit{Stop} \\ \Box \quad & h \to l_1 \to l_2 \to \mathit{Stop} \end{aligned}$$

P is deterministic hence, by Lemma 3.1.3, $L\text{-}Det(P)$ is satisfied. Nevertheless, the low-level user will not view P deterministically. The low-level user is unable to distinguish between the trace $s = \langle l_1 \rangle$ and the trace $t = \langle h, l_1 \rangle$, however after s the event l_2 is refused, whereas after t the event l_2 can be accepted.

(End of Example)

A modified condition can be given that takes into account events that are outside the low-level user's alphabet. It is necessary to ensure that low-level user cannot accept and refuse an event after any two traces with the same low-level purge.

Definition 3.1.5 L is *independent* in process P, written $L\text{-}Ind(P)$, if and only if P is non-divergent and

$$(s, \{a\}) \in failures\, [\![P]\!] \quad \Leftrightarrow \quad (P/t)^0 \cap \{a\} = \emptyset$$

for all $a \in L$ and traces s, t of P with $s \upharpoonright L = t \upharpoonright L$.

Example 3.1.6 Let P be as defined in Example 3.1.4. $L\text{-}Ind(P)$ is not satisfied because $(\langle l_1 \rangle, \{l_2\}) \in failures\, [\![P]\!]$ and $(P/\langle h, l_1 \rangle)^0 = \{l_2\}$.

(End of Example)

If $L\text{-}Ind(P)$ holds the high-level user has no opportunity to affect the low-level user. If the low-level user could be affected there would have to be some trace with high-level events after which some L labelled event, which was available after a trace with the same low-level purge, was no longer available, or vice versa. Since independence does not allow the low-level user to observe any nondeterminism even when due to internal events (it implies L-determinism), the high-level user is also unable to select which of a set of behaviours the low-level user can observe; for every purged low-level trace there can only be one set of events available to the low-level user.

Independence can be used as a security condition. In other words, if L is independent in P, the process P does not let any information flow to a user who only communicates in L. The condition is limited to processes that present a deterministic interface to the low-level user and processes that only have nondeterminisms due to internal events will not be classed as secure. Independence has been shown to be equivalent to the lazy security condition which is discussed in the next section and we will concentrate on this latter condition.

Example 3.1.7 Let $L = \{l_1, l_2\}$ and $H = \{h_1, h_2\}$. Consider the process

$$\begin{array}{rcl} P = & & l_1 \to (l_1 \to P \,\Box\, l_2 \to P \,\Box\, h_1 \to P) \\ & \Box & l_2 \to (l_1 \to P \,\Box\, l_2 \to P \,\Box\, h_2 \to P) \end{array}$$

P is a process that always allows the low-level user to communicate either an l_1 or l_2 event. The available event for the high-level user, as well as whether one is available, is influenced by the low-level user. This process is secure since the high-level user cannot influence the low-level user. The process satisfies $L\text{-}Ind(P)$ since, after any traces s of P, l_1 and l_2 can be accepted but not refused.

If we reverse the roles of high-level and low-level user, independence no longer holds. The process appears nondeterministic to a user observing the system through the H labelled events since, for example, $(\langle\rangle, \{h_1\}) \in$ *failures* $[\![P]\!]$ but $(P/\langle l_1\rangle)^0 \cap \{h_1\} \neq \emptyset$. An arbitrary amount of information can be transmitted since every L labelled event alters which H labelled events are available.

(End of Example)

Determinism does not imply independence; H labelled events that are deterministically chosen and lead to deterministic states can make the system appear nondeterministic to the low-level user. This is demonstrated in Example 3.1.6. Conversely, independence does not imply determinism for the same reason as L-determinism; i.e. high-level events can still be accepted and refused. For example, consider any nondeterministic process that only communicates in H.

An alternative condition for low determinism is given in [Foc96]. This condition requires the process, before the high-level user acts, to be deterministic and places no restrictions on the process after an H labelled event. In other words, we are only interested in ensuring no event can be both accepted and refused after a low-level trace.

Definition 3.1.8 A process P is *low-deterministic* with respect to L, written $LowDet_L(P)$, if and only if $P \setminus_H$ is a non-divergent, deterministic process.

This definition is clearly much weaker than L-determinism and independence since it takes no account of the effects any H labelled event can have on the system. It is not proposed as a security condition, but it is useful because $P \setminus_H$ is a common term in several security conditions and when certain of these conditions are satisfied it becomes equivalent to the other low determinism definitions. It is therefore sufficient to show low-determinism to demonstrate either independence or determinism for such processes. This is not too surprising since any nondeterminism that is visible to the low-level user in a low-deterministic process must be the result of some high-level behaviour.

For a general process P:

Lemma 3.1.9 $L\text{-}Ind(P) \Rightarrow L\text{-}Det(P) \Rightarrow LowDet(P)$.

Proof

($L\text{-}Ind(P) \Rightarrow L\text{-}Det(P)$) This follows from Definitions 3.1.5 & 3.1.2, by considering the cases where $t = s$ in 3.1.5.

($L\text{-}Det(P) \Rightarrow LowDet(P)$) This follows from Definitions 3.1.2 & 3.1.8, by considering the cases where $s \in L^*$ in 3.1.2. □

Lemma 3.1.10 If P is a non-divergent, deterministic process then, for all L, $LowDet_L(P)$ holds.

Proof Follows directly from Lemma 3.1.3 and Lemma 3.1.9. □

3.2 Deterministic Abstractions

A *process abstraction* is a function that hides or removes some computational details of a process. In a process algebra context this might be done by the application of the algebra's constructs (e.g. hiding in CSP) or by directly manipulating the process' denotational semantics. Applying a process abstraction can allow us to remove details that are unimportant to the property of the process being considered or hide details that cannot be observed. Removing these details, which will not necessarily remove their potential effects, can make it easier to deal with the required properties.

In a security context an abstraction of the process might be the low-level or high-level user's view of the process. A high-level user might be aware of the low-level events that have taken place; potentially even how internal nondeterminisms have been resolved and which state a system is currently in. The low-level user is expected to be unaware of high-level events otherwise the only secure systems are those where there are no potential high-level events. It is not too surprising that abstractions that lead to low-level views of the system have an active role in many security conditions. The question is which abstractions to use and what should be done with them.

The use of CSP abstractions and determinism for looking at security and non-interference properties was first given by Roscoe, Woodcock and Wulf in [RWW94] and [Ros95a]. The work on these security conditions was further expanded in [RW95], [Wul97] and [Ros98]; more detailed discussions on the conditions, and abstractions in general, can be found in the latter two references.

CSP and (effectively) process abstractions had already been used to examine existing security conditions (e.g. [All91]) and give new ones (e.g. [Rya91] and [Jac90b]). The conditions in [RWW94] and [Ros95a] were the first conditions connecting non-interference with nondeterminism. Previous conditions tended to be based around process traces. A comparison of some of these results can be found in [Gra92].

Eager Abstraction

The first abstraction considered in [RWW94] and [Ros95a] is the hiding of high-level events. The low-level user is unable to observe any high-level events and hiding prevents these events from being observed. The resulting process will only contain low-level events and the high-level events will be indistinguishable from internal actions. Since the high-level events become

τ events, any state that was previously reachable, even if it required a high-level action to occur, is still available.

Definition 3.2.1 Given a process P the *eager* abstraction is

$$\mathcal{E}_H(P) \mathrel{\widehat{=}} P \setminus H$$

The corresponding security condition requires this abstraction to be deterministic.

Definition 3.2.2 A process P is said to be *eagerly secure* if $\mathcal{E}_H(P)$ is deterministic.

Several other CSP security conditions are given by using traces, including ones such as the *generalised non-interference* property of McCullough [McC88], that are equivalent to the eager security condition. In [Ros95a][3] a traces formulation is given that is equivalent to the eager security condition; if two traces of P are only different in their high-level events it is required that the subsequent behaviour of P after these traces appears the same to L. In other words:

§Proposition 3.2.3 For non-divergent $P \setminus H$, $\mathcal{E}_H(P)$ is deterministic if and only if

$$s \upharpoonright L = t \upharpoonright L \Rightarrow (P/s) \setminus H =_{\mathrm{FD}} (P/t) \setminus H$$

Hiding can introduce nondeterminism since it replaces external events with internal ones. If the communication of different H labelled events leads to different low-level events the resulting eager abstraction will not be deterministic, hence the process will not be eagerly secure. In other words, it is not necessary for the low-level user to observe a high-level choice to be affected by it.

Example 3.2.4 Let $L = \{l_1, l_2\}$ and $H = \{h_1, h_2\}$. Consider the processes

$$\begin{array}{rcl} P_1 & = & h_1 \rightarrow l_1 \rightarrow Stop \mathbin{\Box} l_2 \rightarrow Stop \\ P_2 & = & h_1 \rightarrow l_1 \rightarrow Stop \mathbin{\Box} h_2 \rightarrow l_2 \rightarrow Stop \end{array}$$

[3] The theorem is proved explicitly in the 1994 draft version of the paper.

Neither of these processes is secure. In P_1 the low-level user is only able to communicate an l_1 if an h_1 occurs, hence the low-level user can infer the occurrence of an h_1; if an l_2 is communicated the low-level user can infer the absence of an h_1 event. Similarly if an l_i can be communicated in P_2 the low-level user can deduce that the corresponding h_i event has taken place.

The eager abstractions of P_1 and P_2 are

$$\begin{aligned}\mathcal{E}_H(P_1) &= (l_2 \to \mathit{Stop}) \triangleright (l_1 \to \mathit{Stop}) \\ \mathcal{E}_H(P_2) &= l_1 \to \mathit{Stop} \sqcap l_2 \to \mathit{Stop}\end{aligned}$$

In other words, neither of the eager abstractions are deterministic and so neither process is eagerly secure.

(End of Example)

For the traces model the eager abstraction correctly represents the low-level user's view of the system. Hiding is not sufficient as an abstraction for more complex models because it does not deal sufficiently with refusals; by hiding high-level events the eager abstraction assumes a willingness on the behalf of the high-level user to always communicate. The abstraction is 'eager' because the environment is unable to prevent or delay the high-level events which are now internal τ events.

As well as being invisible to the low-level user the H labelled events are under the control of the high-level user. We cannot rely on the high-level user always being willing to communicate; the high-level user is free to choose whether to communicate as well as what to communicate. If the process before and after an H labelled event can be distinguished by the low-level user, they will be able to make inferences on whether the event has taken place or not.

Example 3.2.5 Let $L = \{l\}$ and $H = \{h\}$. Consider the processes

$$\begin{aligned}P_3 &= l \to P_3 \\ P_4 &= h \to l \to P_4\end{aligned}$$

P_3 is a secure process since it cannot be influenced by the high-level user. P_4 is not a secure process because the low-level user is only able to communicate an l event once the corresponding h event has taken place; the low-level user can deduce the number of h events that have taken place.

The eager abstractions of these two process are identical, i.e.

$$\mathcal{E}_H(P_3) = \mathcal{E}_H(P_4) = \mu X.\ l \rightarrow X$$

The abstraction is deterministic and so both processes are eagerly secure.

(End of Example)

Since eager abstraction assumes the high-level user is always willing to communicate it classes some insecure processes as secure. It is unable to make a fine enough distinction between processes, e.g. giving the same abstraction to both a secure and insecure process in Example 3.2.5. Two low-level system view abstractions should only be the same when the two systems are indistinguishable to the low-level user.

Hiding will also introduce divergence if an infinite sequence of H labelled events is possible in P. This can be viewed as a denial of service by the high-level user; communicating an infinite number of events and preventing the low-level user from communicating. Since deterministic processes are required to be divergence free such processes will not be eagerly secure.

Example 3.2.6 Let $L = \{l\}$ and $H = \{h\}$. Consider the processes

$$\begin{aligned} P_5 &= l \rightarrow P_5 \,\Box\, h \rightarrow P_5 \\ P_6 &= h \rightarrow P_6 \end{aligned}$$

The eager abstractions of these two processes are both divergent and hence nondeterministic. Whether P_5 is secure or not depends on whether we believe the high-level user can prevent the low-level user from communicating by performing an unbounded number of h events. P_6 is certainly secure; since the low-level user cannot communicate with this process it is not possible for any information to flow from high to low.

(End of Example)

The eager abstraction classes some secure processes as insecure (e.g. P_6). More seriously some insecure processes are classed as secure (e.g. P_4); in general the eager abstraction is not sufficient for a security condition. It is appropriate if high-level events can be assumed to occur without delay and the interpretation of unbounded high-level events is acceptable (or for processes without such infinite behaviour).

Lazy Abstraction

The eager abstraction of a process P, $\mathcal{E}_H(P)$, does not fully express the low-level view of the system. An abstraction that takes into account the fact that the high-level user can refuse events is necessary to give an accurate low-level description of the system. Such low-level system view abstractions are given in [Wul97] and [Ros98][4].

The low-level view of the system should be consistent with all possible high-level behaviours. The most nondeterministic, and hence most general process, that the high-level user can behave as is $Chaos_H$. Such a high-level user is able to accept or refuse any of the H labelled events available. A process taken in parallel with $Chaos_H$ will be able to exhibit all the different behaviours that the high-level user can cause. This suggests that the correct low-level abstraction is this parallel process with the high-level events hidden.

Definition 3.2.7 Given a process P the *lazy* abstraction (over $\mathcal{F}$) is

$$\mathcal{L}_H^{\mathcal{F}}(P) \mathrel{\widehat{=}} (P \underset{H}{\parallel} Chaos_H) \setminus H$$

The corresponding security condition requires this abstraction to be deterministic. As will be shown, the definition of lazy abstraction will change depending on the model being used. For processes that are divergence-free and finitely nondeterministic, it can be shown that the determinism of the abstractions are equivalent. The statement of a process being lazily secure is therefore given without reference to the model being used.

Definition 3.2.8 A process P is said to be *lazily secure* if $\mathcal{L}_H(P)$ is deterministic.

[4]The earlier papers ([RWW94] and [Ros95a]) use an obfuscation technique which is discussed in the 'Interleaving' section on page 37.

Every subset of H can now be refused before the high-level events are hidden. Processes whose low-level appearance differs depending on which events the high-level user is willing to accept will now appear nondeterministic; hence they will fail to be lazily secure.

Example 3.2.9 Let P_4 be as defined in Example 3.2.5, where it is argued as being insecure. The lazy abstraction is

$$\mathcal{L}_H^{\mathcal{F}}(P_4) = \mathit{Stop} \sqcap l \rightarrow \mathcal{L}_H^{\mathcal{F}}(P_4)$$

This is not deterministic and hence not lazily secure. The abstraction has highlighted the fact that, from the low-level user's viewpoint, the process can either deadlock (if the high-level user refuses to act) or accept an l event.

The eager abstraction of P_3 was the same as that for P_4. The lazy abstraction of P_3 is just $\mu\, X.l \rightarrow X$. In other words, the lazy abstractions of the two processes are not the same.

Process P_2 in Example 3.2.4 has a similar problem. It was shown as insecure by the eager abstraction because the L labelled event that occurs is dependent on the H labelled event that has preceded it. It is also insecure because there is only an L labelled event available if the high-level user has communicated. The lazy abstraction can deadlock after the empty trace.

(End of Example)

There is still a problem with the lazy abstraction because it is able to diverge whenever the system can accept an unbounded number of H labelled events. The failures/divergences model takes a catastrophic view of divergences, thus preventing the extraction of further information from the model. Furthermore, beyond the fact that we may not be interested in such denial of service attacks or believe in their validity, processes that are no longer able to communicate with the low-level user, such as P_6 in Example 3.2.6, are not allowed. We will usually want to assume that no such attacks will take place; that the high-level user will not communicate such an unbounded number of event or that the system will not permit these events without giving the low-level user the opportunity to communicate.

Definition 3.2.7 gives a lazy abstraction over the stable failures ($\mathcal{F}$) model. Divergence is not an issue over $\mathcal{F}$ since divergences are not recorded by the model. Under the failures/divergences model ($\mathcal{N}$) we can construct an abstraction that behaves in the same way, except it does not introduce any divergence if there is unbounded high-level behaviour. A *projection* operator is defined with the required effect.[5]

Definition 3.2.10 The projection of P onto alphabet L, written $P\,@L$, is defined

$$\begin{array}{rcl} \mathit{failures}\,[\![P\,@L]\!] & \mathrel{\widehat{=}} & \{\,(s \upharpoonright L, X) \mid (s, X \cap L) \in \mathit{failures}\,[\![P]\!]\,\} \cup \\ & & \{\,(s, X) \mid s \in \mathit{divergences}\,[\![P\,@L]\!]\,\} \\ \mathit{divergences}\,[\![P\,@L]\!] & \mathrel{\widehat{=}} & \{\,(s \upharpoonright L) \frown t \mid s \in \mathit{divergences}\,[\![P]\!] \wedge t \in L^*\} \end{array}$$

The only divergences in $P\,@L$ are related to divergences in P, as are the only traces to contain H labelled events. The lazy security condition only holds when the abstraction is deterministic and hence has no divergence; this requires P to be divergence-free. Under this condition $P\,@L$ reduces to a non-divergent process whose failures are

$$\{\,(s \upharpoonright L, X) \mid (s, X \cap L) \in \mathit{failures}\,[\![P]\!]\,\}$$

The process $P\,@L$ contains the low-level failures both before and after high-level events; in other words, it contains failures corresponding to the high-level user both refusing and accepting the H labelled events. The lazy abstraction for $\mathcal{N}$ is therefore defined to be the projection onto the low-level alphabet.

Definition 3.2.11 Given a process P, the *lazy* abstraction (over $\mathcal{N}$) is

$$\mathcal{L}_H^{\mathcal{N}}(P) \mathrel{\widehat{=}} P\,@(\Sigma \setminus H)$$

Example 3.2.12 Let P_5 be as defined in Example 3.2.6. The lazy abstraction of P_5 is

$$\mathcal{L}_H^{\mathcal{N}}(P_5) = l \rightarrow \mathcal{L}_H^{\mathcal{N}}(P_5)$$

[5] In the infinite traces/failures/divergences model ($\mathcal{U}$) the lazy abstraction can be given using CSP operators rather than direct manipulation of the failures and divergences. Under $\mathcal{U}$ we can give a 'fairness' process $\mathit{SemiFair}(L, H)$ that only allows infinitely many H labelled events to occur if infinitely many L labelled events occur. The lazy abstraction $\mathcal{L}_H^{\mathcal{U}}(P)$ is then $(P \underset{\Sigma}{\parallel} \mathit{SemiFair}(L, H)) \setminus H$. This process can only diverge if P does.

and so P_5 is lazily secure. This process is intuitively secure because an l event it always available to the low-level user, whether or not the high-level user is willing to communicate.

(End of Example)

Example 3.2.13 Let P_6 be as defined in Example 3.2.6. The lazy abstraction of P_6 is

$$\mathcal{L}_H^{\mathcal{N}}(P_6) = \mathit{Stop}$$

and so P_6 is lazily secure.

This example generalises to any non-divergent process that communicates only in H. In other words, if $\Sigma P = H$ and P is not divergent then $\mathcal{L}_H^{\mathcal{N}}(P) = \mathit{Stop}$ and so the process is lazily secure; a process that does not communicate in L cannot pass information to the low-level user.

(End of Example)

The corresponding situation to Example 3.2.13, when the process does not communicate in H, is not so simple. If $\Sigma P = L$ then $\mathcal{L}_H^{\mathcal{N}}(P) = P$; this process will only be lazily secure if P is deterministic. Nevertheless, since the process does not communicate in H, the high-level user is unable to pass information to the low-level user. This problem is unavoidable if our security condition relies on the determinism of an accurate low-level system view.

The lazy security condition satisfies several desirable security properties.

- It is shown in [RW95] that if P and Q are both lazily secure properties then their parallel composition, $P \; {}_{\Sigma P}\|_{\Sigma Q} \; Q$, is also lazily secure; the desirability of security properties being compositional is discussed in Section 5.1.
- The condition is preserved by CSP refinement; this follows since deterministic processes are maximal under refinement and $P \sqsubseteq Q \Rightarrow \mathcal{L}_H(P) \sqsubseteq \mathcal{L}_H(Q)$. In other words, a secure process cannot be refined by an insecure one. The preservation of refinement means, for example, that processes can be replaced by semantically equivalent ones, allowing the use of various compression techniques in automatic evaluation [RGG95].

- Lazy security is automatically checkable using tools such as FDR (see [FSEL], [Ros94] and [Sca98]); this tool has efficient decision procedures for determinism checking. For a divergence-free, finitely nondeterministic process the determinism of the lazy abstractions under different models is equivalent. $\mathcal{L}_H^{\mathcal{F}}(P)$ can be directly computed by FDR since it uses standard CSP operators.

- For deterministic processes the condition agrees with many other security processes. This is discussed in [Ros95a, section 7], [RW95] and highlighted in some of the following sections. It is shown explicity in [Foc96] for BNDC (see Section 4.3) and in Section 6.1 for the local non-interference conditions.

Interleaving Abstraction

The eager and lazy security conditions, considered in Sections 3.2.2 and 3.2.8 respectively, are based around the idea of concealing the high-level events in a process and producing a process that represents how the low-level user perceives the system. An alternative approach is to obscure the high-level events, so that an H labelled event cannot necessarily be connected to the high-level user. The original lazy security conditions, given in [RWW94] and [Ros95a], were based around obfuscating the high-level events by interleaving a process with Run_H. Applying the notation of [Wul97] :

Definition 3.2.14 Given a process P the *interleaving* abstraction is

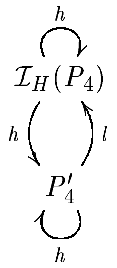
$$\mathcal{I}_H(P) \mathrel{\widehat{=}} P \mid\mid\mid Run_H$$

One immediate consequence of this definition is that the interleaving abstraction, unlike some of the previous abstractions, cannot introduce any divergence to a process.

The basis of the abstraction is that whenever an H labelled event might originate from P, the observed event might also have come from the interleaved Run_H. If P is a deterministic system and $\mathcal{I}_H(P)$ is nondeterministic, then there must be some state where the acceptance or refusal of an $h \in H$ labelled event makes the process distinguishable to the low-level user. The interleaving abstraction will be nondeterministic because following the h event from P leads to a new state, whereas following the h event from Run_H does not change the state; these two states, which are due to accepting and to refusing the event h in P respectively, are distinguishable.

Example 3.2.15 Let P_4 be as defined in Example 3.2.5. The interleaving abstraction of P_4 is

$$\begin{aligned} \mathcal{I}_H(P_4) &= h \to \mathcal{I}_H(P_4) \,\square\, h \to P_4' \\ P_4' &= h \to P_4' \,\square\, l \to \mathcal{I}_H(P_4) \end{aligned}$$



This abstraction is nondeterministic since an l can be both accepted and refused after a single h event. P_4 is insecure because the availability of an l event depends on the high-level user communicating the corresponding h. This is shown in $\mathcal{I}_H(P_4)$ by the h events from the initial state; an h event either leads back to the same state which corresponds to the high-level user refusing the event in P_4, or to a new state corresponding to the high-level user accepting the event.

(End of Example)

As with the eager and lazy conditions, the interleaving security conditions requires the abstraction to be deterministic. It is possible to give a traces formulation that uses interleaving and is equivalent to the lazy security condition. A formulation is given in [Ros95a] that parallels that for eager abstraction (Proposition 3.2.3); if two traces of P are only different in their high-level events it is required that the subsequent behaviour of P after these traces is the same when obfuscated by Run_H.

§Proposition 3.2.16 For non-divergent $P \setminus H$, $\mathcal{L}_H(P)$ is deterministic if and only if

$$s \upharpoonright L = t \upharpoonright L \Rightarrow (P/s) \;|||\; Run_H =_{\mathrm{FD}} (P/t) \;|||\; Run_H$$

The interleaving abstraction and the lazy abstraction appear very different. They have diametrically opposed paradigms, the first adding new events to obfuscate the high-level events and the second concealing the existing events. The abstractions are actually closely related. The lazy abstraction is strictly stronger than the interleaving one; in other words, if two processes can be distinguished by the lazy abstraction, they can also be distinguished by the interleaving abstraction. For processes that are deterministic under abstraction the result is even stronger.

§Theorem 3.2.17 The following conditions are equivalent:

(i) The abstract process $\mathcal{I}_H(P)$ is deterministic.
(ii) The abstract process $\mathcal{L}_H(P)$ is deterministic.
(iii) L is independent in P.

There are two main disadvantages to the interleaving abstraction when looking at security[6]. Firstly it cannot be said to be how the low-level user observes the system. All of the process' H labelled events, as well as those introduced by Run_H are visible. As previously argued, the low-level user's view of the system cannot contain any high-level events or it is trivial for information to flow. Secondly the additional transitions produce a much larger system for an automatic verification tool such as FDR to check and one that "does not lead itself nearly as well to facilitating compression"[Ros98]. This effects both the size of the problems that can be dealt with and the time taken to check them.

The interleaving abstraction has the advantage that it is defined in terms of the standard interleaving operator and Run_H function. FDR implements the interleaving operator directly and can verify whether or not a process is deterministic. The lazy abstraction over $\mathcal{N}$ does not use standard operators although it is equivalent for divergence-free, finitely nondeterministic processes to the abstraction over $\mathcal{F}$. The LTS for the interleaving abstraction is also easy to reason about since it only adds new transitions and not states; it is used in Section 6.1 to prove the equivalence of lazy abstraction to the local non-interference security properties for deterministic processes.

[6]Outside of a security context interleaving makes too many distinctions between process. For example $P = h \to l \to Stop$ and $P' = h \to h \to l \to Stop$ are distinguishable despite being indistinguishable to the low-level user.

Mixed Abstraction

Unlike process algebras such as CCS, CSP does not distinguish between input and output events; communication between processes is by the synchronisation of events rather than the matching of inputs to outputs. It is sometimes necessary to distinguish between inputs and outputs, or rather to differentiate between events that can be delayed and those that cannot. The reason we moved from the eager abstraction to the lazy one was that the former acted as if the high-level user was always willing to communicate, whereas the latter did not. If an event represents some action on the behalf of the high-level user we will want that event to be delayable for the reasons mentioned in the previous sections. If the event is not an action by the high-level user, but rather some feedback, it may be reasonable to assume the event cannot be delayed. For example, if the event represents some output to a high-level user's VDU or an alarm, it may not be one that the high-level user can prevent.

Example 3.2.18 Let $L = \{l\}$ and $H = \{d, s\}$. Consider the process

$$P_7 = l \to P_7 \;\Box\; d \to s \to P_7$$

The lazy abstraction of this process is

$$\mathcal{L}_H(P_7) = l \to \mathcal{L}_H(P_7) \sqcap \mathit{Stop}$$

This process is not lazily secure because the high-level user is able to communicate a d event and then refuse to communicate an s event, thus deadlocking the system. If s is a delayable event it is desirable that the lazy abstraction is not deterministic. If s is not a delayable event we would wish to regard this process as secure because the low-level user will always be able to communicate an l.

(End of Example)

The distinction between delayable events and undelayable *signal* events is made by the lazy and eager abstractions respectively. If we divide the set of high-level events into two parts, the delayable events D and the signal events S, we can form a hybrid condition. The events in S are observable to the high-level user, and so future decisions may be based on them, but they occur without the 'agreement' of the high-level user and hence do not explicitly affect the low-level user's view of the system. These events will be hidden; i.e. they are assumed to occur eagerly. The events in D do require the co-operation of the high-level user and hence will be treated by the lazy abstraction.

Definition 3.2.19 Let $H = D \cup S$ where events in D are delayable and those in S are not. Given a process P, the *mixed* abstraction is given by

$$\mathcal{M}_H^S(P) \mathrel{\widehat{=}} \mathcal{L}_H(P \setminus S)$$

As with the previous abstractions, there is a corresponding security condition that requires the mixed abstraction to be deterministic. The mixed abstraction will diverge if $P \setminus S$ is divergent; this corresponds to an unbounded number of high-level signal events. If this is undesirable then the abstraction cannot be used when $P \setminus S$ is divergent.

Example 3.2.20 Let $D = \{d\}$, $S = \{s\}$ and P_7 be as defined in Example 3.2.18. The mixed abstraction of P_7 is

$$\begin{aligned} \mathcal{M}_H^S(P_7) &= \mathcal{L}_H(\mu\, X . l \to X \;\Box\; h \to X) \\ &= \mu\, X . l \to X \end{aligned}$$

This is deterministic and so P_7 satisfies the mixed security condition.

(End of Example)

3.3 Nondeterminism

Independence and the deterministic abstractions provide a strong security framework for processes that appear deterministic to the low-level user. In addition, the deterministic processes that they class as insecure are those where the high-level user introduces some nondeterminism and hence can pass information. They provide very natural specifications of security which are in agreement with each other, as well as several more general conditions[7], for processes in their domain even though they approach the problem in distinct ways. Furthermore, by concentrating on deterministic properties, they avoid the difficulties introduced by differing models of concurrency.

The determinism based conditions are too strong however, since they identify some processes which do not pass any undesirable information as insecure. In other words, there are some nondeterministic processes, that also appear nondeterministic to the low-level user, which are secure.

Example 3.3.1 Let P be an arbitrary process that only communicates in L. This process is secure because the high-level user is given no opportunity to interact with the system and hence cannot have any effect on the low-level user. The lazy abstraction $\mathcal{L}_H(P)$ is just P, and so the process will only be lazily secure if it is deterministic. The lazy abstraction, by removing the distinction between internal choices and nondeterminism due to the high-level user, cannot be used to distinguish secure and insecure nondeterministic processes.

(End of Example)

There are also nondeterministic secure processes that communicate in both H and L, although the processes that are believed to be secure vary with the different security conditions (see Chapter 4). All these conditions acknowledge some processes as never giving the high-level user the opportunity to pass information to the low-level user. The existence of secure processes that are not deterministic or independent in L means there is some need to extend beyond the determinism based conditions.

There are several approaches for modelling and interpreting nondeterministic processes and general discussions can be found, for example, in [FG93a], [Ros98] and [Miln89]. Specific discussion on the pros and cons of the different formulations and interpretations of nondeterminism, as they relate to security, can be found in [Ros95a] and [RRF99]. The emphasis in the two papers is different, the former highlights the difficulties nondeterminism introduces and advocates determinism, whilst the latter looks at the relative advantages of behavioural models and transition systems.

The first question for nondeterministic processes is how we are going to distinguish between them. Different process algebras and associated models

[7] Including invariance (Definition 4.2.12), BNDC (Definition 4.3.2) and the local non-interference conditions (Chapters 5 and 6).

make differing assumptions on how processes can be observed and hence what it means for two processes to be equivalent. The equivalences we are primarily concerned with are failures/divergences from CSP and weak bisimulation from CCS.

The effect of choosing a model and associated equivalence assumes that the users of the system are using the same equivalence to observe the system. If the low-level user can observe the system in a way that differentiates between processes we consider equivalent, there might be the opportunity for inferring additional, undesirable information; for deterministic systems there is no additional distinction between processes. These deductions are in addition to any inferences from nondeterminisms that we can already differentiate in the model.

Example 3.3.2 Consider the processes

$$\begin{array}{rcl} P & = & Stop \sqcap (l_1 \rightarrow Stop \mathbin{\Box} l_2 \rightarrow Stop) \\ P' & = & Stop \sqcap (l_1 \rightarrow Stop \sqcap l_2 \rightarrow Stop) \end{array}$$

The processes P and P' are failures/divergences equivalent but they are not weakly bisimilar. If, for example, different actions by the high-level user can lead a system S to behave like P or P', a low-level user who can make distinctions between these processes, say by refusals testing, will be able to deduce which high-level event has taken place. A low-level user who can only observe failures will be unable to make this deduction. The decision as to whether S is secure or insecure will be dependent on the equivalence being used.

(End of Example)

It may be reasonable to make an assumption about how the users of the system are able to observe it to negate this problem. A more serious issue is that our model and equivalence might be too weak to distinguish between processes that are insecure and those that are secure.

Example 3.3.3 Consider the process $LEAK$ which echoes to the low-level user the events the high-level user has communicated

$$LEAK = h?x \rightarrow l!x \rightarrow LEAK$$

This is a process that is clearly insecure and none of the security conditions discussed herein would be satisfied. On the other hand, it would seem reasonable that the process $Chaos_{H \cup L}$ (as defined on page 21) is secure, since any observations by the low-level user may have happened with or without high-level communications.

Under the various denotational models of CSP we have

$$LEAK \sqcap Chaos_{H \cup L} = Chaos_{H \cup L}$$

This follows since *Chaos* is the zero of the external choice operator. It is difficult to think of $LEAK \sqcap Chaos_{H \cup L}$ as secure, since it can make an initial choice to behave as an insecure process.

(End of Example)

The first offshoot of this is that, if we want to differentiate between such processes or have a security condition that does so, we cannot arbitrarily replace a nondeterministic process with one that is semantically equivalent to it. The extension of this to general refinement is discussed in Section 7.1. It also illustrates that we need to be careful what we are considering the equivalence over; if we are just comparing the system or some abstraction of it we may be in the situation of being unable to distinguish between secure and insecure processes.

Example 3.3.4 The process $Chaos_L$ should be considered secure since, as in Example 3.3.1, it does not communicate in H. The process

$$LEAK \sqcap Chaos_L$$

appears insecure since it can initially become $LEAK$. Furthermore, whenever the high-level user is given the opportunity to communicate, the process is behaving as $LEAK$ and is insecure. The low-level views of both $Chaos_L$ and $LEAK \sqcap Chaos_L$ are the same under CSP's models and so a failures/divergences condition based purely on the low-level user's view of the system would be unable to distinguish them.

(End of Example)

This problem is not limited to the failures/divergences equivalence. A similar pair of secure and insecure processes, that are identical under weak bisimulation, are shown later in Theorem 4.3.11; we can built a system that can exhibit every low-level observation from an insecure process. This suggests that it is not sufficient to look at simple process abstractions and that it is necessary to look more closely at the behaviours of the individual states.

The behavioural models make different implicit assumptions about how the users are observing the system. In particular they are unable to tell when a nondeterminism is resolved[8]. The three processes

$$\begin{aligned} P_1 &= (l_0 \to l_1 \to \mathit{Stop}) \sqcap (l_0 \to l_2 \to \mathit{Stop}) \\ P_2 &= (l_0 \to l_1 \to \mathit{Stop}) \Box (l_0 \to l_2 \to \mathit{Stop}) \\ P_3 &= l_0 \to (l_1 \to \mathit{Stop} \sqcap l_2 \to \mathit{Stop}) \end{aligned}$$

are failures/divergences equivalent but exhibit nondeterminism in different ways.

A low-level user who is able to make snapshots of a process and then return to these states, for example by a core dump or some special operator, will be able to differentiate between these three processes. If a snapshot is taken before an l_0 event is communicated, but when it is available (so the internal choice in P_1 has taken place), P_1 can be differentiated from P_2 and P_3 because its snapshot will only ultimately offer one of l_1 or l_2. Alternatively, if a snapshot is taken immediately after the l_0 event, P_2 and P_3 can be differentiated because the snapshot of P_2 is only ever able to communicate one of l_1 or l_2, whereas P_3's can communicate either. These are all valid distinctions because, even if the low-level user cannot guarantee taking the snapshot at the required time, it is possible that it will occur at that time.

If we move over to a transition system based equivalence we can deal with snapshots. We can identify the individual states of a process and these are the points where we might take snapshots. Under weak bisimulation the three processes P_1, P_2 and P_3 are all distinguishable.

Finally, it is important to ask how we are modelling nondeterminism and there are several ways we might look at it. *Probabilistic* nondeterminism arises when a process behaves according to the laws of probability. Under

[8]A discussion on how the high-level user can exploit the differences in these types of nondeterminism can be found on page 87.

this nondeterminism model we have completely established distributions and deductions can be based on the statistical behaviour of the system. The process $P = (h \rightarrow P_1) \triangleright P_2$ would not be secure if the distributions at P_1 and P_2 were different. Probabilistic nondeterminism has been discussed in a security context, for example in [WJ90] and [McL90], but this is outside the scope of this thesis.

An alternative view of nondeterminism, and the one that is usually considered when creating concurrent systems, is *under-specification* or *don't-care* nondeterminism. Under this view, nondeterminism is seen as an abstraction of what is actually occurring. Indeed, a process does not have to exhibit all the behaviours shown in its model, but every behaviour it does exhibit must be shown. The actual behaviour in an implementation might be deterministic, probabilistic, resolved at an unmodelled level or by unmodelled agents.

This ambiguity about how the under-specification nondeterminism is being resolved causes a problem because we cannot tell whether two nondeterministic choices at different parts of the system, which appear equivalent, are actually resolved in the same way. Alternatively, two choices that are different might have a common refinement and be resolved in the same way.

Example 3.3.5 Consider the process

$$P = (h \rightarrow (l_1 \rightarrow \mathit{Stop} \sqcap l_2 \rightarrow \mathit{Stop})) \triangleright (l_1 \rightarrow \mathit{Stop} \sqcap l_2 \rightarrow \mathit{Stop})$$

If the first nondeterministic choice is always resolved as an l_1 and the second as an l_2, i.e. the process actually behaves as

$$P' = (h \rightarrow l_1 \rightarrow \mathit{Stop}) \triangleright l_2 \rightarrow \mathit{Stop}$$

the low-level user will be able to deduce when an h event has taken place or not. The nondeterminisms may be resolved this way if, for example, P' has been used as an implementation of the specification P, or the unmodelled part of the system which makes the choice is dependent on the number of h events that have taken place.

(End of Example)

The ambiguity is not present in the probabilistic case because it is not possible for two nondeterministic choices that we identify as equivalent to behave differently. An alternative approach is to add further identification to choices so that we can make enough discriminations to differentiate between nondeterminisms that may behave differently. An operational semantics that adds such additional information can be found in [Low99]. This semantic adds additional labels to all transitions and makes nondeterministic choices by placing a preference ordering on these labels. Although there are many different orderings of the labels, any two states which have nondeterministic choices marked in the same way, will behave in the same way for all orderings.

The security properties for nondeterministic processes that are considered in Chapter 4 assume that there is no probabilistic information to be determined about either local or global behaviours. In addition, they also assume that, as far as the users can determine, all of the nondeterministic behaviours that appear in the transition system can arise. If they did not the fact that a refinement P' of P existed that was insecure would mean that P was classed as insecure. This treatment of nondeterminism does not fit into any of the previous moulds. All transitions in the nondeterministic system are viewed as actual possibilities that may occur, and so this is *possibilistic* nondeterminism. One of the aims of the nondeterministic security conditions is therefore to ensure the available nondeterministic choices do not change.

Example 3.3.6 Let P be as defined in Example 3.3.5. With a possibilistic interpretation of nondeterminism, both of the $l_1 \to \mathit{Stop} \sqcap l_2 \to \mathit{Stop}$ choices must have the potential for offering l_1 and l_2. The two choices are therefore equivalent and indistinguishable to the low-level user.

(End of Example)

This interpretation of nondeterminism has not always been stated explicity in the literature. Without it we are left with the under-specification nondeterminism[9], whose ambiguities mean an implementation is not necessarily secure. This is the *refinement paradox*, discussed in Section 7.1, where a secure process can be refined by an insecure one.

There are various approaches to defining security properties for nondeterministic processes. Unlike the determinism based conditions, which agree

[9]Conditions based on other interpretations are explicit.

on which processes are classified as secure, there is no general agreement for nondeterministic processes. This is in part due to the different interpretations of equivalence but it is also due to varying opinions on which parts of the system need to be equivalent. This in turn comes from the diverse assumptions that can be made as to the abilities of the users. These are differences and difficulties which are not present in deterministic systems.

Chapter 4

Nondeterministic Conditions

There are two main paradigms for discussing the flow of information: non-interference and non-deducibility. The first condition is concerned with the low-level user being 'interfered with' by the high-level user. The latter condition is concerned with the low-level user deducing information about the high-level user's behaviour. These two conditions were introduced in the early/mid eighties; subsequently new conditions were introduced that were variants which enhanced or fixed problems with the earlier conditions. This chapter introduces these two conditions and some of their later counterparts; it is primarily concerned with how they apply to nondeterministic systems.

Section 4.1 introduces non-deducibility and shows that it fails as a secure condition for several reasons; in particular it cannot detect deadlocks, allows the high-level user to send information to the low-level user and is not compositional. The main rewriting of the condition, *non-deducibility on strategies*, is then introduced. There are two different approaches to this new condition, based on interleaving traces and composing with high-level process; these are compared and contrasted.

Section 4.2 introduces non-interference and since the original formulation was for deterministic systems two flavours of non-interference for nondeterministic systems are discussed. *Invariance* takes a similar approach to non-interference, by considering the potential behaviours (which may include failures) after a trace. The *nondeterministic non-interference* family takes a more global system view by comparing different process abstractions. Problems are highlighted for both these methods.

The two approaches, non-interference and non-deducibility, are brought together in Section 4.3 where several conditions considered by Focardi and Gorrieri are discussed. Their condition of *bisimulation non-deducibility on compositions* is based on non-deducibility on strategies which, because of its formulation, has been usefully related to nondeterministic non-interference. Several new results and difficulties are shown for this condition, such as its relationship to other properties and a method for generating insecure processes that are incorrectly identified as secure.

There are several common problems that run through the various conditions, including the ones mentioned above for non-deducibility. These problems are discussed for the various conditions and examples given. A more detailed overview of some of these problems is given in the next chapter, in Section 5.1, as motivation for introducing some new security conditions. Every attempt has been made to use original examples to illustrate these flaws; nevertheless it is inevitable, in keeping to simple examples, that there are similar to be found in the literature. The only significant example is marked in the text.

Finally, several of the properties in Sections 4.1 and 4.2 make a distinction between inputs and outputs, although the distinctions between the two items is not always made clear in the literature. When an example requires the distinction to be made, the high-level outputs will be the set $\bar{H}$; it is assumed that events are either inputs or outputs and so the high-level input events are $H \setminus \bar{H}$. Similarly for low-level events. Note that this is not the CCS input / output convention.

4.1 Non-Deducibility

The non-deducibility security property was first given by David Sutherland in 1986 [Sut86], where it was also contrasted with the non-interference properties from [GM82] (see Section 4.2). Under this property there is no information flow if the low-level user is unable to deduce anything about the high-level user's behaviour. Sutherland considers the traces of a process (which he calls *possible worlds*) and *information functions*, whose domains are the process' traces, which are considered as *views* of the system.

Definition 4.1.1 Given a set of possible worlds W and two functions f and g with domain W, information does *not* flow from f to g, i.e. the views are *non-deducible*, if and only if the function $f \times g$ from W to the cross product of the images of f and g is onto.

In other words, if the condition does not hold, there must be $w \in \mathbf{ran}(g)$ and $v \in \mathbf{ran}(f)$ such that, for all possible worlds t, $f(t) = v \Rightarrow g(t) \neq w$[1]. Hence a user who observes the system through the view g and who observes a w, can deduce that v is not observable through view f.

The non-deducibility property is then put in a security context by defining f and g. We are trying to prevent the low-level user's observations allowing any deductions about the high-level behaviour (i.e. inputs), and so the two functions are chosen to correspond to these two sets of events. The function g, originally called *view*, consists of the low-level events from a given trace. The function f, originally called *hidden*, is chosen to characterise the high-level behaviour and is defined as the high-level input events from the trace.

Definition 4.1.2 Let δ be a trace of P,

$$\begin{array}{rcccl} f(\delta) & = & hidden(\delta) & = & \delta \upharpoonright (H \setminus \bar{H}) \\ g(\delta) & = & view(\delta) & = & \delta \upharpoonright L \end{array}$$

The low-level user's view of the system, i.e. the low-level inputs and outputs, is g. Undesirable information is obtained by the low-level user when they can deduce something about f, the high-level inputs. This occurs when the condition does not hold, as discussed above, because there is some w which the low-level user can observe which allows the elimination of some potential high-level input.

Example 4.1.3 Let $L = \{l\}$ and $H = \{h\}$ where all events are input events. Consider the process

$$P = h \rightarrow l \rightarrow Stop$$

[1]Proof in [Sut86]

Now

$$
\begin{aligned}
\mathit{traces}(P) &= \{\langle\rangle, \langle h\rangle, \langle h, l\rangle\} \\
\mathbf{ran}(f) &= \{\langle\rangle, \langle h\rangle\} \\
\mathbf{ran}(g) &= \{\langle\rangle, \langle l\rangle\} \\
\mathbf{ran}(f \times g) &= \{(\langle\rangle, \langle\rangle), (\langle h\rangle, \langle\rangle), (\langle h\rangle, \langle l\rangle)\}
\end{aligned}
$$

Now $\mathbf{ran}(f \times g) \neq (\mathbf{ran}\ f \times \mathbf{ran}\ g)$ since $(\langle\rangle, \langle l\rangle) \in (\mathbf{ran}\ f \times \mathbf{ran}\ g)$. In other words, whilst the low-level user cannot deduce whether an h event is still available if the l event has not been communicated, once the l event has occurred the low-level user knows the h event has been communicated.

(End of Example)

The non-deducibility condition can be given in a more traditional process algebra based formulation. Allen [All91] puts non-deducibility into a CSP traces setting, as well as considering conditions with more than two levels of security.

Definition 4.1.4 A process P is *non-deducible* if

$$
\begin{aligned}
&\forall\, tr \in \mathit{traces}\,[\![P \restriction L]\!], tr' \in \mathit{traces}\,[\![P \restriction (H \setminus \bar{H})]\!] \bullet \\
&\quad \exists\, tr'' \in \mathit{traces}\,[\![P]\!] \bullet tr'' \restriction L = tr \;\wedge\; tr'' \restriction (H \setminus \bar{H}) = tr'
\end{aligned}
$$

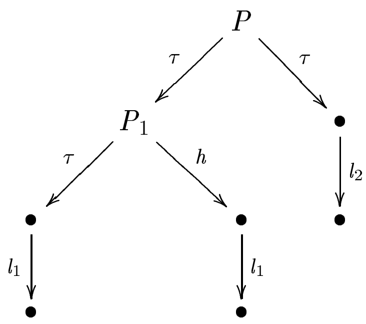
In other words, for every low-level view of the system every possible high-level view might also have occurred.

The information flow relationship is symmetric, at least with respect to high-level inputs, whereas we are usually concerned with asymmetric information flow. In other words, with non-deducibility, not only is the high-level user unable to affect the low-level user, but the low-level user cannot affect the high-level user's potential inputs. Sutherland argues for this relationship by moving the problem to the choice of whether information is labelled as high or low; the potential absence of high-level information is a suitable inference.

There are processes which the non-deducibility paradigm calls insecure which it would seem reasonable to call secure. The symmetry of non-deducibility does not allow the low-level user to deduce anything about high-level behaviours even if this information is not due to any action (or inaction) on the part of the high-level user. This information might be directly due to the low-level user's actions, or to some internal nondeterminism.

Example 4.1.5 Let $L = \{l_1, l_2\}$ and $H = \{h\}$ where all the events are inputs. Consider the process

$$\begin{aligned} P = \quad & ((h \to l_1 \to \mathit{Stop}) \rhd l_1 \to \mathit{Stop}) \\ \sqcap \quad & l_2 \to \mathit{Stop} \end{aligned}$$



P is a process which we could argue as being secure. If the process is in state P_1 where the high-level user is able to communicate an h event, neither the occurrence or non-occurrence of this event affects what the low-level user can observe. The state P_1 is equivalent to $P_1' \underset{h}{\overset{\tau}{\rightrightarrows}} \bullet \xrightarrow{l_1} \bullet$.

This process is not non-deducible since the low-level user, on being able to communicate an l_2 event, is able to deduce that no h event has taken place. This is despite the fact that the absence of the h event was not due to any decision taken by the high-level user.

(End of Example)

Example 4.1.6 Let $L = \{l\}$ and $H = \{h\}$ where all the events are inputs. Consider the process

$$P = l \to h \to \mathit{STOP}$$

This process is not non-deducible, despite the high-level user being unable to influence the low-level user.

(End of Example)

One of the potential problems with non-deducibility is that the chosen purge functions are unable to distinguish traces containing different interleavings of the low-level and high-level traces. If non-deducibility holds the low-level user might be unable to tell which of several high-level traces has occurred, but different traces might only be possible at different points with respect to the low-level trace. This inability to distinguish between events that have occurred 'before' and 'after' can, for example, prevent the low-level user from detecting potential deadlocks caused by high-level actions / refusals.

Example 4.1.7 Let $L = \{l\}$ and $H = \{h\}$ where all events are input events. Consider the process

$$P = l \to P \;\Box\; h \to h \to P$$

Now

$$\begin{array}{rcll} traces(P \upharpoonright L) & = & L^* & \\ traces(P \upharpoonright H) & = & H^* & \\ \langle l \rangle^n \frown \langle h \rangle^m & \in & traces(P) & \qquad \forall\, n, m \end{array}$$

so P satisfies non-deducibility. If the trace has an odd number of high-level events the last event in the trace must be an h. The process is not secure because the next event must be another h and so the high-level user is able to cause a deadlock for the low-level user by refusing h; P is not lazily secure for this reason. The non-deducibility condition is 'fooled' because a trace with one additional l event is always possible, even though it must have occurred before the last h event.

The high-level user has the opportunity to prevent the low-level user interacting with the system on every alternate h event. Two co-operating users could use this as a covert channel that allows them to transmit an arbitrary amount of information.

(End of Example)

The lazy security condition does not imply non-deducibility; for example, the lazy condition is asymmetric and permits processes which have information flow from low to high, whereas non-deducibility is symmetric. One implication of Example 4.1.7 is that the reverse implication does not hold, even for deterministic processes.

Non-deducibility prevents the low-level user from deducing anything about the behaviour of the high-level user, but it does not prevent the high-level user from passing on information. The results of the high-level user's actions can be different, so long as no information about those actions can be deduced from the results. Non-deducibility allows the low-level user to observe these different system behaviours.

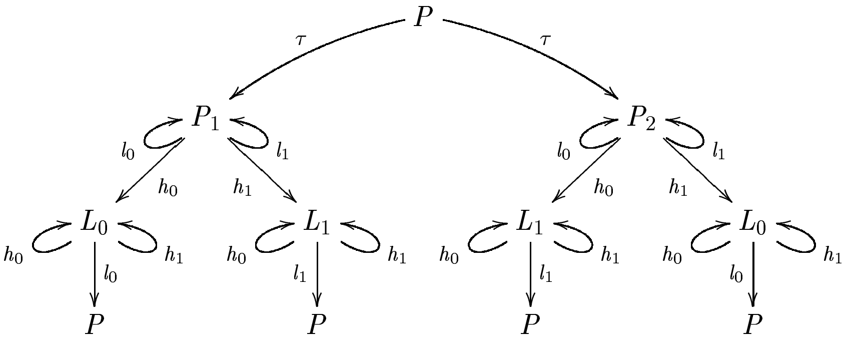
In [WJ90] a system is given which allows an arbitrary amount of information to be passed from high to low. The system allows the high-level user to observe how certain internal nondeterminisms have been resolved, and then use this information to influence future high-level inputs. By providing different high-level behaviours (and the results of those behaviours) after different branches of the nondeterministic choice, the non-deducibility condition is maintained. Since high-level outputs are not considered by non-deducibility this information flow is not detected. Non-deducibility, by not considering some information that is available to the high-level user, is unable to model all the ways the high-level user and the system may interact.

The high-level user can use knowledge of internal nondeterminisms to dictate what the low-level user observes. Depending on our view of the high-level user's abilities, this information might involve the system explicitly informing the high-level user of the nondeterminism's resolution through output events, or the high-level user might observe the explicit state the LTS is currently in.

Example 4.1.8 Let $L = \{l_0, l_1\}$ and $H = \{h_0, h_1\}$ where all the events are inputs. Consider the processes

$$\begin{array}{rcl}
P & = & P_1 \sqcap P_2 \\
P_1 & = & l_0 \to P_1 \,\Box\, l_1 \to P_1 \,\Box\, h_0 \to L_0 \,\Box\, h_1 \to L_1 \\
P_2 & = & l_0 \to P_2 \,\Box\, l_1 \to P_2 \,\Box\, h_0 \to L_1 \,\Box\, h_1 \to L_0 \\
L_0 & = & h_0 \to L_0 \,\Box\, h_1 \to L_0 \,\Box\, l_0 \to P \\
L_1 & = & h_0 \to L_1 \,\Box\, h_1 \to L_1 \,\Box\, l_1 \to P
\end{array}$$

The additions of the L labelled loops at P_1 and P_2 are to prevent the low-level user deducing that a high-level event must have occurred because a low-level event has occurred. Similarly the H labelled loops at L_0 and L_1 prevent upper bounds on the number of high-level events being deduced.



P satisfies non-deducibility. The nondeterministic choice ensures that any low-level trace is observable after every high-level trace; for example after an h_0 event both l_0 and l_1 are possible, depending on whether the system is in state P_1 or P_2.

If the high-level user knows whether the system is currently in state P_1 or P_2, the high-level event can be chosen to dictate which low-level event will follow. This can lead to a covert channel; for example two co-operating users can ensure that the only low-level events that are ever communicated come as the result of the high-level inputs.

If our model does not provide the high-level user with knowledge about which state the system is currently in, we can create a similar system which passes this information to the high-level user via the outputs $\bar{H} = \{\bar{h}_0, \bar{h}_1\}$

$$Q = \bar{h}_0 \to Q_1 \sqcap \bar{h}_1 \to Q_2 \qquad etc\ldots$$

This process will still satisfy non-deducibility since the information on how the nondeterminism has been resolved is passed to the high-level user through outputs.

(End of Example)

Non-deducibility also fails to be compositional; this was first shown by McCullough in [McC87] and further discussed in [McC88] and [McC90]. In other words we can 'hook-up' two systems that satisfy the condition and get a resulting system that does not satisfy it. The following example, due to McCullough, shows how small information leaks which do not invalidate the security condition can be combined into larger leaks that do.

Example 4.1.9 Let $L = \{halt, odd, even\}$ and $H = \{h_1, h_2\}$ where $\bar{H} = \{h_2\}$ and all the other events are inputs[2]. Consider the process

$$\begin{array}{rcl} P_{even} & = & halt \to even \to Stop \,\square\, h_1 \to P_{odd} \,\square\, h_2 \to P_{odd} \\ P_{odd} & = & halt \to odd \to Stop \,\square\, h_1 \to P_{even} \,\square\, h_2 \to P_{even} \end{array}$$

P_{even} is a process that accepts high-level inputs and outputs until a low-level *halt* event is communicated, after which it lets the low-level user know the parity of the number of high-level events. This process is non-deducibly secure because after any number of h_1 events an *even* or *odd* event can ultimately be communicated; the parity can be altered by the h_2 events which are not considered by the condition.

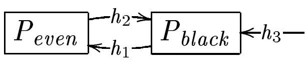
We can create a similar process. Let $L = \{halt, black, white\}$ and $H = \{h_1, h_2, h_3\}$ where $\bar{H} = \{h_1\}$ and all the other events are inputs. Consider the process

$$\begin{array}{rcl} P_{black} & = & halt \to black \to Stop \,\square\, h_3 \to P_{white} \,\square\, h_2 \to P_{white} \,\square\, h_1 \to P_{white} \\ P_{white} & = & halt \to white \to Stop \,\square\, h_3 \to P_{white} \,\square\, h_2 \to P_{black} \,\square\, h_1 \to P_{black} \end{array}$$

The process P_{black} also satisfies non-deducibility by analogous reasoning.

Consider the composition of these processes

$$P_{even} \underset{\{halt, h_1, h_2\}}{\|} P_{black}$$



If no h_3 events have occurred the parity of the two component processes must match, since the other high-level inputs and outputs are matched. In other words, if no h_3 events have occurred the low-level user will be able to observe an *odd* and *white* event together or an *even* and *black* event, but no others. If the parity is broken the low-level user can deduce that at least one h_3 event (which is a high-level input) has been communicated.

(End of Example)

Various conditions have been proposed to overcome the compositional problem with non-deducibility, for example McCullough's *hook-up* security

[2]Non-deducibility does not differentiate between low-level inputs and outputs and so no distinction has been made for the events in L. An alternative view of the system would have *halt* as an input and *odd*/*even* as outputs.

([McC87]) and *restrictive* condition ([McC90]). The first of these places limitations on the relative positions of inputs and outputs. The second condition roughly states "equivalent states are affected equivalently by equivalent inputs, produce equivalent outputs and then remain equivalent", and explicitly requires *input totality*[3] to prevent the low-level user distinguishing states.

Non-Deducibility on Strategies

An alternative approach to non-deducibility was introduced by Wittbold and Johnson in [WJ90]; their condition is called *non-deducibility on strategies*, and was formulated for a state machine model. A strategy for the high-level user is how the user decides to act based on previous inputs and outputs. Informally, a process satisfies non-deducibility on strategies if whatever view the low-level user has of the system, it is consistent with every high-level strategy[4].

There are two ways non-deducibility on strategies has been approached in the literature. The formulation in [WJ90] is to say that, whatever low-level view has occurred, every valid high-level view might also have occurred. A trace based formulation can be given for this [RS99][5] that differs with that for non-deducibility (see Definition 4.1.4) only by removing the distinction between high-level inputs and outputs:

Definition 4.1.10 A process P is *non-deducible on strategies* if

$$\begin{array}{l}\forall\, tr \in traces\,[\![P \restriction L]\!], tr' \in traces\,[\![P \restriction H]\!] \bullet \\ \qquad \exists\, tr'' \in traces\,[\![P]\!] \bullet tr'' \restriction L = tr \ \wedge\ tr'' \restriction H = tr'\end{array}$$

The second approach is to say that the result of composing any high-level strategy with the system should appear identical to the low-level user. In other words, the processes $\Pi \underset{H}{\|} P$ should be equivalent for all possible

[3] A process is *input total* if every input event is always available.

[4] The high-level strategies are restricted to deterministic ones in the original formulation.

[5] In the paper this formulation is given as a test which must be passed for a given high-level strategy. Since $Run_H \sqsubseteq_T P$ for all high-level P, it is only necessary to test the original process in parallel with Run_H (i.e. the original process).

high-level processes Π. A formulation of this, from [FG93b], is discussed further in Section 4.3.

These two approaches do not agree on which processes are satisfied, even if we restrict the high-level strategies to deterministic ones. The traces approach requires, for every low-level trace, that every high-level trace might have taken place. To observe a given low-level event it may be the case that some high-level events cannot have taken place, even though the high-level strategy would have allowed synchronisation with them. The compositional formulation allows for this.

Example 4.1.11 Let $L = \{l_1, l_2\}$ and $H = \{h\}$. Consider the process

$$P = \tau.l_1.0 + \tau.l_2.0 + h.l_1.0$$

$$\begin{tikzcd}
& P \arrow[dl, "\tau"'] \arrow[d, "\tau"] \arrow[dr, "h"] & \\
\bullet \arrow[d, "l_1"'] & \bullet \arrow[d, "l_2"] & \bullet \arrow[d, "l_1"] \\
\bullet & \bullet & \bullet
\end{tikzcd}$$

P does not satisfy the traces formulation since the availability of an l_2 event implies the absence of an h event. The process does satisfy the composition formulation since, for all possible high-level strategies, the low level user can observe the traces $\langle\rangle$, $\langle l_1\rangle$ and $\langle l_2\rangle$.

It seems reasonable to argue for this process being insecure. From a non-deducibility point of view the occurrence of an l_2 event allows the low-level user to deduce the absence of an h event. If the high-level user is offered the opportunity to communicate before the τ actions are resolved, for example if they are abstracting an internal 'timeout' type event, the high-level user can ensure that the low-level user never observes an l_2; the appearance of an l_2 now indicates a definite choice by the high-level user.

(End of Example)

There is an alternative reason why the two approaches may disagree. The high-level strategy may be unimportant if the h labelled events do not affect the process, at least with respect to its low-level appearance.

Example 4.1.12 Let $L = \{l\}$ and $H = \{h\}$. Consider the process

$$P = l \rightarrow (l \rightarrow P \Box h \rightarrow P)$$

P does not satisfy the traces formulation since the low-level user can infer an upper limit on the number of h events that have taken place, whereas $traces\,[\![P \upharpoonright H]\!] = \{h\}^*$. The composition formulation is satisfied since every possible high-level process still leaves a system that can always communicate an l event.

Process P can be argued as being secure since there is no way for the high-level user to affect what the low-level user is able to communicate; indeed, P is lazily secure. The deduction the low-level user is able to make about an upper bound for the number of high-level events is not dependent on any high-level behaviour.

(End of Example)

In Example 4.1.8 there were additional loops added to the system to prevent the low-level user from deducing how many high-level events had taken place. The traces approach also requires any number of high-level events (from a potential high-level trace) to be available at any stage in the process. The second approach is more flexible; it allows for some relationship between the number of low-level and high-level events that have occurred at any state. The original Wittbold and Johnson formulation did not have this problem since it required strict synchronisation between the number of high-level and low-level events.

The traces formulation seems closer to the original non-deducibility condition, although the compositional approach appears to be a more reasonable condition; it places the burden on the high-level strategies rather than which strategies the system allows to execute. Example 4.1.11 and Example 4.1.12 demonstrate arguments for and against both. It is certainly more desirable for a condition to reject a secure process as insecure (the trace based formulation in Example 4.1.12) to one that accepts some insecure process as secure (the composition formulation in Example 4.1.11). Nevertheless, the traces formulation still disagrees with the lazy security condition for deterministic processes because it incorrectly identifies Example 4.1.7 as secure.

It can be seen from the traces formulation that non-deducibility on strategies implies non-deducibility. A process that fails to be non-deducibly secure, such as Example 4.1.3, must necessarily also fail non-deducibility on strategies. The reverse implication can clearly fail to hold for processes which pass information through outputs.

Example 4.1.13 Let P_{even} be as defined in Example 4.1.9, where it is shown to be non-deducible. This process does not satisfy non-deducibility on strategies since the low-level user can obtain the parity of the number of high-level events that have taken place.

(End of Example)

Wittbold and Johnson's condition still suffers from the problems highlighted with Example 4.1.7 and Example 4.1.8; the inability to differentiate between different interleavings of high-level and low-level traces, and the high-level user's ability to base decisions on nondeterminisms. The high-level user's previous ability to get the information on how the nondeterminisms have been resolved through outputs has been removed. Their condition is compositional however, as shown by Millen in [Mill90][6].

Example 4.1.14 Let P and Q be as defined in Example 4.1.8. Since there are no high-level outputs and P satisfies non-deducibility, P also satisfies non-deducibility on strategies. The process Q, which produces information to the high-level user as to which state it is in, does not satisfy the new condition.

(End of Example)

4.2 Non-Interference

The definition of *non-interference* originated with Joseph Goguen and Jose Meseguer in 1982 [GM82] and later in [GM84]; they cite [Fei80] and [FLR77] as giving less general and less formal concepts which non-interference builds upon, and [Rus81a] as providing a related automaton model based approach. Their intention was to introduce an approach to secure systems providing a formal way of specifying and verifying security policies, keeping a sharp

[6] A compositional argument is also given in [WJ90] but requires additional restrictions.

distinction between the policy and the system model. The security policies, which declared which information flows were not permitted, utilised noninterference. Informally this was introduced as:

> "one group of users, using a certain set of commands, is *non-interfering* with another group of users if what the first group does with those commands has no effect on what the second group of users can see."

In other words, if we are trying to show that a high-level user is non-interfering with a low-level user, we wish to demonstrate that the low-level user's view of the system is independent of the behaviour of the high-level user. An alternative view is to say that if we have a trace of the system and we alter the high-level inputs, the only change should be to the high-level outputs. Unlike non-deducibility, non-interference is an asymmetric condition.

The original definition of non-interference, for deterministic systems, is given by comparing the output to the low-level user after some inputs, to the low-level user's output after the same inputs with the high-level components purged. More formally, and adopting a more CSP like notation (as with [Rya91]):

Definition 4.2.1 Let the alphabet of P be fully partitioned by disjoint interfaces H and L. We say that H does not interfere with (or is *non-interfering* with) L if

$$\forall\, tr \in I^*, c \in I \bullet \mathit{Result}_L(P, tr, c) = \mathit{Result}_L(P, tr \upharpoonright L, c)$$

The set of possible inputs to P is given by I. The L labelled outputs resulting from the input c, when the process P is in a state after the trace tr (of inputs), is given by $\mathit{Result}_L(P, tr, c)$. The purge of the trace is the same as the standard CSP notation, i.e. $tr \upharpoonright L$.

The properties of, and hence the distinctions between, inputs and outputs are not defined. For example, it is not stated whether outputs must always be accepted or whether the system can refuse inputs. As seen with the eager and lazy abstractions in Chapter 3, this distinction is an important one. Their interpretation, as discussed later, has led to different formulations of non-interference.

Example 4.2.2 Let $L = \{query, even, odd\}$ and $H = \{h_1, \ldots, h_n\}$ where $\bar{L} = \{even, odd\}$ and all the other events are inputs. Consider the process

$$
\begin{aligned}
P_{even} &= query \rightarrow even \rightarrow P_{even} \,\Box\, (\Box\, i : \{1, \ldots, n\} \bullet h_i \rightarrow P_{odd}) \\
P_{odd} &= query \rightarrow odd \rightarrow P_{odd} \,\Box\, (\Box\, i : \{1, \ldots, n\} \bullet h_i \rightarrow P_{even})
\end{aligned}
$$

P_{even} is a process that accepts high-level inputs and does not pass on their content to the low-level user. There is an additional source of information however, since the low-level user can receive some parity information based on the number of inputs the high-level user has made. The low-level user is unable to infer anything about which H labelled events have taken place but may be able to detect when an event has taken place.

The system fails to satisfy the non-interference conditions because the system can change state, without adding anything to its purged trace, and permit the low-level user to observe a different event. For example,

$$
\begin{array}{lll}
Results_L(P_{even}, tr, query) = \{odd\} & \text{where} & tr \in H^* \wedge \#tr \ \mathbf{mod}\ 2 = 1 \\
Results_L(P_{even}, tr, query) = \{even\} & \text{where} & tr \in H^* \wedge \#tr \ \mathbf{mod}\ 2 = 0
\end{array}
$$

Since for all $tr \in H^*$ we have the same purged trace, i.e. $tr \restriction L = \langle\rangle$, the condition does not hold. This opportunity exists between any two L labelled events, hence an arbitrary amount of information might be transmitted between the two users.

(End of Example)

Unfortunately non-interference as given by Goguen and Meseguer is not compositional. This was first demonstrated by McCullough in [McC87] and further discussed in [McC88], along with similar examples for non-deducibility. McCullough's counterexample ties together outputs which depend on appearing ambiguous, but whose composition only gives certain combinations depending on the high-level inputs. As discussed in Section 5.1, being compositional is a very desirable property for a security condition, and is a flaw with this formulation of non-interference.

Nondeterministic Non-Interference

As originally stated the non-interference condition was for deterministic systems, where there is a unique output for any given input trace; for such systems we have no problem in specifying equivalence. It is desirable to extend the notion of non-interference to nondeterministic systems. For nondeterministic systems there is some uncertainty as to how to the low-level user should judge the outputs to be equivalent.

One approach to modelling a nondeterministic non-interference condition is to put the condition into a CCS context. Focardi and Gorierri discuss such an extension in [FG93a] and [FG97]. Their condition considers every system trace containing high-level inputs, and looks to see if there exists another trace with the same low-level events but without the high-level inputs. Since the two traces are indistinguishable to the low-level user it is argued that the low-level view of the system cannot be affected by high-level inputs.

Definition 4.2.3 A process P satisfies *nondeterministic non-interference* (NNI) if and only if

$$(P \setminus_{(H \setminus \bar{H})}) \setminus H \ =_{\mathrm{T}} P \setminus H$$

In other words, the traces of the process when H labelled inputs are not permitted ($P \setminus_{(H \setminus \bar{H})}$) should be the same as those for the process when these inputs are permitted, once all the H labelled events (both input and output) have been purged.

Example 4.2.4 Let the process P_{even} be as defined in Example 4.2.2. Any combination of *odd*s and *even*s will be a trace of $P_{even} \setminus H$; the traces of $(P_{even} \setminus_{(H \setminus \bar{H})}) \setminus H$ are $\{even\}^*$. Hence P_{even} does not satisfy NNI.

(End of Example)

Example 4.2.5 Let $L = \{l_1, l_2\}$ and $H = \{h\}$ where all the events are inputs. Consider the process

$$P = ((h \to l_1 \to Stop) \triangleright l_1 \to Stop) \sqcap l_2 \to Stop$$

as defined in Example 4.1.5. The traces of $P \setminus H$ and $(P \setminus_H) \setminus H$ are the same; i.e. $\{\langle\rangle, \langle l_1\rangle, \langle l_2\rangle\}$. Therefore P satisfies NNI.

(End of Example)

There is the important question raised earlier regarding the distinction between high-level inputs and outputs. The purge of a valid trace containing high-level outputs is not necessarily a trace of the system. Any L labelled event that can only take place after such an output will indicate to the low-level user the occurrence of such a high-level output. We could argue that any information the low-level user has about the high-level user is an undesirable information flow. Alternatively, if these output events are signals which cannot be delayed by the high-level user, it can be argued that this is not a genuine flow of information, since the high-level user is not given the choice between accepting and refusing.

Example 4.2.6 Let $L = \{l_1, \ldots, l_n\}$ be low-level input events and let $\bar{H} = \{h_1, \ldots, h_n\}$ be high-level output events. Consider the process

$$P = \Box\, i : \{1, \ldots, n\} \bullet l_i \rightarrow h_i \rightarrow P$$

This is a process that echos to the high-level user's output whatever the low-level user does. P might give the trace $\delta = \langle l_3, h_3, l_2\rangle$; the corresponding purged trace $\delta \upharpoonright L = \langle l_3, l_2\rangle$ which is also a trace of $(P \setminus_{H \setminus \bar{H}}) \setminus H$. This process therefore satisfies NNI.

The occurrence of a second L labelled event indicates the H labelled event corresponding to the first l_i has taken place. Similarly for all future L labelled events. If the high-level user has the option of refusing any output whether the system accepts or refuses a low-level input will indicate whether the high-level user is active.

(End of Example)

The approach in [FG93a] and [FG97] is to introduce a stronger condition. This requires every trace to remain a valid trace when it is purged of all of the high-level inputs and outputs. In other words, all high-level events are refusable. This condition no longer differentiates between high-level inputs and outputs.

Definition 4.2.7 A process P satisfies *strong nondeterministic non-interference*[7] (SNNI) if and only if

$$P \setminus_H \, =_{\mathrm{T}} P \setminus H$$

The problem with trace based equivalences being unable to detect liveness becomes relevant for Focardi and Gorierri's two conditions. Some cases, such as in Example 4.2.6, are detected due to the distinctions between hiding and restriction. This will not hold if the h event is not required for (i.e. it adds no) additional traces; for example, when h occurring causes deadlock, or the absence of an h causes deadlock and its occurrence returns the process to an earlier state.

Example 4.2.8 Let $L = \{l\}$ and $H = \{h\}$. Consider the process

$$P = l \to P \,\Box\, h \to \mathit{Stop}$$

This satisfies SNNI since the traces of $P \setminus H$ and $P \setminus_H$ are the same, i.e. $\{l\}^*$. This process can deadlock if and only if an h event occurs.

(End of Example)

Unlike the original non-interference condition we are no longer considering the events available after a trace, rather we are once again comparing a limited number of complete behaviours. This reintroduces the deadlock problems that can occur after several high-level events. Hence SNNI disagrees with the lazy security condition for deterministic processes.

Example 4.2.9 Let $L = \{l\}$, $H = \{h\}$ and let $P = l \to P \,\Box\, h \to h \to P$.

$$\begin{aligned} P \setminus_H &= l \to P \setminus_H \\ P \setminus H &= (l \to P \setminus H) \rhd (P \setminus H) \end{aligned}$$

Any set of l events will be a trace of both processes. Hence the process will satisfy both NNI and SNNI; it also satisfies non-deducibility (see Example 4.1.7).

(End of Example)

[7] The same condition is given by Jacob [Jac90b] in the CSP formulation $P \underset{H}{\|} \mathit{Stop} =_{\mathrm{T}} P \setminus H$.

The trace equivalence used for SNNI is too weak to express the required security property, as are the limited abstractions of the processes. More powerful equivalences for this condition have led to further CCS based conditions. Similarly there are modifications of the condition that range over the states. These new conditions have been used primarily as tools for testing stronger notions of non-deducibility. These are discussed more fully in Section 4.3.

Invariance

The issue of which properties distinguish input and output events is also resolved if we move to a process algebra, such as CSP, that makes no distinctions between inputs and outputs. It is sometimes desirable to have some distinction, for example in the case of high-level signal events that cannot be refused. By being specific about the properties of such events they can still be modelled; e.g. the signal events in Definition 3.2.19 being replaced by internal τ actions.

In [Rya91] Ryan presented a formulation of non-interference for CSP, and hence removed the distinction between inputs and outputs; this version has been further discussed by Ryan and Schneider in [RS99]. Other authors have written similar CSP formulations, for example [All91] and [Jac90b].

Ryan's condition, which is for non-divergent processes, looks at the events the low-level user is able to accept of refuse after a trace. For the low-level user to be unable to observe or infer any high-level information these acceptances and refusals need to be the same whenever the traces are indistinguishable to the low-level user.

Definition 4.2.10 A process P is *trace-invariant* in L if

$$s \upharpoonright L = t \upharpoonright L \Rightarrow (P/s)^0 \cap L = (P/t)^0 \cap L$$

If trace-invariance does not hold there must be an event not in L that can influence what the low-level user can observe[8]. If this event is from H, rather than from an internal τ event, we have an undesirable information flow. For a system that does not display any internal choices trace-invariance is sufficient; indeed, it is shown in [Wul97] that for L-deterministic processes trace-invariance is equivalent to the lazy security condition. If there are internal nondeterministic choices, to fully distinguish between τ and H labelled transitions, it is necessary to strengthen the condition to include failures.

Example 4.2.11 Let $L = \{l_1, l_2\}$ and $H = \{h\}$. Consider the process

$$\begin{array}{rcl} P & = & (h \rightarrow P_1) \triangleright P_2 \\ P_1 & = & l_1 \rightarrow \mathit{Stop} \sqcap l_2 \rightarrow \mathit{Stop} \\ P_2 & = & l_1 \rightarrow \mathit{Stop} \,\Box\, l_2 \rightarrow \mathit{Stop} \end{array}$$

This process satisfies trace-invariance since there is no difference between the traces from the internal and external choices of l_1 and l_2.

(End of Example)

Definition 4.2.12 A process P is *failures-invariant* in L if

$$\begin{array}{rcl} s \upharpoonright L = t \upharpoonright L & \Rightarrow & (P/s)^0 \cap L = (P/t)^0 \cap L \;\wedge \\ & & \textsc{Refs}(P/s) \cap \mathbb{P}\, L = \textsc{Refs}(P/t) \cap \mathbb{P}\, L \end{array}$$

Example 4.2.13 Let P be process defined in Example 4.2.11. P is not failures-invariant since the refusals for P_1 are not the same as those for P_2.

(End of Example)

Unlike the original formulation of non-interference this invariance condition is preserved under parallel composition. This result is proved in [Rya91].

The invariance conditions pass one basic check since they agree with the lazy security condition for both deterministic processes and those that are independent in L. This result, which is shown in [Wul97], holds for all L-deterministic processes and only trace invariance is required.

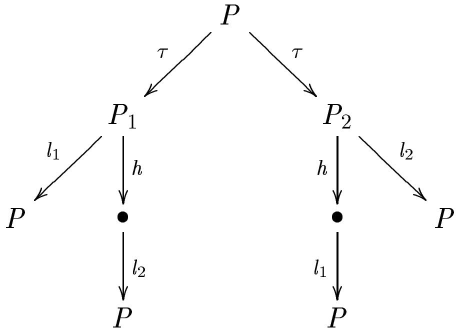
There are still problems with the non-interference / invariance conditions. It is still possible for the high-level user to influence what the low-level user is able to communicate. If the process has nondeterminisms it may

[8]Nondeterminisms introduced by identical $l \in L$ labellings are accounted for by $/$.

be possible for the high-level user to make the low-level user believe they have been resolved a different way. From the low-level user's perspective the nondeterminism has simply been resolved.

Example 4.2.14 Let $L = \{l_1, l_2\}$ and $H = \{h\}$. Consider the process

$$\begin{aligned} P &= P_1 \sqcap P_2 \\ P_1 &= l_1 \rightarrow P \;\Box\; h \rightarrow l_2 \rightarrow P \\ P_2 &= l_2 \rightarrow P \;\Box\; h \rightarrow l_1 \rightarrow P \end{aligned}$$



From the low-level user's perspective this is a process that nondeterministically decides to offer either an l_1 or an l_2 event. Now

$$\begin{aligned} \textsc{Refs}(P/\langle\rangle) \cap \mathbb{P}\, L &= \{\{l_2\}, \{l_1\}\} \\ &= \textsc{Refs}(P/\langle h \rangle) \cap \mathbb{P}\, L \end{aligned}$$

and similarly for the initial actions. The process therefore satisfies the various non-interference conditions. With the small modification of h labelled loops to each of the unnamed nodes (as in Example 4.1.8), the process will also satisfy the non-deducibility conditions.

If the high-level user has the opportunity to act it is after the nondeterminism has been resolved and before the low-level user has tried to communicate. In both states P_1 and P_2 the high-level user is able to determine, based on whether an h is communicated or not, whether the low-level user will have an l_1 or l_2 event available. If we permit the high-level user to know whether the system is in state P_1 or P_2, the system is certainly not secure since the high-level user can dictate what the low-level user can communicate.

(End of Example)

The failures model allows the detection of deadlock. If the occurrence of an H labelled event is required to deadlock the system, as in Example 4.2.8, failures-invariance will not be satisfied. Nevertheless, it is still possible for a process to satisfy the failures-invariance property whilst allowing the high-level user to exploit delayable events. An H labelled event may be required to prevent deadlock.

Example 4.2.15 Let $L = \{l\}$ and $H = \{h\}$. Consider the process

$$P = (l \to P) \sqcap (h \to P)$$

This process is failures-invariant; after any number of h events the process can always nondeterministically choose to accept or refuse an l event. When an h event is available the high-level user can choose to delay communicating it for as long as required, hence deadlocking the low-level user's view of the process.

(End of Example)

Failures-invariance also rejects some processes that are secure. If the process has nondeterminisms where some resolutions lead to L labelled events whilst others lead to H labelled events, there is an obligation for the L labelled events to be available after the H labelled events. This is due to the equivalence of the various low-level purges.

Example 4.2.16 Let $L = \{l\}$ and $H = \{h\}$. Consider the process

$$P = (l \to \mathit{Stop}) \sqcap (h \to \mathit{Stop})$$

This process does not satisfy failures-invariance; $\langle h \rangle \upharpoonright L = \langle\rangle \upharpoonright L$ but after the empty trace an l event may be available whereas after the trace $\langle h \rangle$ no events are available. This process is secure because the low-level user can deduce no information about the behaviour of the high-level user.

(End of Example)

4.3 Bisimulation Non-Deducibility on Compositions

Wittbold and Johnson's non-deducibility on strategies condition [WJ90] was discussed on page 58. Recall that this condition was the authors' way of modifying non-deducibility so that some account was made of high-level outputs. A high-level user interacts with the system and is aware of previous inputs and outputs; the decision as to which inputs follow is a *strategy* based on this information. For a system to satisfy non-deducibility on strategies the low-level user must be unable to deduce any information, irrespective of the strategy used by the high-level user.

Non-deducibility on strategies was put into a CCS formulation by Focardi and Gorrieri in [FG93b]. Their version is called *non-deducibility on compositions* and is based on the compositional approach rather than the traces one. This formulation is less restrictive than the original formulation which was very strict with respect to when inputs and outputs occurred. If the low-level system views are unaltered by any high-level actions, they must all be equal to the process when no high-level actions are prevented. In other words, we check to see that the low-level system view is not modified by the composition of any high-level process Π. Rewriting their condition in CSP gives:

Definition 4.3.1 P satisfies *non-deducibility on compositions* (NDC) if and only if

$$\forall \Pi \text{ where } \Sigma\Pi \subseteq H \bullet P \setminus H =_{\mathrm{T}} (P \underset{H}{\|} \Pi) \setminus H$$

Non-deducibility on compositions is far harder to deal with than the other properties we have looked at, since it requires composition with every possible high-level process. It is potentially very time consuming, assuming it is going to be possible, which may not be the case. A static characterisation is needed, i.e. a property which does not require composition with all high-level processes, but one that implies, or is equal to, NDC.

It is proved in [FG93a] that NDC is equal to SNNI (see Definition 4.2.7). In other words, for non-deducibility on compositions to hold it is only necessary to check that the low-level user views the system when the high-level user is always willing to communicate as identical to the system where the high-level user is never willing to communicate. This is perhaps surprising, since the two conditions are based on different intuitions; it only holds because of the trace equivalence.

The nondeterministic non-interference conditions, as given in Definitions

4.2.3 (NNI) and 4.2.7 (SNNI), use trace equivalence. It is not too surprising that this equivalence is too weak to fully express the required properties even for deterministic processes, as demonstrated for example in Example 4.2.8, due to the limitations of the traces model.

To detect more information flows it is necessary to use a more sophisticated notion of equivalence. Both non-deducibility on compositions and nondeterministic non-interference can be, and have been, considered with different equivalences. We can take essence of these conditions[9] as giving a class of security properties whose members are distinguished by different equivalences.

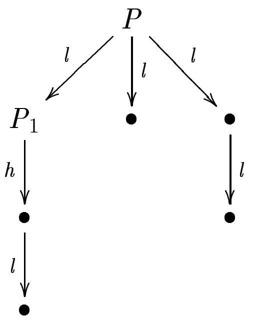
Focardi and Gorrieri's primary condition, *bisimulation non-deducibility on compositions* (BNDC), was introduced in [FG93a], [FG93b] and [FG94]. A system P satisfies BNDC if, for every high-level process Π, a low-level user is unable to distinguish between $(P \underset{H}{\parallel} \Pi) \setminus H$ and $P \setminus H$; the two processes being treated as equivalent if they are weakly-bisimilar.

Definition 4.3.2 A process P satisfies *bisimulation non-deducibility on compositions* (BNDC) if and only if

$$\forall \Pi \text{ where } \Sigma\Pi \subseteq H \bullet P \setminus H \approx_B (P \underset{H}{\parallel} \Pi) \setminus H$$

Example 4.3.3 Let $L = \{l\}$ and $H = \{h\}$. Consider the process

$$\begin{array}{rl} P = & \quad l \rightarrow h \rightarrow l \rightarrow Stop \\ & \Box \ \ l \rightarrow Stop \\ & \Box \ \ l \rightarrow l \rightarrow Stop \end{array}$$



If we compose this process with either the high-level process $Stop_H$ or $h \rightarrow Stop$, then remove the h events, in both cases the resulting process is

[9] i.e. the processes that need to be shown as equivalent.

weakly bisimilar to

$$Q_1 = l \to \mathit{Stop} \,\Box\, l \to l \to \mathit{Stop}$$

If we take $\Pi = h \to \mathit{Stop} \sqcap \mathit{Stop}$ as the high-level process, the resulting process is weakly bisimilar to

$$\begin{aligned} Q_2 = \quad & l \to \mathit{Stop} \,\Box\, l \to l \to \mathit{Stop} \\ \Box \quad & l \to (\mathit{Stop} \sqcap l \to \mathit{Stop}) \end{aligned}$$

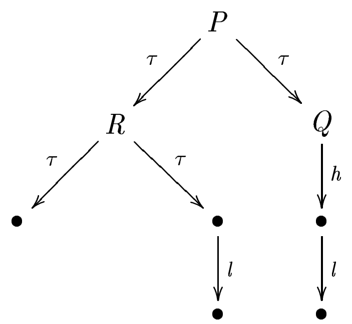
Q_1 and Q_2 are not weakly bisimilar. Q_2 is able to communicate an l event and then nondeterministically decide whether to offer a further l event. In other words, the nondeterministic choice can occur after the first l event; Q_1 can only make the nondeterministic choice during the first l event. The high-level user is only able to influence P at P_1; it makes an observable difference whether the h event at state P_1 is resolved deterministically or nondeterministically. This process fails to satisfy BNDC.

(End of Example)

BNDC is able to detect deadlocks since weak-bisimulation can deal with them. For example, the process in Example 4.2.9 does not satisfy BNDC since composing the process with $h \to \mathit{Stop}$ enables the system to enter a state with no available l events without having communicated an l event, whereas hiding the h events always allows a single l event to occur. Nevertheless the condition can still be satisfied by processes which give the high-level user the opportunity to indefinitely delay, if it is possible for the process to deadlock by some other means.

Example 4.3.4 Let $L = \{l\}$ and $H = \{h\}$. Consider the processes

$$\begin{aligned} P &= Q \sqcap R \\ Q &= h \to l \to \mathit{Stop} \\ R &= \mathit{Stop} \sqcap l \to \mathit{Stop} \end{aligned}$$



This process satisfies BNDC since

$$\begin{aligned} \forall\, \Pi \text{ where } \Sigma\Pi \subseteq H \bullet P \setminus H &\approx_B R \\ &\approx_B (P \underset{H}{\parallel} \Pi) \setminus H \end{aligned}$$

If the process is in state Q it is possible for the high-level user to delay the h event for an arbitrary amount of time. This is different to state R where the process will either deadlock or offer the event l, but cannot deadlock for some period of time and then offer an l. The process can be viewed as insecure because of such a high-level option; i.e. the system can go from a stable state that refuses l events to one that accepts them, only when the high-level user participates.

(End of Example)

Similarly the condition can be satisfied by a process that can be deadlocked by high-level inaction or deadlock after the high-level event.

Example 4.3.5 Let $L = \{l\}$ and $H = \{h\}$. Consider the process

$$P = h \rightarrow (l \rightarrow \mathit{Stop} \sqcap \mathit{Stop}) \sqcap (l \rightarrow \mathit{Stop})$$

This process satisfies BNDC since its composition is always bisimilar to $l \rightarrow \mathit{Stop} \sqcap \mathit{Stop}$. It suffers the same problem as the process in Example 4.3.4 since the h event can be delayed for an arbitrary amount of time.

(End of Example)

Once again it is desirable to have a static characterisation so that is it not necessary to compose with every high-level process to check BNDC. This is still an open problem. In [FG95] the *strong bisimulation strong nondeterministic non-interference* (SBSNNI) condition was proposed. This is a sufficient condition for BDNC which avoids universal quantification. SBSNNI is defined using a similar condition, *bisimulation strong nondeterministic non-interference* (BSNNI). This in turn is the SNNI condition when the trace equivalence has been replaced with weak-bisimulation.

Definition 4.3.6 A process P satisfies *bisimulation strong nondeterministic non-interference* (BSNNI) if and only if

$$P \setminus_H \approx_B P \setminus H$$

BSNNI is not a sufficiently strong security condition. Despite using an equivalence that can cope with deadlocks, since it looks at the process with the H labelled events restricted, it cannot detect some deadlocks that can be caused by the high-level user after such an event. For example, the process in Example 4.2.9 which can exhibit deadlock because the second h event is delayable, satisfies both SNNI and BSNNI. Moving to a stronger equivalence has not solved the problem.

This condition can detect potential deadlocks that are caused by delaying an initial high-level event. Every high-level transition has some state from which it is available without any other high-level events needing to be communicated. The stronger condition SBSNNI is built by ensuring the BSNNI condition holds for every reachable state in the process; i.e. it exploits local information.

Definition 4.3.7 A process P satisfies *strong bisimulation strong nondeterministic non-interference* (SBSNNI) if and only if

$$\forall\, P', \gamma \text{ such that } P \stackrel{\gamma}{\Longrightarrow} P' \bullet P' \setminus_H \approx_B P' \setminus H$$

Example 4.3.8 Let $L = \{l_1, \ldots, l_n\}$ and $H = \{h_1, \ldots, h_n\}$. Consider the process

$$P = (\Box\, h_i \rightarrow l_i \rightarrow P) \triangleright \sqcap\, l_i \rightarrow P$$

This process is insecure since the high-level user has the opportunity of communicating an arbitrary amount of information to the low-level user. It satisfies SBSNNI since the only state with H labelled events is equivalent to $\mu\, X. \sqcap\, l_i \rightarrow X$ whether we hide or restrict H labelled events.

(End of Example)

BSNNI and SBSNNI can both be used in proving that BNDC holds. SBSNNI is shown to be sufficient for BNDC in [FG95] and BNDC is shown to be sufficient for BSNNI in [FG94]. In other words:

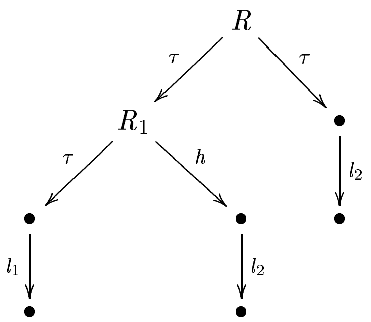
§Theorem 4.3.9 *SBSNNI* $\subset$ *BNDC* $\subset$ *BSNNI*.

The success of SBSNNI implies the success of BNDC, and the failure of BSNNI the failure of BNDC. The inclusion of these conditions is strict; in other words, it is not true that the failure of SBSNNI implies the failure of BNDC, or that the success of BSNNI implies the success of BNDC. The process in Example 4.2.9 has already been shown to satisfy BSNNI and fail BNDC. The proof that $SBSNNI \subset BNDC$ is a strict inclusion as given in [FG95] and [FG97] is incorrect; it uses Example 4.3.3 as a process that satisfies BNDC but not SBSNNI, whereas it does not satisfy either.

Lemma 4.3.10 $SBSNNI \neq BNDC$

Proof Let $L = \{l_1, l_2\}$ and $H = \{h\}$. Consider the process

$$\begin{aligned} R = & \quad ((h \to l_2 \to Stop) \triangleright l_1 \to Stop) \\ \sqcap & \quad l_2 \to Stop \end{aligned}$$



This process fails to satisfy SBSNNI. R fails SBSNNI because state R_1 fails BSNNI; $\langle l_2 \rangle \in traces\, \llbracket P_1 \setminus H \rrbracket$ but $\langle l_2 \rangle \notin traces\, \llbracket P_1 \setminus_H \; \rrbracket$. The process does satisfy BNDC since

$$\begin{aligned} \forall\, \Pi \text{ where } \Sigma\Pi \subseteq H \bullet R \setminus H \quad & \approx_B \quad (l_1 \to Stop) \sqcap (l_2 \to Stop) \\ & \approx_B \quad (R \underset{H}{\parallel} \Pi) \setminus H \end{aligned}$$

□

It is debatable whether we should accept R as a secure process. It satisfies a mixed bunch of the properties. R is not non-deducible since the occurrence of an l_1 event implies no h event has taken place, and similarly does not satisfy the traces based non-deducibility on strategies, but it is non-deducible on compositions. It is not failures-invariant and does not satisfy SBSNNI, but it does satisfy NNI and ultimately BNDC. The argument for R is that, however the high-level user chooses to act, the low-level user can view the system as giving a nondeterministic choice between l_1 and l_2. Conversely, not only does an l_1 event indicate no h event has taken place, but at state R_1 the high-level user might have the opportunity to prevent the low-level user ever receiving an l_1 event. Although the τ event at R_1 might occur before the high-level user has the opportunity to act, it might represent an internal action which will always give the high-level user time to act; this is certainly the case if we think of R_1 as being an implementation of the time-out operator.

In general we can take any process P and use it to construct a process P' that satisfies BNDC and is able to behave like P [For97]. For example, letting P' be a process which nondeterministically chooses to act like P, or like any of the potential low-level system views. Since P' can behave as P, it will be insecure whenever P is. In particular, whenever it allows the high-level user to do anything it will behave like P. This is similar to the idea behind the $\mathit{LEAK} \sqcap \mathit{Chaos}_L$ process in Example 3.3.4.

Theorem 4.3.11 Let P be a process. If $S = \{(P \underset{H}{\parallel} Q) \setminus H \mid \Sigma Q \subseteq H\}$[10] then $P' = (\bigsqcap S) \sqcap P$ satisfies BNDC.

Proof We need to show $\forall\, \Pi$ where $\Sigma\Pi \subseteq H$ that $P' \setminus H \approx_B (P \underset{H}{\parallel} \Pi) \setminus H$. Now

$$
\begin{array}{ll}
 & P' \setminus H \\
\approx_B & \hfill [\ \text{Definition}\ ] \\
 & ((\bigsqcap S) \sqcap P)) \setminus H \\
\approx_B & \hfill [\ \text{No process in } S \text{ contains any } H \text{ labelled actions}\ ] \\
 & (\bigsqcap S) \sqcap P \setminus H \\
\approx_B & \hfill [\ P \equiv P \underset{H}{\parallel} \mathit{Run}_H\ ] \\
 & (\bigsqcap S) \sqcap (P \underset{H}{\parallel} \mathit{Run}_H) \setminus H \\
\approx_B & \hfill [\ (P \underset{H}{\parallel} \mathit{Run}_H) \setminus H \in S\ ] \\
 & \bigsqcap S
\end{array}
$$

[10] S is quotiented by $\approx_B$. In other words, if two processes are equivalent they cannot both be in S.

$$
\begin{array}{ll}
& (P' \underset{H}{\parallel} \Pi) \setminus H \\
\approx_B & \hfill [\text{ No process in } S \text{ contains any } H \text{ labelled actions }] \\
& (\sqcap S) \sqcap (P \underset{H}{\parallel} \Pi) \setminus H \\
\approx_B & \hfill [\ (P \underset{H}{\parallel} \Pi) \setminus H \in S\ ] \\
& \sqcap S
\end{array}
$$

Therefore, $P' \setminus H \approx_B \ \sqcap S \approx_B (P \underset{H}{\parallel} \Pi) \setminus H$ so P' satisfies BNDC. □

Example 4.3.12 Let $L = \{l_1, \ldots, l_n\}$ and $H = \{h_1, \ldots, h_n\}$. Consider the process

$$P = \Box\, h_i \rightarrow l_i \rightarrow P$$

This process is clearly insecure because the high-level user's behaviour is mirrored directly to the low-level user. This process does not satisfy BNDC, although a modification of it was shown in 4.3.8 which satisfies SBSNNI and by implication BNDC. Alternatively we can construct a P', according to the formula in Theorem 4.3.11 which may choose to behave like P and satisfies BNDC.

(End of Example)

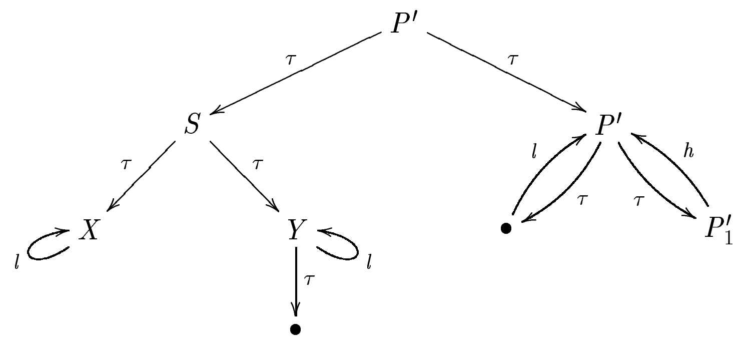
If S is not finite, P' is not necessarily a practical counterexample. An infinitely nondeterministic system is not necessarily fully implementable; the discussion in [Ros98, Section 10.3] points out that "Any implementation would have to include a mechanism which guaranteed to select between an infinity of choices in a finite time." The existence of an infinite number of high-level processes does not imply S is infinite.

Example 4.3.13 Let $L = \{l\}$ and $H = \{h\}$. Consider the process

$$P = (l \rightarrow P) \sqcap (h \rightarrow P)$$

as defined in Example 4.2.15. P does not satisfy BNDC since composing with $Stop_H$ is not equivalent to composing with Run_H. Now

$$
\begin{array}{lcl}
S & = & \{\mu X.(l \rightarrow X), \mu Y.(l \rightarrow Y \sqcap Stop)\} \\
P' & = & (\sqcap S) \sqcap P
\end{array}
$$



The process P' satisfies BNDC by Theorem 4.3.11. The process is not secure for the same reason P is not secure; when an h event is available in state P'_1, the high-level user can choose to delay communicating it for as long as required, hence deadlocking the low-level user's view of the process. (End of Example)

For non-divergent deterministic processes we can compare these conditions with the determinism based eagerly secure and lazily secure conditions. We can actually consider a larger variety of processes, namely those that satisfy $LowDet_L(P)$. This covers a large number of interesting processes, including those that appear deterministic to the low-level user, since determinism implies low-determinism (see Lemma 3.1.10) as does independence (see Lemma 3.1.9).

Proposition 4.3.14 If P is a low-deterministic process and BNDC holds then P is eagerly secure.

Proof We need to show that $P \setminus H$ is deterministic.

$$\begin{array}{ll} & \text{BNDC holds} \\ \Rightarrow & \hfill [\text{ Theorem 4.3.9 }] \\ & \text{BSNNI holds} \\ \Rightarrow & \hfill [\text{ Definition 4.3.6 }] \\ & P \setminus H \;\approx_B P \setminus_H \end{array}$$

Since P is low-deterministic, $P \setminus_H$ is deterministic by definition. Hence, since weak-bisimulation preserves determinism, $P \setminus H$ is deterministic as required. □

Versions of the non-deducibility on compositions and nondeterministic non-interference conditions have also been considered using a failures equivalence [Foc96]. It is shown that Theorem 4.3.9 can be extended to the new failures based SFSNNI, FNDC and FSNNI properties. These conditions have then been used by the authors to show that BNDC, SBSNNI and lazy security agree for low-deterministic processes.

§Theorem 4.3.15 If P is a non-divergent, low-deterministic process then BNDC holds if and only if P is lazily secure. Similarly for SBSNNI.

Neither BNDC nor SBSNNI (see Example 4.3.8) appear to be strict enough, since they incorrectly identify some insecure processes as secure. Both conditions allow the resolution of nondeterminisms by the high-level user, which can give the low-level user a means of making deductions. The problem with BNDC seems particular severe, since we can construct a supposedly secure process which is capable of leaking whatever information the high-level user requires to the low-level user.

None of the security conditions considered in this chapter appear to be adequate. It is not even the case that all the conditions coincide for deterministic processes; for those that disagree with the lazy security condition we have demonstrated that there are processes, where the low-level user can identify whether the high-level user is active or inactive, which are incorrectly identified as secure. For those conditions that agree with the determinism based conditions, we have demonstrated that there are questionable nondeterministic processes which are incorrectly identified.

Chapter 5

Local Non-Interference

A variety of different security conditions have been given in the previous two chapters which endeavour to show whether a process P is free from undesirable information flow. This chapter discusses a new family of conditions, *local non-interference*, first introduced in [For97]. These conditions are intended to be akin to Goguen and Meseguer's non-interference [GM82] and also applicable for nondeterministic processes.

Section 5.1 provides the motivation for the new conditions by considering several properties that are desirable in a security condition. Examples are given for the properties that have not already been discussed at length in the previous chapters. The successes and failures of the previously considered conditions are then given in relation to these properties.

Section 5.2 introduces the new conditions and looks at how they behave on some of the previously troublesome examples. A strong condition and a weak condition are given, depending on how multiple H labelled transitions are considered, and formulations based on weak bisimulation and failures/divergences are examined.

Some elementary properties for the local non-interference conditions are discussed in Section 5.3. Finally, in Section 5.4, the relationship between these conditions is given for both deterministic and nondeterministic processes.

5.1 Motivation

The different security conditions which have previously been discussed, disagree over which processes should be considered secure and insecure. It is perhaps unreasonable to introduce a new condition based purely around such a disagreement since arguments can be given both ways for many processes. There are however several motivating properties that an information flow security condition might be reasonably expected to achieve. Unfortunately it is not the case that every proposed condition meets these properties; indeed, every security condition that has been considered in Chapter 3 and Chapter 4 fails some property. The basic motivation for the *local non-interference* conditions is satisfaction of the following properties:

i. Processes that are not independent in L should not automatically be labelled insecure.

ii. The low-level user should not be able to deduce which choices have, or have not, been made by the high-level user.

iii. The high-level user should not be able to pass any information to the low-level user.

iv. High-level action or inaction should not be able to delay the low-level user, temporarily or permanently.

v. A secure process should not be able to get into a state where it becomes insecure.

There are other desirable properties which are less fundamental to the underlying ideal of information flow. These are properties which are influenced by the expectation of how a security condition should behave.

vi. The security condition should be compositional[1].

vii. For deterministic processes there should be agreement with the lazy security condition as to which processes are secure.

[1] The compositional property is also referred to by 'composable' and 'hook-up' in the literature.

Furthermore, and in keeping with the problem statement given in Chapter 1, we are looking for an asymmetric condition which is for a system with two users. The system is also an untimed one, although the ability of users to distinguish states by their available events permits the detection of, for example, the delays mentioned in property (iv).

Properties which deal with deterministic (and low deterministic) processes have been discussed in the preceding chapters and such conditions seem to be well understood. Determinism is not a prerequisite for security; as mentioned previously there are certainly nondeterministic processes that it would seem reasonable to class as secure, such as any process that only communicates L labelled events. The search for secure nondeterministic processes has led to many different characterisations and is the central theme of this thesis. The lazy security condition, which is otherwise highly satisfactory, is not designed for nondeterministic processes.

The second of these seven properties is similar to the non-deducibility ideal. It seems reasonable that if the low-level user can deduce that the high-level user has performed or refused a given event, the system is insecure and information can flow. This property is weaker than the non-deducibility condition since it is only applicable when the H labelled events have been offered to the high-level user. In other words, it is not a basic requirement that the low-level user cannot deduce information about the availability of H labelled events when such availability is not due to high-level behaviours. It does not seem to constitute a flow of information from high *to* low if, for example, such a deduction is due to the low-level user influencing the system and altering the available high-level events. It can be seen from Definition 4.1.4 and Definition 4.1.10 that arguing for unexpurgated non-deducibility argues against asymmetric information flow.

Example 5.1.1 Let $L = \{l_1, l_2\}$ and $H = \{h_1, h_2\}$. Consider the process

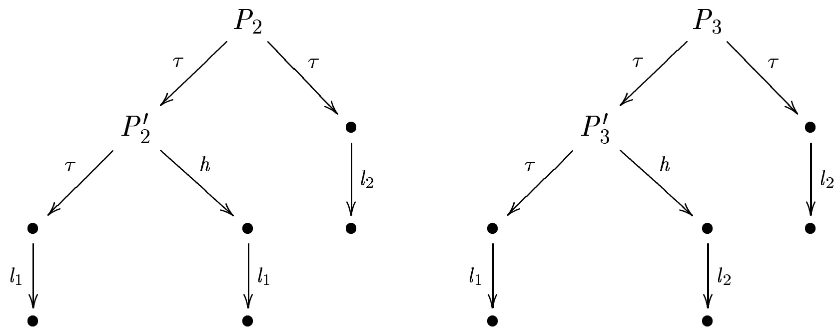
$$P_1 = l_1 \to h_1 \to Stop \,\Box\, l_2 \to h_2 \to Stop$$

The low-level user can make several deductions about the high-level user's available events: before a low-level event no H labelled event is available and after an l_i event, h_i becomes available. The change in available high-level events is caused by the low-level user and the only high-level choice, whether or not to communicate an h_i when one is available, cannot be deduced by the low-level user.

This process should be treated as secure since the only deductions about H labelled events are those due to the behaviour of the low-level user. P_1 is classed as secure by all the conditions other than the two afore-mentioned non-deducibility ones.

(End of Example)

Example 5.1.2 Let $L = \{l_1, l_2\}$ and $H = \{h\}$. Consider the processes P_2 (P from Example 4.1.5) and P_3 (R from Lemma 4.3.10)



$$
\begin{array}{lcl}
P_2 & = & ((h \to l_1 \to \mathit{Stop}) \rhd l_1 \to \mathit{Stop}) \sqcap l_2 \to \mathit{Stop} \\
P_3 & = & ((h \to l_2 \to \mathit{Stop}) \rhd l_1 \to \mathit{Stop}) \sqcap l_2 \to \mathit{Stop}
\end{array}
$$

The arguments for the security (or insecurity) of these processes have already been given in the relevant sections; we would like to classify P_2 as secure and P_3 are insecure. The important distinction to be made between the processes is the nature of the deductions the low-level user is able to make. In process P_2 the deduction occurs when an l_2 event becomes available; the deduction, that an h event has not occurred, is not an indication of the high-level user's behaviour, rather how an internal nondeterminism has been resolved. In P_3 the deduction occurs when an l_1 event is available; in this case there has been an opportunity for an h to be communicated and the deduction that no such event has occurred is cogent.

BNDC does not distinguish between these two processes and classifies them both as secure. They are seen as giving the low-level user a nondeterministic choice between l_1 and l_2 without distinguishing how the high-level user can affect it.

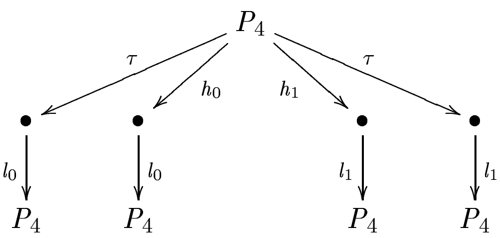
(End of Example)

The high-level user should not be able to resolve any low-level nondeterminisms. In P_3 the high-level user might have the opportunity to resolve such a nondeterminism. More importantly this re-resolves the previous nondeterminism once the internal choice has already been made. In P_2 the high-level user does not affect the existing internal choice. Deductions from such resolutions are perhaps clearer when the high-level action takes place at the same time as the nondeterministic choice, as in Example 4.1.11.

The third property, that the high-level user should not be able to pass information to the low-level user, is also susceptible to the resolution of nondeterminisms. This property is akin to the non-interference framework and it is the flow of information from high to low that is the underlying system behaviour we are trying to prevent. For nondeterministic systems it is not enough to say the low-level system view after some high-level events is a possible view before such events. Example 4.1.8 shows the high-level user passing information to the low-level user, the non-deducibility condition being satisfied because the high-level user's actions are obfuscated by the outcome of a nondeterminism. Similarly Example 4.2.14 demonstrates how an information flow can be acceptable to failures-invariance and Theorem 4.3.11 demonstrates it for BNDC.

Example 5.1.3 Let $L = \{l_0, l_1\}$ and $H = \{h_0, h_1\}$. Consider the process

$$\begin{aligned} P_4 = \quad & \tau.l_0.P_4 + h_0.l_0.P_4 \\ + \quad & h_1.l_1.P_4 + \tau.l_1.P_4 \end{aligned}$$



If an l_i event is available the low-level can deduce that an h_{1-i} event has not just been communicated. Furthermore, if the internal actions always give the high-level user the opportunity to act, there is an obvious means of communication since the low-level events mirror the high-level events. Even if this is not the case, each h_i action of the high-level user will be observable in the corresponding l_i action available to the low-level user. We can expand P_4 to a similar process over a far larger alphabet, for example $H = \{h_a, \ldots, h_z\}$ and $L = \{l_a, \ldots, l_z\}$; the low-level user might be expected to distinguish a coherent high-level message from any noise generated by the system.

This process satisfies SBSNNI and hence BNDC. It does not satisfy failures-invariance since different low-level events are accepted and refused after an h_0 or h_1 event.

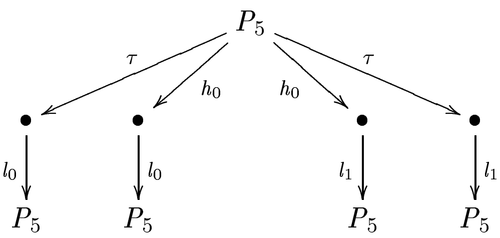
(End of Example)

There are two potential flows of information in P_4 because different high-level behaviours resolve the nondeterministic choice in different ways. The fact that two different choices can lead to differing low-level system views is against both the non-deducibility and non-interference paradigms, especially since the results of these high-level choices is deterministic.

It is important to distinguish between different labelled high-level events. The deductions that the low-level user is able to make with process P_4 are that the high-level user has not communicated certain events, rather than whether the high-level user has communicated or not[2]. If we prevent such inferences on the availability of high-level events, the process can be made secure.

Example 5.1.4 Let $L = \{l_0, l_1\}$ and $H = \{h_0, h_1\}$. Consider the process

$$\begin{aligned} P_5 = \quad & \tau.l_0.P_4 + h_0.l_0.P_4 \\ + \quad & h_0.l_1.P_4 + \tau.l_1.P_4 \end{aligned}$$



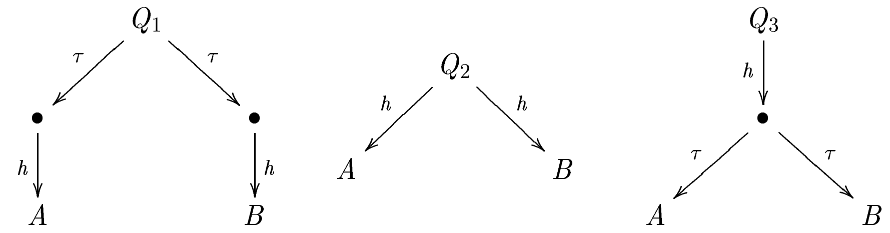
The low-level user is unable to deduce anything from the availability of an l_i event other than the fact that an h_1 is not available. Since this event is never available and this is not due to any choice by the high-level user, it is hard to argue for this being an information flow. Furthermore, the high-level user cannot influence the low-level user by the communication of an h_0 event since there is no control over the resolution of the nondeterminism.

(End of Example)

[2]This is just considering for the moment the fact that every low-level trace is available with or without the corresponding high-level intervention, rather than any reflected 'observations' of high-level events.

The intervention of the high-level user in P_5 causes the nondeterminism to be resolved in a different place. The important point is that the high-level user is not resolving the nondeterminism or offering different resolutions. In other words, so long as the result of the high-level user's actions does not influence the low-level user's view of the system, it does not matter what is done, including causing a nondeterministic choice.

If the occurrence of an H labelled event is going to cause some nondeterministic choice there are three ways we might look at it[3].



before *during* *after*

Firstly we can view the nondeterministic choice as taking place before the high-level event and the occurrence of the event leading to the resolution of the choice. Secondly we might view the choice as happening while the event is being communicated. Finally the nondeterministic choice can be considered as occurring after the high-level event's communication. (These are the different ways we might implement a nondeterministic choice around an H labelled event; once written it is immutable.)

It is necessary for these three views of nondeterministic choice to be treated differently since they behave differently. Resolving the nondeterminism before the high-level event, as in Q_1, is the most susceptible to problems. It gives the high-level user the choice as to whether to communicate, as well as how to communicate[4], based on the system's behaviour. The problems in Example 4.1.8 and Example 4.2.14 are due to such behaviours. Q_2 and Q_3 are not susceptible to such high-level manipulation since the nondeterminism is still to be resolved at the point when the h event is communicated and the high-level user cannot base a decision on how it will be resolved. It is the change from a 'before' type choice in Example 4.2.14 to the 'during'

[3] These three types of nondeterminism are equivalent under the failures-divergences model. Under weak-bisimulation they are all different.

choice in P_5 that transforms an insecure process into a secure one.

The distinction between Q_2 and Q_3 is not so great since neither allows the high-level user to influence the nondeterministic choice. The difference becomes clearer when we consider the effects of interleaving the process with another. Interleaving with Q_3 allows the interleaved choice to happen on either side of the nondeterministic choice (once the h event has taken place). In Q_2, since the act of communicating an h event resolves the nondeterminism, the interleaved choice can occur before or after the h event, but not after the h event and before the nondeterministic choice. This distinction is discussed further in Section 6.4.

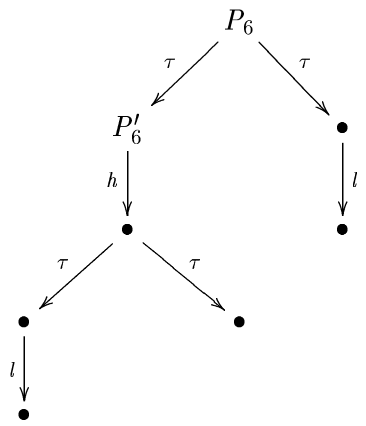
The fourth property, that the high-level user should be unable to delay the low-level user, has been discussed at length in the preceding sections. The types of cases we wish to classify as insecure are those where the high-level user is required to communicate for low-level events to be available (such as Example 4.1.3 and Example 4.3.3) and where a high-level event leads to the system deadlocking (such as Example 4.2.8). As has been shown in Example 4.2.15 and Example 4.3.4 it is not sufficient to just use an equivalence which is able to show liveness properties. Similarly, conditions that only deal with traces, such as trace-invariance (Definition 4.2.10), are not necessarily unable to detect such problems.

The fifth property requires that a secure process cannot become an insecure process. If a system can start to behave in an insecure manner at some stage in its operation it would seem reasonable to call the entire system insecure. A low-level user who starts the system in such a state, or starts considering the system from this state, will be able to make use of the insecurity. In particular, a high-level (or low-level) user might be expected to exploit any insecurity which arises in a system. The low-level user may be unable to guarantee that they are receiving high-level information, but they may still be receiving it. A good demonstration of this is given in Theorem 4.3.11, where a 'secure' (BNDC) process can be built that behaves like an insecure process whenever the high-level user is given the opportunity to act.

Example 5.1.5 Let $L = \{l\}$ and $H = \{h\}$. Consider the process

[4] If there are several events available to the high-level user.

$$P_6 = \quad h \rightarrow (l \rightarrow Stop \sqcap Stop) \\ \sqcap \quad l \rightarrow Stop$$



State P_6' is insecure (and viewed as such by all the nondeterministic conditions) since the h event is required before any low-level event can be communicated. The process as a whole is insecure because the high-level user can exploit the delay once the system is in this state. Indeed, if the low-level user is unable to communicate for some arbitrary period of time and then an l event becomes available, the only possibility is that the high-level user has communicated an h event.

P_6 satisfies failures-invariance since the low-level refusals before and after the h event are the same. BNDC is satisfied because the composition with any high-level process is bisimilar to $l \rightarrow Stop \sqcap Stop$. (Non-deducibility only fails to be satisfied because the h event can never occur after the l event.)

(End of Example)

Example 4.2.14, Example 4.2.15 and Theorem 4.3.11 show that this property is not met by several of the security conditions. An argument against this property is that the low-level user may require the knowledge that the system is in an insecure state to deduce / observe any information. In all three examples the insecurity is deemed acceptable to the relevant conditions because a different internal system behaviour leads to a state with complementary available events. Nevertheless, this argument is not necessarily true; in the given examples the low-level user does not require knowledge about the system state and it is easy to argue for the insecurity of these processes. Example 4.2.14 would allow a co-operating pair of users to communicate any message without error and the covert channel is always available, Example 4.2.15 may give the high-level user the opportunity to communicate and Theorem 4.3.11 allows the construction of a process that may pass information in any arbitrary manner.

The sixth property, that the security condition should be compositional, means that the result of composing two secure processes should be another secure process. We might expect a condition to be compositional because each system is effectively using the other system and any use of a secure system should allow no undesirable information flow. Since neither system should be behaving in an insecure manner it would be reasonable to assume there is security.

The assumption that a secure system could be built up from secure components, was a tacit one that turned out to be false for several security conditions [McC87]. For example, McCullough demonstrated that neither non-interference or non-deducibility could be composed. Two systems with small problems, which were not sufficient to fail the conditions, could be combined to create a much larger, significant problem. The failures with conditions that are not compositional can be traced to the fact that combining processes allows the communication of information that the users cannot exploit in the individual processes; Example 4.1.9 has the systems communicating some high-level parity which is not observable, but which allows the combined process to synchronise.

There are several additional reasons why it is desirable for a system to be compositional. Firstly it is useful at the development stage. When building large systems it is often easier to split them up into subsystems; concurrency is based on such independent processes which then interact. Compositional conditions allow proofs to be built up along with such development. It is also easier to perform such proofs because the components may be smaller. Automatic model checking tools, such as FDR [FSEL], rely on enumerating the state space of the system; we are more likely to be able to verify a compositional condition, since it can be split into smaller components, and hence potentially check much larger systems.

Secondly not all systems and components are designed to be used on their own. The composition of such systems is likely in a 'real' environment. This might be a network of such systems, but it could equally well be separate parts (e.g. peripherals) of a single system. Components that are labelled as 'secure' may be expected to behave as secure when they are correctly used, irrespective of how they are connected up. This might not be to the types of systems original envisioned; they may, for example, be used as 'off the shelf' components.

The final property, that there should be agreement on which deterministic process are viewed as secure, has two foundations. Firstly there are good reasons for believing that the determinism based conditions (see Chapter 3) correctly identify which deterministic processes are secure. For deterministic processes many of the difficult questions, such as which model to use, which equivalence to use and how nondeterminism is interpreted, are removed. Secondly many of the nondeterministic conditions, such as invariance and BNDC (see Chapter 4), have been shown to agree on the deterministic processes.

None of the nondeterministic security conditions in Chapter 4 satisfy all of these security properties. Indeed, all of the considered conditions fail a variety of the properties. The earlier properties are particularly worrying because their failure implies there is information flow from the high-level user to the low-level user. This is highlighted by the failure of non-deducibility, failures-invariance and BNDC to satisfy property (iii).

5.2 Local Non-Interference

The key idea of non-interference, as given by Goguen and Meseguer [GM82], is that the high-level user should have no effect on what the low-level user can observe. One problem with the nondeterministic treatments of non-interference has been the meaning of "no effect". It has been shown earlier, in Example 5.1.3, that it is not sufficient for all of the low-level options to still be available after different H labelled events if the high-level user is able to manipulate which of those options are available.

If there is no information flow from high to low in a system, the low-level user's view of the system must be totally unaffected by the high-level user's actions. In particular, for a nondeterminism this means there should be no discernible difference between the nondeterministic choices available to the low-level user because of any high-level behaviour. One way of specifying this is to say that whenever an H labelled transition takes place, the process is taken to a state that is indistinguishable (to the low-level user) to the state the process was in before the transition. The interpretation of nondeterminism being used here is a possibilistic one. We require the high-level user to keep exactly the same possibilities available to the low-level user, at least to the level of being indistinguishable under our chosen equivalence. This corresponds to a literal interpretation of the transition system where no refinement is permitted.

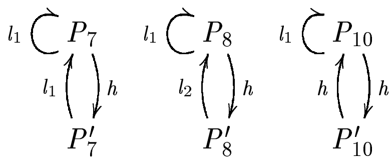
Given a state σ in a process that has a $t \in H$ labelled transition to a state σ', we can consider what the process σ can do before communicating an event in H; this is the process $\sigma \setminus_H$ where we are restricting the process to only allow L labelled events. Since we require H labelled transitions to take us to states that are indistinguishable to the low-level user, it is necessary that this is equivalent to what the process σ' can do before communicating any events in H, i.e. $\sigma' \setminus_H$.

The idea behind *strong local non-interference* is to ensure that there is an equivalence between every two restricted states that have an H labelled transition between them. This ensures no high-level event resolves any non-determinism which may be present or alters the low-level view of the system by leading to a state with different low-level events. The condition is called *local* since a given equivalence takes no account of any preceding states and *strong* because it is required for every H labelled transition.

Definition 5.2.1 Let P be a process whose alphabet is partitioned by H and L. P satisfies *strong local non-interference*, written $\mathsf{SLNI}_L(P)$, if $\sigma \xrightarrow{h} \sigma'$, where $h \in H$ and $\sigma, \sigma' \in \text{states}(P)$, implies $\sigma \setminus_H \approx_B \sigma' \setminus_H$.

Example 5.2.2 Let $L = \{l_1, l_2\}$ and $H = \{h\}$. Consider the processes

$$\begin{aligned} P_7 &= l_1 \to P_7 \,\Box\, h \to l_1 \to P_7 \\ P_8 &= l_1 \to P_8 \,\Box\, h \to l_2 \to P_8 \\ P_9 &= l_1 \to P_9 \,\Box\, h \to P_9 \\ P_{10} &= l_1 \to P_{10} \,\Box\, h \to h \to P_{10} \end{aligned}$$



P_7 satisfies strong local non-interference because there is only one state with an h event, the initial state, and $P_7 \setminus_H = \mu X.l_1 \to X = P_7' \setminus_H$.

This process is secure because there is always an l_1 event available for the low-level user to communicate. Similarly $\mathsf{SLNI}_L(P_9)$ holds since the h event returns the process to the state it came from and hence the low-level view of the system is unaffected.

P_8 does not satisfy strong local non-interference; $P_8 \setminus_H = \mu X . l_1 \to X$ whereas $P_8' \setminus_H = l_2 \to \mu X . l_1 \to X$. The low-level user can deduce whether the high-level user has acted based on the available l_i event.

P_{10} is the canonical example from Chapter 4. The process is insecure because the high-level user can delay the low-level user on every alternate h event. $\mathsf{SLNI}_L(P_{10})$ does not hold because $P_{10} \setminus_H = \mu X . l_1 \to X$ but $P_{10}' \setminus_H = \mathit{Stop}$. This indicates that odd h events can deadlock the system and even h events are required to prevent deadlock.

(End of Example)

As demonstrated by P_{10}, strong local non-interference will not classify as secure a process which allows high-level action (or inaction) to delay the low-level user. In any such process the states before and after the H labelled transition will be distinguishable; if they were not distinguishable the transition would not be necessary (or vice versa) for the availability of L labelled events and so there would be no possibility of delaying the low-level user. This remains true if both the delayed and non-delayed states are available elsewhere in the system since the condition only considers the local states.

Example 5.2.3 Let $L = \{l\}$, $H = \{h\}$ and consider the process

$$P = (h \to l \to \mathit{Stop}) \sqcap (\mathit{Stop} \sqcap l \to \mathit{Stop})$$

as given in Example 4.3.4. This process is shown as satisfying BNDC although the high-level user can delay the low-level user for an arbitrary amount of time. $\mathsf{SLNI}_L(P)$ is not satisfied because $Q \setminus_H$, the restricted state before the h event, is Stop but the state after the h event can communicate an l. The potential deadlock or l event after state R are not considered since they are not local.

(End of Example)

The equivalence used in Definition 5.2.1 is weak bisimulation. Since $P \approx_B Q$ implies $P =_{\mathrm{FD}} Q$ (which in turn implies $P =_{\mathrm{T}} Q$) for non-divergent processes (Theorem 5.4.2), this makes the conditions stricter than their failures/divergences (or traces) counterparts. The stricter the equivalence, the more processes that can be differentiated, so a user who observes the system recognising a stricter equivalence can gain more information that one observing it with a weaker equivalence. We want processes to be secure against the strongest possible users, so the weak bisimulation is primarily used. For example, as discussed in Section 3.3, failures/divergences does not differentiate between different views of nondeterminism and so cannot detect some information flows that are available to a user who can take snapshots of a process.

Strong local non-interference was defined using weak bisimulation for two additional reasons. Firstly it does not have a catastrophic view of divergence; a τ-loop is viewed as being executed any finite number of times. Without modelling infinite traces this interpretation allows the modelling of systems such as channels that can lose messages any finite number of times. A further advantage of weak bisimulation is that it can be tested in polynomial time, whereas testing failures/divergences is PSPACE-complete [KS90, PT87, Fer89].

The concept of local non-interference can be extended to other equivalences, giving a family of different security conditions. Of particular value is the condition where weak bisimulation equivalence is replaced by the failures/divergences one. This formulation is particularly useful when comparing the strong local non-interference properties with the lazy security property.

Definition 5.2.4 Let P be a process whose alphabet is partitioned by H and L. P satisfies *strong FD local non-interference*, written $\mathsf{SLNI}_L^{FD}(P)$, if $\sigma \xrightarrow{h} \sigma'$, where $h \in H$ and $\sigma, \sigma' \in \text{states}(P)$, implies $\sigma \setminus_H =_{\text{FD}} \sigma' \setminus_H$.

We can consider how strong local non-interference deals with some of the nondeterministic examples which raised problems in the previous sections. Strong local non-interference looks at a process at the points where H labelled transitions occur and does not consider how any previous nondeterminisms have been resolved. The high-level user is never given the chance to re-resolve any nondeterministic choices and so must ensure that the current, local state is preserved.

Example 5.2.5 Let $L = \{l_1, l_2\}$, $H = \{h\}$ and consider the processes P_2 and P_3 from Example 5.1.2; P_2 is a process we would view as secure whereas P_3 is insecure. The only state we need to check for P_2 is P_2', since this is the only state with an H labelled transition. $\mathsf{SLNI}_L(P_2)$ is satisfied since $\tau.l_1.0$ is bisimilar to $l_1.0$. Once again there is a single state we need to check for P_3, but this time the system after the h event $l_2.0$ is not bisimilar to $P_3' \setminus_H = \tau.l_1$; $\mathsf{SLNI}_L(P_3)$ is not satisfied. A choice between l_1 and l_2 is made in P_3's initial state and by P_3' this choice cannot be changed, if the system is to be secure.

(End of Example)

Example 5.2.6 Let $L = \{l_1, l_2\}$, $H = \{h\}$ and consider the process

$$P = (l_1 \to P \mathrel{\Box} h \to l_2 \to P) \sqcap (l_2 \to P \mathrel{\Box} h \to l_1 \to P)$$

as defined in Example 4.2.14. $\mathsf{SLNI}_L(P)$ is not satisfied because whenever an h event is available, communicating it changes the available low-level event and $l_1 \to P \setminus_H$ is not bisimilar to $l_2 \to P \setminus_H$.

(End of Example)

A high-level event is also prevented from resolving any current or future nondeterministic choice.

Example 5.2.7 Let $L = \{l_1, l_2\}$, $H = \{h\}$ and

$$P_{11} = \tau.l_1.0 + \tau.l_2.0 + h.l_1.0$$

as given in Example 4.1.11. $\mathsf{SLNI}_L(P_{11})$ is not satisfied because

$$P_{11} \setminus_H = \tau.l_1.0 + \tau.l_2.0$$

is not bisimilar to $l_1.0$. This process has been argued as insecure because an l_2 event allows the low-level user to deduce the absence of an h event and a high-level user who is given the opportunity to act before the τ events can guarantee only an l_1 is available.

(End of Example)

Example 5.2.8 Let $L = \{l_0, l_1\}$, $H = \{h_0, h_1\}$ and consider the processes P_4 and P_5 from Example 5.1.3 and Example 5.1.4 respectively. $\mathsf{SLNI}_L(P_4)$

is not satisfied because neither of the states after the H labelled transitions are bisimilar to $\mu X.(l_0 \to X \sqcap l_1 \to X)$. We have argued for this process being insecure.

$\mathsf{SLNI}_L(P_5)$ is also not satisfied for the same reason that $\mathsf{SLNI}_L(P_4)$ fails. The condition is being over cautious here since we would like to argue for this process being secure. The two branches after the h_0 event are considered separately and both differ from $P_5 \setminus_H$.

(End of Example)

Strong local non-interference might be considered too strict since it fails to take into account the nondeterminism that arises from identically labelled $t \in H$ transitions which lead from the same state. When the high-level user communicates such an event there is no choice made (by the high-level user) over which transition takes place. If we are prepared to accept the internal choice as taking place once the high-level user has communicated the event, so that the decision to communicate a t cannot be based on the outcome of the nondeterminism, we need to weaken the condition.

There are two ways we can consider adapting strong local non-interference for identically labelled $t \in H$ transitions: firstly we could apply an operator to the process which rewrites any multiple H labelled transitions before applying SLNI; alternatively we can add a new condition that takes such multiple transitions into account. Both of these approaches say that we can replace any 'during' nondeterministic choice for the high-level user with an 'after' nondeterministic choice (see page 87). It seems reasonable to do this replacement, even though $h \to A \,\Box\, h \to B$ and $h \to (A \sqcap B)$ are not bisimilar, because $(h \to A \,\Box\, h \to B)/\langle h\rangle$ is bisimilar to $(h \to (A \sqcap B))/\langle h\rangle$.

The rewriting operator is applied over the transition system. For every multiple $h \in H$ labelling from the same state σ, a new state s^h_σ is created; a new h labelled transition goes from σ to s^h_σ and the old h labelled transitions from σ are altered to originate from s^h_σ and be labelled by a τ.[5]

[5]The s^h_σ are distinct from each other by construction. Without loss of generality we will assume they are also distinct from the $\sigma \in S$ since they are arbitrary names. Similarly for the t^h_σ.

Definition 5.2.9 Let $\langle S_1, T_1, \alpha_1, \beta_1, \lambda_1 \rangle$ be the LTS for a process P. Let

$$\begin{aligned} R = \{(s,h) \mid s \in S_1 \wedge (\exists\, t_1, t_2 \in T_1 \quad \bullet \quad & t1 \neq t2 \wedge h \in H \wedge \\ & \alpha_1(t_1) = \alpha_1(t_2) = s \wedge \\ & \lambda_1(t_1) = \lambda_1(t_2) = h)\} \end{aligned}$$

The LTS of the rewritten system, $\mathcal{R}_H(P)$, is $\langle S_2, T_2, \alpha_2, \beta_2, \lambda_2 \rangle$ where

- $S_2 = S_1 \cup \{s_a^b \mid (a,b) \in R\}$
- $T_2 = T_1 \cup \{t_a^b \mid (a,b) \in R\}$
- $\forall\, t \in T_1 \bullet$ if $(\alpha_1(t), \lambda_1(t)) \in R$ then $\alpha_2(t) = s_{\alpha_1(t)}^{\lambda_1(t)}$ else $\alpha_2(t) = \alpha_1(t)$
- $\forall\, t \in T_1 \bullet \beta_2(t) = \beta_1(t)$
- $\forall\, t \in T_1 \bullet$ if $(\alpha_1(t), \lambda_1(t)) \in R$ then $\lambda_2(t) = \tau$ else $\lambda_2(t) = \lambda_1(t)$
- $\forall\, t_a^b \in \{t_a^b \mid (a,b) \in R\} \bullet \alpha_2(t_a^b) = a,\ \beta_2(t_a^b) = s_a^b,\ \lambda_2(t_a^b) = b$

The alternate approach is to take the multiple H labelled transitions directly into account. This condition is called *local non-interference*[6]. The word 'strong' is dropped from this condition since the equivalence is not required on either side of every H labelled transition, rather a grouping of identical $t \in H$ labelled transitions from a single state is permitted.

Definition 5.2.10 Let P be a process whose alphabet is partitioned by H and L. P is said to satisfy *local non-interference*, written $\mathsf{LNI}_L(P)$, if $\sigma \xrightarrow{h} \sigma_1, \ldots, \sigma \xrightarrow{h} \sigma_n$, where $h \in H$, $\sigma, \sigma_1, \ldots, \sigma_n \in \text{states}(P)$, $n > 0$ and the states σ_i are a complete enumeration of the h-reachable states, implies $\sigma \setminus_H \approx_B \sqcap\, \sigma_i \setminus_H$.

These two approaches for dealing with multiple H labelled transitions can be shown to be equivalent.

[6]When both the new conditions, i.e. "strong local non-interference" and "local non-interference", are being referred to at the same time, the generic "local non-interference" will be used.

Theorem 5.2.11 $\mathsf{LNI}_L(P) \Leftrightarrow \mathsf{SLNI}_L(\mathcal{R}_H(P))$.

Proof This follows directly since, (i) for all states in P there is a corresponding state in $\mathcal{R}_H(P)$ which, by the construction in Definition 5.2.9, is identical when the H labelled transitions are restricted and (ii) the new states in $\mathcal{R}_H(P)$ (when restricted) are bisimilar to the $\sqcap\, \sigma_i \backslash_H$ in Definition 5.2.10. $\square$

The processes that did not satisfy strong local non-interference because of its treatment of identical H labelled transitions are now satisfied by the new condition.

Example 5.2.12 Let $L = \{l_0, l_1\}$, $H = \{h_0\}$ and consider the process P_5 from Example 5.1.4. Applying the transformation from Definition 5.2.9 gives the process

$$\begin{aligned} \mathcal{R}_H(P_5) = & \quad \tau.l_0.\mathcal{R}_H(P_5) + \tau.l_1.\mathcal{R}_H(P_5) \\ & + \; h_0.(\tau.l_0.\mathcal{R}_H(P_5) + \tau.l_1.\mathcal{R}_H(P_5)) \end{aligned}$$

$\mathcal{R}_H(P_5)$ satisfies SLNI since the state before and after the h_0 event is the same. Alternatively, $\mathsf{LNI}_L(P_5)$ is satisfied because $P_5 \backslash_H$ is equivalent to the nondeterministic choice of the restricted states after the two possible h_0 labelled transitions. $\mathsf{SLNI}_L(P_5)$ does not hold, as shown in Example 5.2.8, since it does not take into account the nondeterministic choice that is made as to which h_0 event occurs from the initial state; it was for precisely this reason that LNI was introduced.

(End of Example)

As with strong local non-interference, the concept of local non-interference can be extended to a family of security conditions using other equivalences. In particular it can be extended to failures/divergences.

Definition 5.2.13 Let P be a process whose alphabet is partitioned by H and L. P is said to satisfy *FD local non-interference*, written $\mathsf{LNI}_L^{FD}(P)$, if $\sigma \xrightarrow{h} \sigma_1, \ldots, \sigma \xrightarrow{h} \sigma_n$, where $h \in H$, $\sigma, \sigma_1, \ldots, \sigma_n \in \text{states}(P)$, $n > 0$ and the states σ_i are a complete enumeration of the h-reachable states, implies $\sigma \backslash_H =_{\text{FD}} \sqcap\, \sigma_i \backslash_H$.

The local non-interference conditions do not make any distinction between input and output events. If the distinction between the two is actually between delayable and signal events, we can adapt the same approach as the mixed abstraction on page 40. Rather than consider $\mathsf{SLNI}_L(P)$ or $\mathsf{LNI}_L(P)$ we consider $\mathsf{SLNI}_L(P \setminus S)$ or $\mathsf{LNI}_L(P \setminus S)$, where the events in S are the high-level signal events. As shown later, in Theorem 6.2.6, this cannot introduce any insecurities into a secure system; a high-level user may always accept delayable events. Any systems that become secure do so because the conditions no longer distinguish between the signal events being accepted or refused.

5.3 Basic Properties

There are a number of basic properties which the local non-interference conditions satisfy. These properties are concerned with the behaviour of the conditions on processes with restricted alphabets and the states of secure processes. Operators that affect existing states or combine process are considered in Chapter 6.

Any process that does nothing will be secure, since there is no way a user can either influence the system or receive information from it. We would expect all security conditions to be satisfied by such a system. [This is not necessarily true of equivalent systems since we may wish to differentiate between livelock and deadlock.]

Theorem 5.3.1 $\mathsf{SLNI}_L(\mathit{Stop})$ and $\mathsf{LNI}_L(\mathit{Stop})$ hold; similarly for SLNI^{FD} and LNI^{FD}.

Proof There are no H labelled transitions, and so the theorem follows directly from the definitions of the local non-interference conditions. $\square$

If a system does not permit any H labelled actions there is no way for the high-level user to influence the system. There is no information for the low-level user to be able to observe or infer and so the system will be secure.

Theorem 5.3.2 If $\Sigma P \cap H = \emptyset$ then $\mathsf{SLNI}_L(P)$ and $\mathsf{LNI}_L(P)$ hold; similarly for SLNI^{FD} and LNI^{FD}.

Proof There are no H labelled transitions, and so the theorem follows directly from the definitions of the local non-interference conditions. $\square$

An implication of this theorem is that for such systems there is no reason, from an interference standpoint, to modify what the low-level user can communicate; in particular it does not matter how nondeterministic the system appears to the low-level user.

Example 5.3.3 Let $L = \{head, tail\}$. The process

$$COIN = head \rightarrow Stop \sqcap tail \rightarrow Stop$$

satisfies the local non-interference conditions.

(End of Example)

Similarly we would expect a system that cannot communicate with the low-level user to be secure. Whatever actions the high-level user communicates, and however these are interpreted by the system, the low-level user is unable to observe the system.

Theorem 5.3.4 If $\Sigma P \cap L = \emptyset$ then $\mathsf{SLNI}_L(P)$ and $\mathsf{LNI}_L(P)$ hold.

Proof Let σ be a state in P. There are no L labelled transitions so $\sigma \setminus_H$ is weakly-bisimilar to *Stop*. Since any state that can be reached from σ by a $t \in H$ labelled transition is also weakly-bisimilar to *Stop* the conditions hold. $\square$

This theorem relies on the fact that weak-bisimulation does not have a catastrophic view of divergence. If we are using a version of the local non-interference conditions with an equivalence, such as failures-divergences, that does not treat divergence in the same way, the additional precondition of non-divergence should be added.

Theorem 5.3.5 If $\Sigma P \cap L = \emptyset$ and P is non-divergent, then $\mathsf{SLNI}_L^{FD}(P)$ and $\mathsf{LNI}_L^{FD}(P)$ hold.

One of the motivating properties for the new conditions was that a secure process should be unable to get into a state where it behaves as an insecure process. This cannot occur with the local non-interference conditions since the equivalences are take over all the relevant states of the process.

Theorem 5.3.6 If $P' \in \mathit{states}\,[\![P]\!]$ then

(i) $\mathsf{SLNI}_L(P) \Rightarrow \mathsf{SLNI}_L(P')$.
(ii) $\mathsf{LNI}_L(P) \Rightarrow \mathsf{LNI}_L(P')$.

Proof (for case i, strong local non-interference)

Suppose, for a contradiction, that $\mathsf{SLNI}_L(P')$ does not hold. Hence, by Definition 5.2.1, there exists $\sigma, \sigma' \in \mathit{states}\,[\![P']\!]$ and $h \in H$ such that $\sigma \xrightarrow{h} \sigma'$ but $\sigma \setminus_H \not\approx_B \sigma' \setminus H$. Now $P' \in \mathit{states}\,[\![P]\!]$ implies $\mathit{states}\,[\![P']\!] \subseteq \mathit{states}\,[\![P]\!]$, hence $\sigma, \sigma' \in \mathit{states}\,[\![P']\!]$. Therefore $\mathsf{SLNI}_L(P)$ does not hold and we have a contradiction.

An analogous proof holds for case ii (local non-interference). $\square$

Example 5.3.7 Let P be a process and construct, as in Theorem 4.3.11, a process $P' = (\bigsqcap S) \sqcap P$ where

$$S = \{(P \underset{H}{\parallel} Q) \setminus H \mid \Sigma Q \subseteq H\}$$

This process will be insecure whenever P is insecure and will behave in such a manner whenever the high-level user is able to communicate. $\mathsf{SLNI}_L(P')$ will be satisfied if and only if $\mathsf{SLNI}_L(P)$ is satisfied; similarly for LNI. Failure to satisfy either condition implies there exists a state in process P that fails the relevant requirement. In process P' we can still reach this state and so, by Theorem 5.3.6, the same failure will occur. Conversely, since the only states with H labelled transitions in P' are also in P it follows that a secure P implies a secure P'.

(End of Example)

The preceding theorem can be used to prove results about the application of guarded alternative operators to processes. The simplest operator is the prefixing of a single initial choice. We would expect that a single L labelled (or internal) event would make no difference to a process' security, since the high-level user cannot influence it.

Proposition 5.3.8 Let $a \in L \cup \{\tau\}$. $\mathsf{SLNI}_L(a \to P)$ holds if and only if $\mathsf{SLNI}_L(P)$ holds. Similarly for LNI.

Proof (for strong local non-interference)

($\Rightarrow$) Since $P \in \mathit{states}\,[\![a \to P]\!]$, by Theorem 5.3.6, $\mathsf{SLNI}_L(a \to P) \Rightarrow \mathsf{SLNI}_L(P)$.

($\Leftarrow$) $\mathsf{SLNI}_L(P) \Rightarrow \mathsf{SLNI}_L(a \to P)$ by Definition 5.2.1, since $a \notin H$.

An analogous proof holds for LNI. $\square$

Prefixing an H labelled event, since it can introduce a delay, requires the remaining system to appear deadlocked to the low-level user.

Proposition 5.3.9 Let $a \in H$. $\mathsf{SLNI}_L(a \to P)$ holds if and only if $\Sigma P \cap L = \emptyset$.

Proof

($\Leftarrow$) $\Sigma P \cap L = \emptyset$ implies, since $a \in H$, $\Sigma(a \to P) \cap L = \emptyset$. Therefore, by Theorem 5.3.4, $\mathsf{SLNI}_L(a \to P)$ holds.

($\Rightarrow$) Let $\mathsf{SLNI}_L(a \to P)$ hold. We need to show that $\Sigma P \cap L = \emptyset$. Suppose, for a contradiction that this is false. Therefore there exists $l \in L$, $h_i \in H$ and $P_i \in \mathit{states}\,[\![P]\!]$ (letting $P \equiv P_0$) so that

$$P_0 \overset{h_1}{\Longrightarrow} P_1 \ldots \overset{h_n}{\Longrightarrow} P_n \overset{l}{\Longrightarrow} P'$$

where $n \geqslant 1$ and $\forall\, i \in \{0, \ldots, n-1\} \bullet P_i \setminus_H \approx_B \mathit{Stop}$. I.e. P_n is the first state reached on the path which has an L labelled transition. Now $P_{n-1} \setminus_H \not\approx_B P_n \setminus_H$ so, by Theorem 5.3.6, $\mathsf{SLNI}_L(a \to P)$ does not hold and we have a contradiction. Hence $\Sigma P \cap L = \emptyset$ as required.

An analogous proof holds for LNI. $\square$

If we take the nondeterministic choice of two secure processes the resulting process will also be secure. The low-level user may be able to deduce which process has been selected, but this is not a flow of information since it is not due to the high-level user.

Theorem 5.3.10 $\mathsf{SLNI}_L(P)$ and $\mathsf{SLNI}_L(Q)$ hold if and only if $\mathsf{SLNI}_L(P \sqcap Q)$ holds. Similarly for LNI.

Proof

($\Leftarrow$) Suppose $\mathsf{SLNI}_L(P \sqcap Q)$ holds. P and Q are states of $P \sqcap Q$ hence, by Theorem 5.3.6, $\mathsf{SLNI}_L(P)$ and $\mathsf{SLNI}_L(Q)$ hold.

($\Rightarrow$) Suppose $\mathsf{SLNI}_L(P)$ and $\mathsf{SLNI}_L(Q)$ hold. We can create the state transition system for $P \sqcap Q$ by taking the systems for P and Q, renaming (if necessary) states to avoid conflict, and adding a new initial state that has a τ labelled transition to each of the two original initial states. The transitions between states in P and Q remain unchanged. The only new state is the initial state, and since this only has τ labelled transitions it satisfies SLNI. Hence $\mathsf{SLNI}_L(P \sqcap Q)$ holds as required. $\square$

5.4 Condition Relationships

In this section various relationships between the local non-interference conditions are considered; between the strong and weaker conditions and between those that use different equivalences. We will show that for deterministic processes all the conditions are equivalent. This result is then extended in Section 6.1 by showing they are equivalent, for deterministic processes, to the lazy security condition.

The strong conditions will imply the weaker ones. If a state has multiple $t \in H$ labelled transitions and a strong condition holds, all the resulting (restricted) states must be equivalent. The outcome of the nondeterministic choice is therefore arbitrary. The reverse implication is clearly false for general processes, as shown in Example 5.2.8.

Theorem 5.4.1 Let P be a process whose alphabet is partitioned by H and L. $\mathsf{SLNI}_L(P) \Rightarrow \mathsf{LNI}_L(P)$ and similarly $\mathsf{SLNI}_L^{FD}(P) \Rightarrow \mathsf{LNI}_L^{FD}(P)$.

Proof Let $\sigma \xrightarrow{t} \sigma_1, \ldots, \sigma \xrightarrow{t} \sigma_n$ where $t \in H$, $\sigma \in \mathit{states}[\![P]\!]$ and the σ_i are a complete enumeration of the t-reachable states from σ. Now $\forall i \bullet \sigma \setminus_H \approx_B \sigma_i \setminus_H$ since $\mathsf{SLNI}_L(P)$ holds. Therefore, since $(X \approx_B Y) \Rightarrow (X \sqcap Y \approx_B Y)$ and $\approx_B$ is transitive, $\sigma \setminus_H \approx_B \bigsqcap \sigma_i \setminus_H$ as required.

An analogous argument holds for the failures/divergences conditions. □

The next relationship theorem compares the weak bisimulation versions of the conditions to the failures/divergences ones. The result is a consequence of the relationship between the equivalences.

Theorem 5.4.2 For non-divergent processes $P \approx_{\mathrm{B}} Q$ implies $P =_{\mathrm{FD}} Q$.

Proof This holds for all non-divergent processes[7]. If we assume for a contradiction (without loss of generality) that $\exists (s, X) \in \mathit{failures}[\![P]\!]$ and $(s, X) \notin \mathit{failures}[\![Q]\!]$ we can show, by using an inductive argument on the paths from these states to P and Q, that $P \not\approx_{\mathrm{B}} Q$. □

Theorem 5.4.3 Let P be a non-divergent process whose alphabet is partitioned by H and L. $\mathsf{SLNI}_L(P) \Rightarrow \mathsf{SLNI}_L^{FD}(P)$ and similarly $\mathsf{LNI}_L(P) \Rightarrow \mathsf{LNI}_L^{FD}(P)$.

Proof The follows directly from Theorem 5.4.2 and Definitions 5.2.1, 5.2.4, 5.2.10 and 5.2.13. □

If P is a non-divergent, low-deterministic process all of the conditions collapse into one. We will first demonstrate that the reverse of Theorem 5.4.1 holds for the failures/divergences conditions over such processes. The reverse of Theorem 5.4.3 will then be proved to give their equivalence to the weak bisimulation conditions.

It is necessary to prove a few elementary results about deterministic processes.

[7] Trivially the processes $P = a.P$ and $Q = a.Q + \tau.Q$ are weakly bisimilar but not failures-divergences equivalent.

Lemma 5.4.4 Let $\sigma \setminus_H =_{\mathrm{FD}} \sqcap \sigma_i \setminus_H$, and let $\sigma \setminus_H$ be deterministic.

(i) $\forall\, i \bullet \sigma_i \setminus_H$ is deterministic.
(ii) $\forall\, i, j \bullet \sigma_i \setminus_H =_{\mathrm{FD}} \sigma_j \setminus_H$.

Proof

(i) *failures* $[\![\sigma_i \setminus_H]\!] \subseteq$ *failures* $[\![\sigma \setminus_H]\!]$, so any nondeterminism in $\sigma_i \setminus_H$ will still exist in $\sigma \setminus_H$.

(ii) By (i) both $\sigma_i \setminus_H$ and $\sigma_j \setminus_H$ are deterministic. Since the non-deterministic choice of two processes can only be deterministic if both processes are deterministic and equivalent, and $\sigma \setminus_H$ is deterministic, $\sigma_i \setminus_H =_{\mathrm{FD}} \sigma_j \setminus_H$. $\square$

Lemma 5.4.5 Let P be a non-divergent, deterministic process. If $P \stackrel{\delta}{\Longrightarrow} P'$ then P' is deterministic.

Proof Suppose, for a contradiction, $\exists\, \gamma, a$ such that $(\gamma, \{a\}) \in$ *failures* $[\![P']\!]$ and $\gamma \frown \langle a \rangle \in$ *traces* $[\![P']\!]$. Since $P \stackrel{\delta}{\Longrightarrow} P'$, $(\delta \frown \gamma, \{a\}) \in$ *failures* $[\![P]\!]$ and $\delta \frown \gamma \frown \langle a \rangle \in$ *traces* $[\![P]\!]$, but this contradicts our hypothesis that P is deterministic. Similarly if P' could diverge so could P. Hence P' is deterministic. $\square$

The previous result shows that every state in a deterministic process is also deterministic. If P is low-deterministic the only restriction is that it behaves deterministically until an H labelled event occurs. If in addition to $LowDet_L(P)$ we have $\mathsf{SLNI}_L^{FD}(P)$ or $\mathsf{LNI}_L^{FD}(P)$ holding, this can be extended to show that every restricted state in P behaves deterministically.

Lemma 5.4.6 Let P be a non-divergent, low-deterministic process that satisfies $\mathsf{SLNI}_L^{FD}(P)$ or $\mathsf{LNI}_L^{FD}(P)$. If $P \stackrel{\delta}{\Longrightarrow} P'$ then $P' \setminus_H$ is determinisitic.

Proof Let $\delta = \lambda_1 \frown \langle h_1 \rangle \frown \ldots \frown \langle h_n \rangle \frown \lambda_{n+1}$ where $\forall\, i \bullet \lambda_i \in L^*$, and hence may be empty. Since δ is a trace of P we can choose P_i's and P_i''s so that, letting $P_0' \hat{=} P$ and $P_{n+1} \hat{=} P'$

$$P_0' \stackrel{\lambda_1}{\Longrightarrow} P_1 \stackrel{h_1}{\longrightarrow} P_1' \stackrel{\lambda_2}{\Longrightarrow} P_2 \ldots P_n \stackrel{h_n}{\longrightarrow} P_n' \stackrel{\lambda_{n+1}}{\Longrightarrow} P_{n+1}$$

If $P'_i \setminus_H$ is deterministic

$$
\begin{array}{ll}
 & P'_i \stackrel{\lambda_{i+1}}{\Longrightarrow} P_{i+1} \\
\Rightarrow & \hfill [\ \lambda_{i+1} \in L^*\ ] \\
 & P'_i \setminus_H \stackrel{\lambda_{i+1}}{\Longrightarrow} P_{i+1} \setminus_H \\
\Rightarrow & \hfill [\ P'_i \setminus_H \text{ is deterministic, Lemma 5.4.5 }] \\
 & P_{i+1} \setminus_H \ \text{ is deterministic}
\end{array}
$$

If $\mathsf{SLNI}_L^{FD}(P)$ holds, $P_i \setminus_H$ is deterministic implies $P'_i \setminus_H$ is deterministic by Definition 5.2.4. If $\mathsf{LNI}_L^{FD}(P)$ holds the implication holds by Lemma 5.4.4(i). By induction we can therefore show $P' \setminus_H$ is deterministic as required. □

The equivalence of the failures/divergences local non-interference conditions can now be shown for low-deterministic processes.

Theorem 5.4.7 If P is a non-divergent, low-deterministic process then $\mathsf{LNI}_L^{FD}(P) \Rightarrow \mathsf{SLNI}_L^{FD}(P)$.

Proof Let σ be a state in P where $\sigma \xrightarrow{t} \sigma_1, \ldots, \sigma \xrightarrow{t} \sigma_n$, $t \in H$, $P \stackrel{\delta}{\Longrightarrow} \sigma$ and the σ_i are a complete enumeration of the t-reachable states from σ. By Lemma 5.4.6 we have $\sigma \setminus_H$ is deterministic.

Let $i \in \{1 \ldots n\}$

$$
\begin{array}{ll}
 & \sigma \setminus_H \\
=_{\mathrm{FD}} & \hfill [\ \mathsf{LNI}_L^{FD}(P) \text{ holds }] \\
 & \sigma_1 \setminus_H \sqcap \ldots \sqcap \sigma_n \setminus_H \\
=_{\mathrm{FD}} & \hfill [\ \sigma \setminus_H \text{ deterministic, Lemma 5.4.4(ii) }] \\
 & \sigma_i \setminus_H \sqcap \ldots \sqcap \sigma_i \setminus_H \\
=_{\mathrm{FD}} & \hfill [\ \sqcap \text{ idempotence }] \\
 & \sigma_i \setminus_H
\end{array}
$$

Since σ and σ_i can be any states linked by an H labelled transition, $\mathsf{SLNI}_L^{FD}(P)$ holds by Definition 5.2.4. □

Theorem 5.4.8 If P is a non-divergent, low-deterministic process then $\mathsf{LNI}_L^{FD}(P)$ holds if and only if $\mathsf{SLNI}_L^{FD}(P)$ holds.

Proof This follows directly from Theorem 5.4.1 ($\Rightarrow$) and Theorem 5.4.7 ($\Leftarrow$). $\square$

For non-divergent, deterministic processes we can show that weak bisimulation and failures/divergences are equivalent. This result does not hold for processes that are merely low-deterministic. Nevertheless, a low-deterministic process that satisfies a local non-interference condition exhibits sufficient determinism that the equivalence can be used to show that the conditions collapse.

Theorem 5.4.9 For non-divergent, deterministic processes, weak bisimulation ($\approx_B$) and failures-divergences equivalence ($=_{\mathrm{FD}}$) are interchangeable.

Proof

(i) $P \approx_{\mathrm{B}} Q$ implies $P =_{\mathrm{FD}} Q$. This is proved in Theorem 5.4.2 for any non-divergent process.

(ii) $P =_{\mathrm{FD}} Q$ implies $P \approx_{\mathrm{B}} Q$.

Define a relation R where $(P', Q') \in R$ if and only if $\exists\, \gamma$ where $P \overset{\gamma}{\Longrightarrow} P'$ and $Q \overset{\gamma}{\Longrightarrow} Q'$. We need to show that this is a weak bisimulation equivalence. $\forall (A, B) \in R$ suppose (without loss of generality) A can accept k but B cannot; since $P =_{\mathrm{FD}} Q$ this implies there is a state $B' \in Q$ which can accept k, where $(A, B') \in R$, but this contradicts the hypothesis that the processes are deterministic. Similarly suppose $\exists\, k, A', B'$ such that $A \xrightarrow{k} A', B \overset{\hat{k}}{\Longrightarrow} B'$ but $(A', B') \notin R$; this contradicts the definition of R. Hence R is a weak bisimulation. $\square$

Theorem 5.4.10 Let P be a non-divergent, low-deterministic process. $\mathsf{SLNI}_L(P)$ holds if and only if $\mathsf{SLNI}_L^{FD}(P)$ holds.

Proof Now $\mathsf{SLNI}_L(P) \Rightarrow \mathsf{SLNI}_L^{FD}(P)$ from Theorem 5.4.3, and

$$
\begin{array}{ll}
 & \mathsf{SLNI}_L^{FD}(P) \\
= & \hfill [\text{ Definition 5.2.4 }] \\
 & \forall\, \sigma, h \bullet \sigma \xrightarrow{h} \sigma' \text{ implies } \sigma \setminus_H \;=_{\mathrm{FD}}\; \sigma' \setminus_H \\
\Rightarrow & \hfill [\text{ Lemma 5.4.6(i), Theorem 5.4.9 }] \\
 & \forall\, \sigma, h \bullet \sigma \xrightarrow{h} \sigma' \text{ implies } \sigma \setminus_H \;\approx_B\; \sigma' \setminus_H \\
= & \hfill [\text{ Definition 5.2.1 }] \\
 & \mathsf{SLNI}_L(P)
\end{array}
$$

□

Theorem 5.4.11 Let P be a non-divergent, low-deterministic process. $\mathsf{LNI}_L(P)$ holds if and only if $\mathsf{LNI}_L^{FD}(P)$ holds.

Proof We have $\mathsf{LNI}_L(P) \Rightarrow \mathsf{LNI}_L^{FD}(P)$ from Theorem 5.4.3.

$$
\begin{array}{ll}
 & \mathsf{LNI}_L^{FD}(P) \\
\equiv & \hfill [\; P \text{ non-divergent, low-deterministic, Theorem 5.4.8 }] \\
 & \mathsf{SLNI}_L^{FD}(P) \\
\Rightarrow & \hfill [\text{ Theorem 5.4.10 }] \\
 & \mathsf{SLNI}_L(P) \\
\Rightarrow & \hfill [\text{ Theorem 5.4.1 }] \\
 & \mathsf{LNI}_L(P)
\end{array}
$$

□

Corollary 5.4.12 Let P be a non-divergent, low-deterministic process. All the local non-interference conditions (SLNI, SLNI^{FD}, LNI and LNI^{FD}) are equivalent.

Proof This follows directly from Theorems 5.4.8, 5.4.10 and 5.4.11. □

Chapter 6

Development & Comparisons

In the previous chapter a new family of conditions was introduced. These conditions require any H labelled transitions in a process P to be between states that are indistinguishable to the low-level user when further H labelled events are prevented. By preventing the high-level user from resolving any nondeterminisms in the process the flow of information is prevented. Some elementary properties for these conditions have been shown as well as details on their inter-relationships.

In this chapter more complicated properties are shown. In Section 6.1 the conditions are compared to two existing security conditions: lazy security and SBSNNI. It is shown, for low-deterministic processes, that all these conditions are in agreement.

A number of laws applicable to constructing secure processes from existing components are then considered. Section 6.2 considers the effects of applying the hiding operator to a secure process. In Section 6.3 the parallel composition operator is considered; these compositional laws are extended in Section 6.4 where the specific case of interleaving (composition over a null alphabet) is considered. Finally, Section 6.5 discusses how to implement the automatic verification of the conditions.

6.1 Comparisons

One of the desirable properties highlighted in Section 5.1 was for any new security condition to be in agreement with existing conditions for deterministic processes. This in itself is meaningful because there is a general agreement amongst the more recent conditions. For example, conditions of Allen [All91], Graham-Cumming [Gra92] and Ryan [Rya91] are shown as equivalent to a traces based condition of Roscoe's [Ros95a] which, together with determinism, implies the lazy security condition. This in turn has been shown to agree with Focardi and Gorierri's condition [Foc96] for deterministic processes and has itself been expressed in several forms which have been shown to be equivalent (see Chapter 3).

In Section 5.4 the local non-interference conditions were shown to be equivalent for low-deterministic processes. In this section we will show that, for low-deterministic processes, these conditions are equivalent to the lazy security condition. Since low-determinism is implied by determinism this will show the required condition. Furthermore, since it is implied by independence and that is equivalent to lazy security, we will show that there are no lazily secure process that the local non-interference conditions classify as insecure. Two other relationships will also be shown; how the different low determinism conditions collapse when a local non-interference condition holds, and how the new conditions are related to Focardi and Gorierri's SBSNNI.

Equivalence for Deterministic Processes

We will show that the interleaving abstraction corresponds to strong local non-interference for low-deterministic processes. The interleaving abstraction has been chosen because of its effect on the LTS of a process. The lazy abstraction for $\mathcal{N}$ is based on denotational rather than operational semantics which makes direct comparison with an LTS difficult. In contrast the interleaving abstraction is based on standard CSP operators. It also leads to an LTS which has the same states as the original process (since Run_H is a one state process) although there are additional transitions.

First, for a non-divergent, low-deterministic process P, strong local non-interference will be shown to imply the determinism of the interleaving abstraction. Two lemmas are needed for this, to show that any failure in P is also exhibited in $P \setminus_H$ when the H labelled events are removed from the trace. This is shown inductively by moving backwards along the trace and pulling the refusals through the H labelled transitions.

Lemma 6.1.1 If $(\langle\rangle, K) \in \mathit{failures}\,[\![P]\!]$ then $(\langle\rangle, K) \in \mathit{failures}\,[\![P \setminus_H\,]\!]$.

Proof

$$
\begin{array}{ll}
 & (\langle\rangle, K) \in \mathit{failures}\,[\![P]\!] \\
\Rightarrow & \hfill [\ P \setminus_H \text{ refuses all } H \text{ events }] \\
 & (\langle\rangle, K \cup H) \in \mathit{failures}\,[\![P \setminus_H\,]\!] \\
\Rightarrow & \hfill [\ \text{Axiom F2}\ ] \\
 & (\langle\rangle, K) \in \mathit{failures}\,[\![P \setminus_H\,]\!]
\end{array}
$$

$\square$

Lemma 6.1.2 Let P be a non-divergent process and let $\mathsf{SLNI}_L^{FD}(P)$ hold. If $(\delta, K) \in \mathit{failures}\,[\![P]\!]$ then $(\delta \upharpoonright L, K) \in \mathit{failures}\,[\![P \setminus_H\,]\!]$.

Proof Let $\delta = \lambda_1 {}^\frown \langle h_1 \rangle {}^\frown \ldots {}^\frown \langle h_n \rangle {}^\frown \lambda_{n+1}$ where $\forall\, i \bullet \lambda_i \in L^*$, and hence may be empty. Since δ is a trace of P we can choose P_i's and P_i''s so that

$$P \stackrel{\lambda_1}{\Longrightarrow} P_1 \stackrel{h_1}{\longrightarrow} P_1' \stackrel{\lambda_2}{\Longrightarrow} P_2 \ldots P_n \stackrel{h_n}{\longrightarrow} P_n' \stackrel{\lambda_{n+1}}{\Longrightarrow} P_{n+1}$$

and $(\langle\rangle, K) \in \mathit{failures}\,[\![P_{n+1}]\!]$. Hence we have $(\langle\rangle, K) \in \mathit{failures}\,[\![P_{n+1} \setminus_H\,]\!]$ by Lemma 6.1.1. If $n = 0$, since δ $(= \lambda_1)$ contains no H labelled actions and $P \stackrel{\lambda_1}{\Longrightarrow} P_1$, we get $(\delta \upharpoonright L, K) \in \mathit{failures}\,[\![P \setminus_H\,]\!]$ as required.

Let $n > 0, i \geqslant 1$

$$
\begin{array}{ll}
 & (\lambda_{i+1} {}^\frown \ldots {}^\frown \lambda_{n+1}, K) \in \mathit{failures}\,[\![P_i \setminus_H\,]\!] \\
\Rightarrow & \hfill [\ P_{i-1}' \stackrel{\lambda_i}{\Longrightarrow} P_i\ ] \\
 & (\lambda_i {}^\frown \ldots {}^\frown \lambda_{n+1}, K) \in \mathit{failures}\,[\![P_{i-1}' \setminus_H\,]\!] \\
\Rightarrow & \hfill [\ \mathsf{SLNI}_L^{FD}(P) \text{ holds }] \\
 & (\lambda_i {}^\frown \ldots {}^\frown \lambda_{n+1}, K) \in \mathit{failures}\,[\![P_{i-1} \setminus_H\,]\!]
\end{array}
$$

By an inductive argument $(\lambda_2 \frown \ldots \frown \lambda_{n+1}, K) \in \mathit{failures}\,[\![P_1 \setminus_H\ ]\!]$. Therefore, since $P \overset{\lambda_1}{\Longrightarrow} P_1$ and $\delta \restriction L = \lambda_1 \frown \ldots \frown \lambda_{n+1}$ we have, as required, $(\delta \restriction L, K) \in \mathit{failures}\,[\![P \setminus_H\ ]\!]$. □

By building a transition system for $P \mid\mid\mid \mathit{Run}_H$ which has the same states as P, the previous lemma can be used to show that strong local non-interference implies the lazy security condition for low-deterministic processes. The proof is by contradiction; we show that any nondeterminism in $P \mid\mid\mid \mathit{Run}_H$ will also be exhibited in $P \setminus_H$.

Theorem 6.1.3 Let P be a non-divergent, low-deterministic process. If $\mathsf{SLNI}_L^{FD}(P)$ holds then $P \mid\mid\mid \mathit{Run}_H$ is deterministic.

Proof We want to show that $P \mid\mid\mid \mathit{Run}_H$ is deterministic. Suppose, for a contradiction, $\exists\,\delta, a$ such that $(\delta, \{a\}) \in \mathit{failures}\,[\![P \mid\mid\mid \mathit{Run}_H]\!]$ and $\delta \frown \langle a \rangle \in \mathit{traces}\,[\![P \mid\mid\mid \mathit{Run}_H]\!]$. Trivially a cannot be in H since $P \mid\mid\mid \mathit{Run}_H$ can always communicate all the events in H.

Consider a transition system for $P \mid\mid\mid \mathit{Run}_H$ generated as follows: take a transition system for P, for each state $s \in P$ add a transition for each element $h \in H$ returning to s.

(i) Since $(\delta, \{a\}) \in \mathit{failures}\,[\![P \mid\mid\mid \mathit{Run}_H]\!]$ we have $\exists\,P'$ such that $P' \overset{\hat{a}}{\not\Longrightarrow}$ and $P \mid\mid\mid \mathit{Run}_H \overset{\delta}{\Longrightarrow} P' \mid\mid\mid \mathit{Run}_H$. Let γ be a path from P to P' along the same transitions as δ but ignoring the new transitions which have been added for the Run_H interleaving. We have $P \overset{\gamma}{\Longrightarrow} P'$ where $P' \overset{\hat{a}}{\not\Longrightarrow}$, and $\delta \restriction L = \gamma \restriction L$ (since only H labelled events can be removed).

$$
\begin{array}{lll}
 & (\langle\rangle, \{a\}) \in \mathit{failures}\,[\![P']\!] & \\
\Rightarrow & & [\ P \overset{\gamma}{\Longrightarrow} P'\ ] \\
 & (\gamma, \{a\}) \in \mathit{failures}\,[\![P]\!] & \\
\Rightarrow & & [\ \text{Lemma 6.1.2}\ ] \\
 & (\gamma \restriction L, \{a\}) \in \mathit{failures}\,[\![P \setminus_H\ ]\!] & \\
\Rightarrow & & [\ \delta \restriction L = \gamma \restriction L\ ] \\
 & (\delta \restriction L, \{a\}) \in \mathit{failures}\,[\![P \setminus_H\ ]\!] &
\end{array}
$$

(ii) Similarly, since $\delta \frown \langle a \rangle \in \mathit{traces}\,[\![P \mid\mid\mid \mathit{Run}_H]\!]$, we can find Q and Q' such that $P \stackrel{\gamma'}{\Longrightarrow} Q$ where $Q \stackrel{a}{\longrightarrow} Q'$ and $\delta \upharpoonright L = \gamma' \upharpoonright L$. Since $\exists\, K$ such that $(\gamma' \frown \langle a \rangle, K) \in \mathit{failures}\,[\![Q']\!]$ we now use a similar argument to the one above to give $((\gamma' \upharpoonright L) \frown \langle a \rangle, K) \in \mathit{failures}\,[\![P \setminus_H\,]\!]$.

We can therefore show $(\delta \upharpoonright L) \frown \langle a \rangle \in \mathit{traces}\,[\![P \setminus_H\,]\!]$. Now $(\delta \upharpoonright L, \{a\}) \in \mathit{failures}\,[\![P \setminus_H\,]\!]$ so we have a contradiction, since P is given as a low-deterministic process. Hence $P \mid\mid\mid \mathit{Run}_H$ is deterministic as required. □

Secondly, for a non-divergent, low-deterministic process P, the lazy security condition will be shown to imply strong local non-interference. We will first show that any failure in a state $\sigma' \setminus_H$ corresponds to a failure in both $\sigma' \mid\mid\mid \mathit{Run}_H$ and, if σ is a state that has an H labelled transition to σ', for $\sigma \mid\mid\mid \mathit{Run}_H$.

Lemma 6.1.4 If $(\delta, \{k\}) \in \mathit{failures}\,[\![\sigma' \setminus_H\,]\!]$ where $\delta \in L^*, k \in L$

(i) $(\delta, \{k\}) \in \mathit{failures}\,[\![\sigma' \mid\mid\mid \mathit{Run}_H]\!]$.

(ii) If $\sigma \stackrel{t}{\longrightarrow} \sigma', t \in H$ then $(\langle t \rangle \frown \delta, \{k\}) \in \mathit{failures}\,[\![\sigma \mid\mid\mid \mathit{Run}_H]\!]$.

Proof (of (ii). (i) is similar)

$$
\begin{array}{ll}
 & (\delta, \{k\}) \in \mathit{failures}\,[\![\sigma' \setminus_H\,]\!] \\
\Rightarrow & \hfill [\ \delta \in L^*, k \in L\ ] \\
 & (\delta, \{k\}) \in \mathit{failures}\,[\![\sigma']\!] \\
\Rightarrow & \hfill [\ \sigma \stackrel{t}{\longrightarrow} \sigma'\ ] \\
 & (\langle t \rangle \frown \delta, \{k\}) \in \mathit{failures}\,[\![\sigma]\!] \\
\Rightarrow & \hfill [\ k \in L\ ] \\
 & (\langle t \rangle \frown \delta, \{k\}) \in \mathit{failures}\,[\![\sigma \mid\mid\mid \mathit{Run}_H]\!]
\end{array}
$$

□

The proof of the implication is in several parts. We need to show that any two states σ and σ' that have an H labelled transition between them are equivalent. To do this we compare the failures of all such $\sigma \setminus_H$ and $\sigma' \setminus_H$. Firstly we show that $\mathit{failures}\,[\![\sigma \setminus_H\,]\!] \subseteq \mathit{failures}\,[\![\sigma' \setminus_H\,]\!]$ in two stages; a refusal K in $\sigma \setminus_H$ after a trace δ cannot occur in $\sigma' \setminus_H$ after a shorter trace $\delta' < \delta$, nor is there a longer trace $\delta \frown k$ $(k \in K)$ in $\sigma' \setminus_H$. The reverse inclusion is then proved using the earlier results.

Theorem 6.1.5 Let P be a non-divergent process. If $P \mid\mid\mid Run_H$ is deterministic then $\mathsf{SLNI}_L^{FD}(P)$ holds.

Proof For all states $\sigma \in P$ we need to show, if $\sigma \xrightarrow{t} \sigma', t \in H$ then $\sigma \setminus_H =_{\mathrm{FD}} \sigma' \setminus_H$.

(i) Let $(\delta, K) \in \mathit{failures}\,[\![\sigma \setminus_H]\!]$ where $\delta = \langle\delta_1\rangle \frown \ldots \frown \langle\delta_n\rangle$. Suppose, for a contradiction, $\exists\, m < n$ such that $(\langle\delta_1\rangle \frown \ldots \frown \langle\delta_m\rangle, \{\delta_{m+1}\}) \in \mathit{failures}\,[\![\sigma' \setminus_H]\!]$. Therefore $(\langle t\rangle \frown \langle\delta_1\rangle \frown \ldots \frown \langle\delta_m\rangle, \{\delta_{m+1}\}) \in \mathit{failures}\,[\![\sigma \mid\mid\mid Run_H]\!](\dagger)$ by Lemma 6.1.4(ii).

$$\begin{array}{ll} & \delta \in \mathit{traces}\,[\![\sigma]\!] \\ \Rightarrow & \hfill [\ t \in H\ ] \\ & \langle t\rangle \frown \delta \in \mathit{traces}\,[\![\sigma \mid\mid\mid Run_H]\!] \\ \Rightarrow & \hfill [\ (\dagger)\ ] \\ & \sigma \mid\mid\mid Run_H \text{ is not deterministic} \\ \Rightarrow & \hfill [\ \text{Lemma 5.4.5}\ ] \\ & P \mid\mid\mid Run_H \text{ is not deterministic} \end{array}$$

This contradicts the hypothesis that $P \mid\mid\mid Run_H$ is deterministic, i.e. no such m exists. Hence $\delta \in \mathit{traces}\,[\![\sigma' \setminus_H]\!]$.

(ii) Let $(\delta, K) \in \mathit{failures}\,[\![\sigma \setminus_H]\!]$. Suppose $\exists\, k \in K$ such that $\delta \frown \langle k\rangle \in \mathit{traces}\,[\![\sigma' \setminus_H]\!]$; trivially $k \notin H$. This implies, by Lemma 6.1.4(ii), that $\langle t\rangle \frown \delta \frown \langle k\rangle \in \mathit{traces}\,[\![\sigma \mid\mid\mid Run_H]\!]$. Now $\sigma \mid\mid\mid Run_H \xrightarrow{t} \sigma \mid\mid\mid Run_H$ since we can follow the t transition from Run_H.

$$\begin{array}{ll} & (\delta, K) \in \mathit{failures}\,[\![\sigma \setminus_H]\!] \\ \Rightarrow & \hfill [\ k \in K, \text{ Axiom F2}\ ] \\ & (\delta, \{k\}) \in \mathit{failures}\,[\![\sigma \setminus_H]\!] \\ \Rightarrow & \hfill [\ \text{Lemma 6.1.4(i)}\ ] \\ & (\delta, \{k\}) \in \mathit{failures}\,[\![\sigma \mid\mid\mid Run_H]\!] \\ \Rightarrow & \hfill [\ \sigma \mid\mid\mid Run_H \xrightarrow{t} \sigma \mid\mid\mid Run_H\ ] \\ & (\langle t\rangle \frown \delta, \{k\}) \in \mathit{failures}\,[\![\sigma \mid\mid\mid Run_H]\!] \end{array}$$

Hence $\sigma \mid\mid\mid Run_H$ is not deterministic; by 5.4.5 this once again contradicts the hypothesis that $P \mid\mid\mid Run_H$ is deterministic.

Parts (i) and (ii) prove that if $(\delta, K) \in \mathit{failures}\,[\![\sigma \setminus_H]\!]$ then $(\delta, K) \in \mathit{failures}\,[\![\sigma' \setminus_H]\!]$.

The "only if" part of the equivalence can be demonstrated by assuming $(\delta, K) \in \mathit{failures}\,[\![\sigma' \setminus_H]\!]$. If a shorter trace γ ($\gamma \frown \langle\varepsilon\rangle \frown \gamma' = \delta$) could lead to a refusal of ε in $\sigma \setminus_H$, the proof in (ii) gives us a contradiction. If after the trace δ, $\sigma \setminus_H$ is able to accept an element of K we get a contradiction from the proof in (i). $\square$

The main result, that the lazy security condition is equivalent to the local non-interference conditions for low-deterministic processes, follows from the preceding theorems, the results in Section 5.4 and those in Section 3.2.

Theorem 6.1.6 Let P be a non-divergent, low-deterministic process. The following conditions are equivalent :

1. $P \mid\mid\mid \mathit{Run}_H$ is deterministic.
2. $\mathsf{SLNI}_L^{FD}(P)$ holds.
3. $\mathsf{LNI}_L^{FD}(P)$ holds.
4. $\mathsf{SLNI}_L(P)$ holds.
5. $\mathsf{LNI}_L(P)$ holds.

Proof

($1 \equiv 2$) The equivalence of $\mathsf{SLNI}_L^{FD}(P)$ and $P \mid\mid\mid \mathit{Run}_H$ for deterministic processes, follows immediately from Theorem 6.1.3 ($\Rightarrow$) and Theorem 6.1.5 ($\Leftarrow$).

($2 \equiv 3$) Theorem 5.4.8 shows $\mathsf{SLNI}_L^{FD}(P)$ and $\mathsf{LNI}_L^{FD}(P)$ are equivalent for low-deterministic processes; this follows directly from Theorem 5.4.1 ($\Rightarrow$) and Theorem 5.4.7 ($\Leftarrow$).

($2 \equiv 4$) By Theorem 5.4.10.

($3 \equiv 5$) By Theorem 5.4.11. $\square$

Corollary 6.1.7 Theorem 6.1.6 holds if the requirement $\mathit{LowDet}(P)$ is replaced with $\mathit{L\text{-}Ind}(P)$ or $\mathit{Det}(P)$.

Proof This follows directly since *L-Ind*$(P) \Rightarrow$ *LowDet*(P) by Lemma 3.1.9, and *Det*$(P) \Rightarrow$ *LowDet*(P), by Lemma 3.1.10. $\square$

Further Comparisons

Corollary 6.1.7 says that, if L is independent in P, the lazy security condition agrees with the local non-interference conditions. It is shown in [Wul97] that independence and being lazily secure are equivalent. In other words, lazy security implies the local non-interference conditions.

Theorem 6.1.8 $\mathcal{L}_H(P)$ is deterministic implies $\mathsf{SLNI}_L(P)$. Similarly for the other local non-interference conditions.

Proof This follows directly from Corollary 6.1.7 and Theorem 3.2.17. $\square$

If local non-interference holds the three low determinism conditions given in Section 3.1 collapse.

Lemma 6.1.9 If P is a non-divergent process and $\mathsf{SLNI}_L^{FD}(P)$ (or $\mathsf{LNI}_L^{FD}(P)$) holds then *L-Ind*$(P) \equiv$ *L-Det*$(P) \equiv$ *LowDet*(P). Similarly for $\mathsf{SLNI}_L(P)$ or $\mathsf{LNI}_L(P)$.

Proof We have *L-Ind*$(P) \Rightarrow$ *L-Det*$(P) \Rightarrow$ *LowDet*(P) by Lemma 3.1.9. It is therefore sufficient to show *LowDet*$(P) \Rightarrow$ *L-Ind*(P).

Suppose, for a contradiction, $\exists\, a \in L$ and traces s, t of P where $s \upharpoonright L = t \upharpoonright L$ such that $(s, \{a\}) \in \mathit{failures}\,[\![P]\!]$ and $(P/t)^0 \cap \{a\} \neq \emptyset$. If $\mathsf{LNI}_L^{FD}(P)$ holds, by Theorem 5.4.8, $\mathsf{SLNI}_L^{FD}(P)$ holds.

$$\begin{array}{ll} & (P/t)^0 \cap \{a\} \neq \emptyset \\ \Rightarrow & \hfill [\text{ Axiom F2 }] \\ & (t \frown \langle a\rangle, \{\}) \in \mathit{failures}\,[\![P]\!] \\ \Rightarrow & \hfill [\ \mathsf{SLNI}_L^{FD}(P) \text{ holds, Lemma 6.1.2 }] \\ & (t \frown \langle a\rangle \upharpoonright L, \{\}) \in \mathit{failures}\,[\![P \setminus_H\ ]\!] \end{array}$$

Similarly, by Lemma 6.1.2, we have $(s \upharpoonright L, \{a\}) \in \mathit{failures}\,[\![P \setminus_H\ ]\!]$. Since $s \upharpoonright L = t \upharpoonright L$ this implies $P \setminus_H$ is not deterministic, but this contradicts the hypothesis that *LowDet*(P) holds.

The proof if $\mathsf{SLNI}_L(P)$ or $\mathsf{LNI}_L(P)$ holds follows from Theorem 6.1.6. $\Box$

This result does not extend to showing an equivalence with determinism. The low deterministic conditions do not restrict the high-level behaviours to being deterministic. It is still possible to build a process that satisfies a local non-interference condition and is low-deterministic, but fails to be deterministic.

Example 6.1.10 Let $L = \{l\}$ and $H = \{h\}$. Consider the process

$$P = (l \rightarrow h \rightarrow \mathit{Stop}) \Box (l \rightarrow \mathit{Stop})$$

This process is secure since the low-level user is always able to communicate a single l event. It is low-deterministic since $P \setminus_H = l \rightarrow \mathit{Stop}$. The process is not deterministic since h can be both accepted and refused after the trace $\langle l \rangle$.

(End of Example)

Finally we can show that the local non-interference conditions imply SBSNNI, which in turn was shown to imply BNDC [FG95]. We need to show that $\sigma \setminus_H \approx_B \sigma \setminus H$ for every state σ. A bisimulation relationship is formed which shows that these states are equivalent (as well as potentially related to other states in the system) and this will imply SBSNNI. The bisimulation is proved in two stages, firstly by showing that $t \notin H$ events from σ are matched and then for the τ labelled transitions caused by hiding $t \in H$ events.

Theorem 6.1.11 If $\mathsf{LNI}_L(P)$ holds then P satisfies SBSNNI. Similarly for $\mathsf{SLNI}_L(P)$.

Proof Let S be the maximum auto-bisimulation of P with respect to H. Let

$$R = \{(\delta \setminus_H, \sigma \setminus H) \mid (\sigma \setminus_H, \delta \setminus_H) \in S\}^1$$

Showing R is a bisimulation equivalence between P and P with respect to H will imply SBSNNI holds.

[1] If $\langle S, T, \alpha, \beta, \lambda \rangle$ is the LTS for P then $\langle S, T, \alpha, \beta, \lambda' \rangle$ is the LTS for $P \setminus X$. The notation $\sigma \setminus H$ is a shorthand for the state σ in $P \setminus X$ which corresponds to the sate σ in P. Similarly for $\sigma \setminus_H$.

Suppose $(\delta \setminus_H, \sigma \setminus H) \in R$.

If $\delta \setminus_H \xrightarrow{t} \delta' \setminus_H$ then, since $(\sigma \setminus_H, \delta \setminus_H) \in S$, there exists $\sigma' \setminus_H$ such that $\sigma \setminus_H \stackrel{\hat{t}}{\Longrightarrow} \sigma' \setminus_H$ and $(\sigma' \setminus_H, \delta' \setminus_H) \in S$. Hence, since (trivially) $t \notin H$, $\sigma \setminus H \stackrel{\hat{t}}{\Longrightarrow} \sigma' \setminus H$ and $(\delta' \setminus_H, \sigma' \setminus H) \in R$, as required.

If $\sigma \xrightarrow{t} \sigma'$ where $t \notin H$ then $\sigma \setminus H \xrightarrow{t} \sigma' \setminus H$. Since $(\sigma \setminus_H, \delta \setminus_H) \in S$ and $t \notin H$, there exists $\delta' \setminus_H$ such that $\delta \setminus_H \stackrel{\hat{t}}{\Longrightarrow} \delta' \setminus_H$ and $(\sigma' \setminus_H, \delta' \setminus_H) \in S$. Hence $(\delta' \setminus_H, \sigma' \setminus H) \in R$, as required.

Let $\sigma \xrightarrow{t} \sigma_1, \ldots, \sigma \xrightarrow{t} \sigma_n$ where $t \in H$, $\sigma \in \mathit{states}\,[\![P]\!]$ and the σ_i are a complete enumeration of the t-reachable states from σ.

Since $\sigma \setminus H \xrightarrow{\tau} \sigma_i \setminus H$ we need to show that $\exists\, \delta_i \bullet \delta \setminus_H \stackrel{\hat{\tau}}{\Longrightarrow} \delta_i \setminus_H$ where $(\delta_i \setminus_H, \sigma_i \setminus H) \in R$. Now

$$
\begin{array}{ll}
 & \sigma \setminus_H \\
\approx_B & \qquad\qquad\qquad\qquad\qquad\qquad [\ \mathsf{LNI}_L(P) \text{ holds}\ ] \\
 & \bigsqcap_i \sigma_i \setminus_H \\
\Rightarrow & \qquad\qquad\qquad\qquad\qquad\qquad [\ \text{Definition of bisimulation}\ ] \\
 & \forall\, i \,\exists\, \sigma_i' \setminus_H \bullet\, \sigma \setminus_H \stackrel{\hat{\tau}}{\Longrightarrow} \sigma_i' \setminus_H \wedge (\sigma_i' \setminus_H, \sigma_i \setminus_H) \in S
\end{array}
$$

Now $(\sigma \setminus_H, \delta \setminus_H) \in S$ and $\sigma \setminus_H \stackrel{\hat{\tau}}{\Longrightarrow} \sigma_i' \setminus_H$ implies there exists $\delta_i \setminus_H$ such that $\delta \setminus_H \stackrel{\hat{\tau}}{\Longrightarrow} \delta_i \setminus_H$ and $(\sigma_i' \setminus_H, \delta_i \setminus_H) \in S$. Since S is an equivalence relation, $(\sigma_i \setminus_H, \delta_i \setminus_H) \in S$. Hence $(\delta_i \setminus_H, \sigma_i \setminus H) \in R$ as required.

Hence R is a bisimulation equivalence and SBSNNI holds.

Since strong local non-interference implies local non-interference (Theorem 5.4.1) it follows immediately that the strong condition also implies SBSNNI. □

The reverse implication does not hold. For example, process P in Example 4.1.11 satisfies SBSNNI but not $\mathsf{SLNI}_L(P)$.

6.2 Hiding

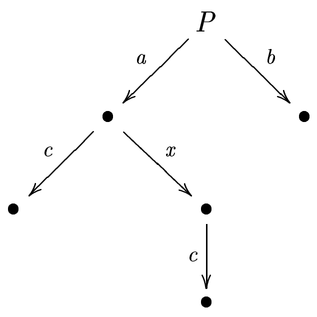
Given a security condition we may be interested in asking whether it is preserved by hiding. This would seem to be a reasonable property to expect of a condition. Hiding L labelled events provides the low-level user with less information about the system; if such a system was insecure we might expect the original system to be insecure, since the low-level user could ignore the information provided by such events.

Since H labelled events are already invisible to the low-level user, and hence from their perspective just another form of internal event, we might also expect to be able to hide them. The behaviours after H labelled events should also be acceptable system behaviours, else the condition might be expected to fail before hiding. Alternatively, as with the motivation for eager abstraction (3.2), hiding the H labelled events is an indication that they are always able to be communicated; a secure condition should not reveal information when the high-level user acts.

There are two facets of hiding which might cause problems for a condition. Firstly hiding can introduce nondeterminism into a system which was previously deterministic. Properties that rely on determinism, such as being lazily secure (Definition 3.2.8), will clearly fail if the low-level user is able to observe a nondeterminism in the resulting system. Any condition that is preserved after hiding L labelled events must be able to permit secure nondeterministic processes. We would expect the hiding of H labelled events to present no additional problems for any conditions, because the behaviours after such events should be acceptable system behaviours.

Example 6.2.1 Let $L = \{a, b, c\}, H = \{x\}$ and consider the process

$$\begin{array}{rl} P = & a \rightarrow (c \rightarrow Stop \,\Box\, x \rightarrow c \rightarrow Stop) \\ \Box & b \rightarrow Stop \end{array}$$



This process satisfies strong local non-interference and is lazily secure since

$$\mathcal{L}_H(P) = a \rightarrow (c \rightarrow Stop) \,\Box\, b \rightarrow Stop$$

The process $P \setminus H$ satisfies strong local non-interference (by Theorem 5.3.2) and is lazily secure since $\mathcal{L}_H(P) = \mathcal{L}_H(P \setminus H)$. $P \setminus \{c\}$ also satisfies both conditions, since

$$\mathcal{L}_H(P \setminus \{c\}) = a \rightarrow Stop \,\Box\, b \rightarrow Stop$$

The process $P \setminus \{a\}$ satisfies local non-interference (since the states around the x event are unchanged), but fails to be lazily secure since the event $\{b\}$ can be both accepted and refused after the empty trace. Similarly $P \setminus \{b\}$ fails to be lazily secure.

(End of Example)

Hiding can also introduce divergence into a process. The effect of divergence on the condition might be dependent on which model is being used. For example, under the failures-divergences model ($\mathcal{N}$) the resulting process would not be deterministic, since determinism requires the process to be divergence free; conditions that rely on determinism will necessarily fail. The weak-bisimulation equivalence does not treat divergences in the catastrophic manner of $\mathcal{N}$ and so this problem does not arise. The desirability, or otherwise, of such treatments of divergence will depend on whether we view the potential occurrence of an infinite number of internal/hidden events as a denial of service.

Example 6.2.2 Let $L = \{l\}, H = \{h\}$ and consider the process

$$P = l \rightarrow P \,\Box\, h \rightarrow P$$

This process satisfies strong local non-interference and is lazily secure [2]. The process $P \setminus X$ where $X \subseteq L \cup H$, satisfies strong local non-interference for all X, but is divergent if $X \neq \emptyset$ and so fails to be lazily secure.

(End of Example)

[2] Ensuring $P = h \rightarrow P$ does not cause divergence in the lazy abstraction, results in different definitions of the abstraction depending on the semantic model being used.

Example 6.2.3 Let P be an arbitrary process communicating in H and L. The process $P \setminus L$, which cannot communicate with the low-level user, satisfies strong local non-interference (Theorem 5.3.4). The process $P \setminus H$, which offers the high-level user no opportunity to affect the system, also satisfies strong local non-interference (Theorem 5.3.2). Both $P \setminus L$ and $P \setminus H$ might have nondeterminism or divergence, depending on P, and so neither is necessarily lazily secure.

(End of Example)

Hiding is a commutative operation that can be combined; in other words

$$\begin{aligned}(P \setminus Y) \setminus X &= (P \setminus X) \setminus Y \\ (P \setminus Y) \setminus X &= P \setminus (X \cup Y)\end{aligned}$$

This is true for the generated transition systems, as can be seen from the hiding firing rules. In order to show that hiding any (finite) number of events from a process P leads to a process that satisfies one of the local non-interference conditions if the original process P did, it is sufficient to show that any single event can be hidden and appeal to the above facts. This can be demonstrated by showing the same auto-bisimulation holds for both P and $P \setminus \{x\}$ under the hypothesis. The conditions will follow.

Hiding does not alter the number of states or transitions, or the mappings from states to transitions, in the LTS. The only effect is to alter some of the transition labellings; if we hide X then $x \in X$ labelled transitions are relabelled as τ events. [If $\langle S, T, \alpha, \beta, \lambda \rangle$ is the LTS for P then $\langle S, T, \alpha, \beta, \lambda' \rangle$ is the LTS for $P \setminus X$. Rather than having to add which process a given state is from, $\sigma \setminus X$ will be used as a shorthand for the state σ in process $P \setminus X$ which corresponds to the state σ in P.]

Lemma 6.2.4 If R is the maximal auto-bisimulation of E with respect to H then, if $l \in L$ and $h \in H$

(i) R is also an auto-bisimulation[3] for $E \setminus \{l\}$ with respect to H

(ii) If $\mathsf{SLNI}_L(E)$ holds then R is also an auto-bisimulation for $E \setminus \{h\}$ with respect to H

(iii) If $\mathsf{LNI}_L(E)$ holds then R is also an auto-bisimulation for $E \setminus \{h\}$ with respect to H

[3] R is not necessarily the *maximal* auto-bisimulation, since previously different states

Proof Let $(P, Q) \in R$. Since the bisimulation is with respect to H, the only transitions we need to verify for the equivalence are the $x \in L \cup \{\tau\}$ labelled ones.

(i) An $x \in L \cup \{\tau\}$ labelled transition in P can occur if and only if a corresponding $x \in L \cup \{\tau\}$ labelled transition can occur in $P \setminus \{l\}$; in the case where $x = l$ the corresponding transition in $P \setminus \{l\}$ is labelled with a τ.

Suppose $P \xrightarrow{\alpha} P', \alpha \notin H$. Since R is an auto-bisimulation for P with respect to H, we have $\exists\, Q'$ such that $Q \overset{\hat{\alpha}}{\Longrightarrow} Q'$ and $(P', Q') \in R$. If $\alpha \neq l$ we can use the same $\overset{\hat{\alpha}}{\Longrightarrow}$ path to give $Q \setminus \{l\} \overset{\hat{\alpha}}{\Longrightarrow} Q' \setminus \{l\}$. If $\alpha = l$ we can generate a path $\overset{\hat{\beta}}{\Longrightarrow}$ by following the same transitions as $\overset{\hat{\alpha}}{\Longrightarrow}$, the α labelled transition being now labelled τ, to give $Q \setminus \{l\} \overset{\hat{\beta}}{\Longrightarrow} Q' \setminus \{l\}$; the transitions (in $\hat{\beta}$) are of the form τ^*, as required.

The cases when $Q \xrightarrow{\alpha} Q', \alpha \notin H$ can be shown similarly. Hence R is an auto-bisimulation for $P \setminus \{l\}$ with respect to H, as required.

(ii) An $x \in L \cup \{\tau\}$ labelled transition in $P \setminus \{h\}$ can occur if and only if a corresponding $x \in L \cup \{\tau\}$ labelled transition can occur in P, *or* if an h labelled transition can occur in P. In the former case we can use an analogous proof to part (i) to show the bisimulation condition is not broken. Now

$$
\begin{array}{lll}
 & P \xrightarrow{h} P' & \\
\Rightarrow & & [\ \mathsf{SLNI}_L(P) \text{ holds}\ ] \\
 & P' \setminus_H \approx_B P \setminus_H & \\
\approx_B & & [\ (P, Q) \in R\ ] \\
 & Q \setminus_H &
\end{array}
$$

Hence when $P \xrightarrow{h} P'$ (and so $P \setminus \{h\} \xrightarrow{\tau} P' \setminus \{h\}$) there exists Q' such that $Q \setminus \{h\} \overset{\hat{\tau}}{\Longrightarrow} Q' \setminus \{h\}$ and $(P', Q') \in R$, namely $Q = Q'$. The cases where $Q \xrightarrow{h} Q'$ can be shown similarly. Hence R is an auto-bisimulation for $P \setminus \{h\}$ with respect to H as required.

(iii) An analogous argument to part (ii) leaves us to verify the cases

may now be equivalent. For example, $P = l \to \mathit{Stop}$ has two distinguishable states, but $P \setminus \{l\} = \mathit{Stop}$ has only one.

with multiple h labelled transitions. Let P be a state of E where $n > 1$, $P \xrightarrow{h} \sigma_1, \ldots, P \xrightarrow{h} \sigma_n$ and the σ_i are a complete enumeration of P's h-reachable states.

$$
\begin{array}{ll}
 & P \setminus_H \\
\approx_B & \hspace{12em} [\ \mathsf{LNI}_L(P) \text{ holds }] \\
 & \sigma_1 \setminus_H \sqcap \ldots \sqcap \sigma_n \setminus_H \\
\Rightarrow & \hspace{12em} [\ \text{Definition 2.2.7}\ ] \\
 & \forall\, i\, \exists\, P_i \bullet P \setminus_H \stackrel{\hat{\tau}}{\Longrightarrow} P_i \setminus_H \ \text{ and } P_i \setminus_H \approx_B \sigma_i \setminus_H
\end{array}
$$

We need to show that $\forall\, i\, \exists\, Q_i$ such that $(Q \setminus \{h\}) \stackrel{\hat{\tau}}{\Longrightarrow} (Q_i \setminus \{h\})$ and $(Q_i \setminus \{h\}) \setminus_H \approx_B (\sigma_i \setminus \{h\}) \setminus_H$. This follows from the above argument and the alternative path based definition of weak-bisimulation (Definition 2.2.10). $\square$

If we are hiding an event $t \in H$ the same relationship will not necessarily be an auto-bisimulation if we relax the requirement that one of the local non-interference conditions holds. This is because a state σ which could accept t, and may be bisimilar to another state σ', might be distinguishable once the t labelled transition becomes hidden and available. Since the bisimulation is with respect to H there is no requirement for σ', or any states leading from it, to ever communicate any H labelled events; i.e. hiding t will not necessarily affect them and keep them equivalent to σ. Without a local non-interference condition $\sigma \setminus \{t\}$ might present a different system view to the low-level user.

Example 6.2.5 Let $L = \{a, b\}, H = \{x\}$ and consider the process

$$
\begin{array}{rcl}
P & = & Q \sqcap Q' \\
Q & = & a \rightarrow \mathit{Stop} \\
Q' & = & a \rightarrow \mathit{Stop} \ \Box\ x \rightarrow b \rightarrow \mathit{Stop}
\end{array}
$$

If R is the maximal auto-bisimulation of P with respect to H then $(Q, Q') \in R$. Clearly $Q' \setminus \{x\}$ and Q are not bisimilar, since Q' can accept b whereas Q can only refuse it.

(End of Example)

We can now use the fact that hiding a single event in a secure process preserves the autobisimulation to show the local non-interference conditions are preserved by hiding.

Theorem 6.2.6 If $X \subseteq H \cup L$ then
(i) $\mathsf{SLNI}_L(P) \Rightarrow \mathsf{SLNI}_L(P \setminus X)$
(ii) $\mathsf{LNI}_L(P) \Rightarrow \mathsf{LNI}_L(P \setminus X)$

Proof The cases where X is a singleton will be proved explicitly. The results for when X is not a singleton set (or empty) follow immediately from the fact that $(P \setminus Y) \setminus X = P \setminus (X \cup Y)$.

Let R be the maximal auto-bisimulation of P with respect to H. Let $X = \{x\}$.

(i) Let $\sigma \setminus \{x\} \xrightarrow{t} \sigma' \setminus \{x\}$ where $t \in H$ and $\sigma \setminus \{x\} \in \mathit{states}\, [\![P \setminus \{x\}]\!]$. Since hiding only alters the labelling of x labelled transitions, $\sigma \in \mathit{states}\, [\![P]\!]$ and $\sigma \xrightarrow{t} \sigma'$.

$$
\begin{array}{ll}
 & \mathsf{SLNI}_L(P) \text{ holds} \\
\Rightarrow & \hfill [\text{ Definition 5.2.1 }] \\
 & \sigma \setminus_H \approx_B \sigma' \setminus_H \\
\Rightarrow & \hfill [\text{ Definition of R }] \\
 & (\sigma, \sigma') \in R \\
\Rightarrow & \hfill [\text{ Lemma 6.2.4 }] \\
 & (\sigma \setminus \{x\}) \setminus_H \approx_B (\sigma' \setminus \{x\}) \setminus_H
\end{array}
$$

(ii) Let $\sigma \setminus \{x\} \xrightarrow{t} \sigma_1 \setminus \{x\}, \ldots, \sigma \setminus \{x\} \xrightarrow{t} \sigma_n \setminus \{x\}$ where $t \in H$, $\sigma \setminus \{x\} \in \mathit{states}\, [\![P \setminus \{x\}]\!]$ and the σ_i are a complete enumeration of the t-reachable states from $\sigma \setminus \{x\}$. Since hiding only alters the labelling of x labelled transitions, $\sigma \in \mathit{states}\, [\![P]\!]$ and $\sigma \xrightarrow{t} \sigma_1, \ldots, \sigma \xrightarrow{t} \sigma_n$.

$$
\begin{array}{ll}
 & \mathsf{LNI}_L(P) \text{ holds} \\
\Rightarrow & \hfill [\text{ Definition 5.2.10 }] \\
 & \sigma \setminus_H \approx_B \sigma_1 \setminus_H \sqcap \ldots \sqcap \sigma_n \setminus_H \\
\Rightarrow & \hfill [\text{ Definition of bisimulation }] \\
 & \forall\, i\, \exists\, \bar{\sigma_i} \bullet \bar{\sigma_i} \setminus_H \approx_B \sigma_i \setminus_H \text{ and } \sigma \setminus_H \stackrel{\hat{\tau}}{\Longrightarrow} \bar{\sigma_i} \setminus_H \\
\Rightarrow & \hfill [\text{ Lemma 6.2.4 }] \\
 & \forall\, i \bullet (\bar{\sigma_i} \setminus \{x\}) \setminus_H \approx_B (\sigma_i \setminus \{x\}) \setminus_H
\end{array}
$$

Hence transitions from $\sqcap(\sigma_i \setminus \{x\}) \setminus_H$ are matched, as required, in $(\sigma \setminus \{x\}) \setminus_H$.

Suppose $(\sigma \setminus \{x\}) \setminus_H \xrightarrow{t} (\sigma' \setminus \{x\}) \setminus_H$ where $t \neq x$. Hence $\sigma \setminus_H \xrightarrow{t} \sigma' \setminus_H$, so there exists $\bar{\sigma} \setminus_H$ such that $\sqcap \sigma_i \setminus_H \stackrel{\hat{t}}{\Longrightarrow} \bar{\sigma} \setminus_H$ and $\sigma' \setminus_H \approx_B \bar{\sigma} \setminus_H$. Now $t \neq x$ implies $\sqcap(\sigma_i \setminus \{x\}) \setminus_H \stackrel{\hat{t}}{\Longrightarrow} (\bar{\sigma} \setminus \{x\}) \setminus_H$ and, by Lemma 6.2.4, $(\sigma' \setminus \{x\}) \setminus_H \approx_B (\bar{\sigma} \setminus \{x\}) \setminus_H$ as required.

An analogous argument holds when $(\sigma \setminus \{x\}) \setminus_H \xrightarrow{t} (\sigma' \setminus \{x\}) \setminus_H$, where $\sigma \xrightarrow{x} \sigma'$ and $x \notin H$.

Suppose $(\sigma \setminus \{x\}) \setminus_H \xrightarrow{\tau} (\delta_i \setminus \{x\}) \setminus_H$, where $\sigma \xrightarrow{x} \sigma'$ and $x \in H$. Let $\sigma \xrightarrow{x} \delta_1, \ldots, \sigma \xrightarrow{x} \delta_m$ where the δ_i are a complete enumeration of the x-reachable states from σ.

$$
\begin{array}{ll}
 & \mathsf{LNI}_L(P) \text{ holds} \\
\Rightarrow & \hfill [\text{ Definition 5.2.10 }] \\
 & \sigma \setminus_H \approx_B \delta_1 \setminus_H \sqcap \ldots \sqcap \delta_m \setminus_H \\
\Rightarrow & \hfill [\text{ Definition of bisimulation }] \\
 & \forall\, i \,\exists\, \bar{\delta}_i \bullet \bar{\delta}_i \setminus_H \approx_B \delta_i \setminus_H \text{ and } \sigma \setminus_H \stackrel{\hat{\tau}}{\Longrightarrow} \bar{\delta}_i \setminus_H \\
\Rightarrow & \hfill [\text{ Lemma 6.2.4 }] \\
 & \forall\, i \bullet (\bar{\delta}_i \setminus \{x\}) \setminus_H \approx_B (\delta_i \setminus \{x\}) \setminus_H
\end{array}
$$

Now $\sigma \setminus_H \approx_B \sqcap \sigma_i \setminus_H$, hence there exists $\psi_i \setminus_H$ such that $\sqcap \sigma_i \setminus_H \stackrel{\hat{\tau}}{\Longrightarrow} \psi_i \setminus_H$ and $\psi_i \setminus_H \approx_B \bar{\delta}_i \setminus_H$. The necessary equivalence follows from the transitivity of bisimulation and Lemma 6.2.4. □

The reverse theorem, if $P \setminus X$ satisfies SLNI (or LNI) then so does P, is clearly not true. The hiding of an L labelled event might remove the information that enables the low-level user to distinguish between two states. The hiding of an H labelled event might prevent the high-level user from being able to put the system into a distinguishable state. Example 6.2.3 shows that $P \setminus L$ and $P \setminus H$ will satisfy both conditions irrespective of the process P.

6.3 Parallel Composition

The advantages of having a security condition that is compositional are outlined in Section 5.1; the compositional property allows for the potential verification of larger systems and a wider applicability of the components. In this section the properties of the local non-interference conditions under parallel composition are considered.

When two processes are communicating we will use the generalized parallel operator, $P \underset{X}{\|} Q$, where P and Q are only required to syncronize over the events in X. This syncronization may include both high-level and low-level events. In the following, except where noted otherwise, let $X = \breve{H} \cup \breve{L}$ where $\breve{H} \subseteq H$ and $\breve{L} \subseteq L$.

Strong Local Non-Interference

The firing rules for parallel composition and restriction can be combined to show how restriction distributes through parallel composition, irrespective of the alphabets being used. Similary, parallel composition can be shown to preserve bisimilarity.

Lemma 6.3.1 $(P \underset{X}{\|} Q) \setminus_H \approx_B (P \setminus_H) \underset{X}{\|} (Q \setminus_H)$.

Proof We need to show that $\left((P \underset{X}{\|} Q) \setminus_H \ , \ (P \setminus_H) \underset{X}{\|} (Q \setminus_H)\right)$ is a bisimulation equivalence. This can be verified by comparing the firing rules for the two equations.

$$\frac{P \xrightarrow{\tau} P'}{(P \underset{X}{\|} Q) \setminus_H \xrightarrow{\tau} (P' \underset{X}{\|} Q) \setminus_H} \qquad \frac{P \xrightarrow{\tau} P'}{(P \setminus_H) \underset{X}{\|} (Q \setminus_H) \xrightarrow{\tau} (P' \setminus_H) \underset{X}{\|} (Q \setminus_H)}$$

$$\frac{Q \xrightarrow{\tau} Q'}{(P \underset{X}{\|} Q) \setminus_H \xrightarrow{\tau} (P \underset{X}{\|} Q') \setminus_H} \qquad \frac{Q \xrightarrow{\tau} Q'}{(P \setminus_H) \underset{X}{\|} (Q \setminus_H) \xrightarrow{\tau} (P \setminus_H) \underset{X}{\|} (Q' \setminus_H)}$$

For $t \in L \setminus \breve{L}$:

$$\frac{P \xrightarrow{t} P'}{(P \underset{X}{\parallel} Q) \setminus_H \xrightarrow{t} (P' \underset{X}{\parallel} Q) \setminus_H} \qquad \frac{P \xrightarrow{t} P'}{(P \setminus_H) \underset{X}{\parallel} (Q \setminus_H) \xrightarrow{t} (P' \setminus_H) \underset{X}{\parallel} (Q \setminus_H)}$$

$$\frac{Q \xrightarrow{t} Q'}{(P \underset{X}{\parallel} Q) \setminus_H \xrightarrow{t} (P \underset{X}{\parallel} Q') \setminus_H} \qquad \frac{Q \xrightarrow{t} Q'}{(P \setminus_H) \underset{X}{\parallel} (Q \setminus_H) \xrightarrow{t} (P \setminus_H) \underset{X}{\parallel} (Q' \setminus_H)}$$

For $t \in \breve{L}$:

$$\frac{P \xrightarrow{t} P', Q \xrightarrow{t} Q'}{(P \underset{X}{\parallel} Q) \setminus_H \xrightarrow{t} (P' \underset{X}{\parallel} Q') \setminus_H} \qquad \frac{P \xrightarrow{t} P', Q \xrightarrow{t} Q'}{(P \setminus_H) \underset{X}{\parallel} (Q \setminus_H) \xrightarrow{t} (P' \setminus_H) \underset{X}{\parallel} (Q' \setminus_H)}$$

Neither process is able to communicate any H labelled events. $\square$

Lemma 6.3.2 If $P \approx_B Q$ then $P \underset{X}{\parallel} R \approx_B Q \underset{X}{\parallel} R$.

The distributivity result can be used to show that strong local non-interference is preserved by parallel composition over any alphabet. This is a strong result since it allows the processes to have different high-level and low-level events as well as common events which may, or may not, require synchronisation. When the combined process communicates over a shared event the separate security of the two processes ensures the resulting system moves to a bisimilar state. When no syncronisation is required the security of the component process making the transition is sufficient to ensure the composition is secure.

Theorem 6.3.3 If $\mathsf{SLNI}_L(P)$ and $\mathsf{SLNI}_L(Q)$ hold then $\mathsf{SLNI}_L(P \underset{X}{\parallel} Q)$ holds.

Proof Let $P' \underset{X}{\parallel} Q'$ be a state of $P \underset{X}{\parallel} Q$.

(Case 1) Either P' or Q' is unable to communicate a $t \in \breve{H}$ labelled transition. Since P' and Q' are required to agree on such events, $P' \underset{X}{\parallel} Q'$ can have no t labelled transitions; hence the condition is not broken.

(Case 2) Both P' and Q' can communicate $t \in \breve{H}$, i.e. $\exists\, P'', Q''$ such that $P' \xrightarrow{t} P''$ and $Q' \xrightarrow{t} Q''$. We need to show $(P' \underset{X}{\parallel} Q') \setminus_H \approx_B (P'' \underset{X}{\parallel} Q'') \setminus_H$. Using Lemma 6.3.2:

$$
\begin{array}{ll}
& (P' \underset{X}{\parallel} Q') \setminus_H \\
\approx_B & \hfill [\text{ Lemma 6.3.1 }] \\
& (P' \setminus_H) \underset{X}{\parallel} (Q' \setminus_H) \\
\approx_B & \hfill [\ \mathsf{SLNI}_L(P) \text{ holds, Theorem 5.3.6 }] \\
& (P'' \setminus_H) \underset{X}{\parallel} (Q' \setminus_H) \\
\approx_B & \hfill [\ \mathsf{SLNI}_L(Q) \text{ holds, Theorem 5.3.6 }] \\
& (P'' \setminus_H) \underset{X}{\parallel} (Q'' \setminus_H) \\
\approx_B & \hfill [\text{ Lemma 6.3.1 }] \\
& (P'' \underset{X}{\parallel} Q'') \setminus_H
\end{array}
$$

(Case 3) $\exists\, P''$ such that $P' \xrightarrow{t} P''$ where $t \in H \setminus \breve{H}$. By a similar argument to the one above we can show $(P' \underset{X}{\parallel} Q') \setminus_H \approx_B (P'' \underset{X}{\parallel} Q') \setminus_H$ as required. The case when $\exists\, Q''$ such that $Q' \xrightarrow{t} Q''$ where $t \in H \setminus \breve{H}$ is analogous. [Since no synchronisation is required for $t \in H \setminus \breve{H}$ the case where both P' and Q' can communicate such a t event is covered under case 3.] □

Lemmas 6.3.1 and 6.3.2 can also be shown to hold, for non-divergent processes, when bisimulation is replaced by failures/divergences equivalence. An analogous proof to that in Theorem 6.3.3 can then be used to show the theorem holds with both equivalences.

Theorem 6.3.4 For non-divergent processes P and Q, if $\mathsf{SLNI}_L^{FD}(P)$ and $\mathsf{SLNI}_L^{FD}(Q)$ hold then $\mathsf{SLNI}_L^{FD}(P \underset{X}{\parallel} Q)$ holds.

We can think of parallel composition in two ways: firstly as the bringing together of two separate systems and secondly as the composition of a system and a user. We expect a secure system to prevent any information flow irrespective of how the users of that system behave. At the very least, the composition of a process with any exclusively high-level or low-level process should be secure; all we have done is placed some restrictions on how one of the users is behaving.

Conversely, if a process is secure however a user of that system behaves, i.e. whenever a process representing a user is placed in parallel with it, the system as a whole is secure. With a high-level user in parallel the system is insecure if the (unhindered) low-level user can obtain information from a potentially restricted set of high-level behaviours; any insecurity will be present in the full system because the same high-level behaviours are available. This is similar to the compositional approach with non-deducibility on strategies. An insecurity when the process is in parallel with a restricted low-level process indicates the low-level user can gain information without needing to exploit all the potential low-level behaviours.

The equivalence of these conditions is shown in the following theorem:

Theorem 6.3.5 The following conditions are equivalent:

1. $\mathsf{SLNI}_L(P)$ holds
2. $\mathsf{SLNI}_L(P \underset{X}{\parallel} Q)$ holds $\forall\, X, Q$ with $\Sigma Q \subseteq H$
3. $\mathsf{SLNI}_L(P \underset{X}{\parallel} Q)$ holds $\forall\, X, Q$ with $\Sigma Q \subseteq L$.

Proof

$(2 \Rightarrow 1)$ $\mathsf{SLNI}_L(P \underset{H}{\parallel} Run_H) \Rightarrow \mathsf{SLNI}_L(P)$, since $P \underset{H}{\parallel} Run_H \equiv P$.

$(3 \Rightarrow 1)$ $\mathsf{SLNI}_L(P \underset{L}{\parallel} Run_L) \Rightarrow \mathsf{SLNI}_L(P)$, since $P \underset{L}{\parallel} Run_L \equiv P$.

$(1 \Rightarrow 2)$ If $\Sigma Q \subseteq H$ then $\mathsf{SLNI}_L(Q)$ holds by Theorem 5.3.4. Hence, by Theorem 6.3.3, $\mathsf{SLNI}_L(P \underset{H}{\parallel} Q)$ holds as required.

$(1 \Rightarrow 3)$ If $\Sigma Q \subseteq L$ then $\mathsf{SLNI}_L(Q)$ holds by Theorem 5.3.2. Hence, by Theorem 6.3.3, $\mathsf{SLNI}_L(P \underset{H}{\parallel} Q)$ holds as required. $\square$

Example 6.3.6 Let $\mathsf{SLNI}_L(P)$ hold. Let Q_H represent a high-level user who only communicates with process P using the high-level alphabet H; similarly let Q_L be a low-level user communicating only in L. The combined system

$$(P \underset{H}{\parallel} Q_H)$$

is secure. In particular, this is true for

$$\begin{aligned} SYS_1 &= P \underset{H}{\parallel} \mathit{Stop} \\ SYS_2 &= P \underset{H}{\parallel} \mathit{Run}_H \\ SYS_3 &= P \underset{H}{\parallel} \mathit{Chaos}_H \end{aligned}$$

i.e. when the high-level user never communicates (SYS_1), is always willing to communicate (SYS_2) and when the high-level user communicates in a very nondeterministic manner (e.g. SYS_3). Similarly

$$P \underset{L}{\parallel} Q_L$$

must always be secure.

Furthermore, since $\mathsf{SLNI}_L(P \underset{H}{\parallel} Q_H)$ and $\mathsf{SLNI}_L(P \underset{L}{\parallel} Q_L)$ hold we have the processes

$$\begin{aligned} \mathit{LOWHIGH} &= (P \underset{H}{\parallel} Q_H) \underset{L}{\parallel} Q_L \\ \mathit{HIGHLOW} &= (P \underset{L}{\parallel} Q_L) \underset{H}{\parallel} Q_H \end{aligned}$$

also satisfy SLNI. In other words, no combinations of high-level and low-level behaviours can lead to the resulting system failing to satisfy the strong local non-interference property.

(End of Example)

If $\mathsf{SLNI}_L(P)$ holds and $\Sigma Q \subseteq H \cup L$ it is interesting to ask under what circumstances $\mathsf{SLNI}_L(P \underset{X}{\parallel} Q)$ is satisfied. If the alphabet Q communicates over is restricted to a subset of L, or a subset of H, it has already been demonstrated that the composition satisfies the condition, irrespective of the events P and Q must synchronise on. The results cannot be extended to arbitrary processes Q, even if X is restricted to H or L.

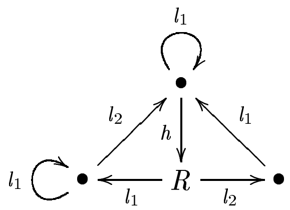
Example 6.3.7 Let $L = \{l_1, l_2\}, H = \{h\}$ and consider the processes

$$\begin{aligned} P &= (l_1 \to P) \Box (h \to l_1 \to P) \\ Q_1 &= h \to l_1 \to Q_1 \\ Q_2 &= h \to l_2 \to Q_2 \end{aligned}$$

$\mathsf{SLNI}_L(P)$ holds since P can always communicate an l_1. However

$$\begin{aligned} P \underset{L}{\|} Q_1 = & \quad h \to h \to l_1 \to (P \underset{L}{\|} Q_1) \\ & \Box \; h \to l_1 \to (P \underset{L}{\|} Q_1) \end{aligned}$$

$$\begin{aligned} P \underset{H}{\|} Q_2 &= \quad l_1 \to (P \underset{H}{\|} Q_2) \Box h \to R \\ R &= \quad l_2 \to l_1 \to P \underset{H}{\|} Q_2 \\ & \Box \; l_1 \to \mu X.(l_1 \to X \Box l_2 \to P \underset{H}{\|} Q_2) \end{aligned}$$



Neither of these processes satisfies SLNI, nor would we wish to classify them as secure.

(End of Example)

Local Non-Interference

For local non-interference we do not have as strong a result as Theorem 6.3.3. In other words we cannot take two arbitrarily secure process and combine them to make another secure process. The distributivity result in Lemma 6.3.1 does not extend to the nondeterministic choice of multiple parallel states. Nevertheless we can show that the composition of a process P which satisfies local non-interference and a process Q that only communicates in H or L is secure.

Example 6.3.8 Let $L = \{l_1, l_2, l_3\}$ and consider the processes

$$\begin{aligned} P_1 &= l_1 \to Stop \\ P_2 &= l_2 \to Stop \\ Q &= l_3 \to Stop \end{aligned}$$

The process $(P_1 \underset{\{\}}{\parallel} Q) \setminus_H \sqcap (P_2 \underset{\{\}}{\parallel} Q) \setminus_H$ makes its nondeterministic choice before process Q has the opportunity to communicate an l_3 event. In particular, once an l_3 has been communicated the availability of any other event has already been determined. The process $(P_1 \setminus_H \sqcap P_2 \setminus_H) \underset{\{\}}{\parallel} Q \setminus_H$ might resolve its nondeterminism after the l_3 event.

(End of Example)

Lemma 6.3.1 can be extended somewhat if we restrict Q to only communicating only in H, whilst still allowing composition over an arbitrary alphabet. This can then be used to show how some states in $P \underset{X}{\parallel} Q$, where P satisfies local non-interference, are related by bisimulation relationships.

Lemma 6.3.9 If $\Sigma Q \subseteq H$ then

$$(P_1 \underset{X}{\parallel} Q) \setminus_H \sqcap \ldots \sqcap (P_n \underset{X}{\parallel} Q) \setminus_H \approx_B (P_1 \setminus_H \sqcap \ldots \sqcap P_n \setminus_H) \underset{X}{\parallel} Q \setminus_H$$

Proof Now $Q \setminus_H \approx_B \mathit{Stop}$ and $P \underset{X}{\parallel} \mathit{Stop} \mathrel{\hat=} P \setminus_X$ (†)

$$\begin{array}{ll}
 & (P_1 \setminus_H \sqcap \ldots \sqcap P_n \setminus_H) \underset{X}{\parallel} Q \setminus_H \\
\approx_B & \hfill [\ (\dagger)\ ] \\
 & (P_1 \setminus_H \sqcap \ldots \sqcap P_n \setminus_H) \setminus_X \\
\approx_B & \hfill [\ \setminus_X \text{ distributes through } \sqcap\ ] \\
 & P_1 \setminus_H \setminus_X \sqcap \ldots \sqcap P_n \setminus_H \setminus_X \\
\approx_B & \hfill [\ (\dagger)\ ] \\
 & P_1 \setminus_H \underset{X}{\parallel} Q \setminus_H \sqcap \ldots \sqcap P_n \setminus_H \underset{X}{\parallel} Q \setminus_H \\
\approx_B & \hfill [\ \text{Lemma 6.3.1}\ ] \\
 & (P_1 \underset{H}{\parallel} Q) \setminus_H \sqcap \ldots \sqcap (P_n \underset{H}{\parallel} Q) \setminus_H
\end{array}$$

□

Corollary 6.3.10 If $\Sigma Q \subseteq H$, $\mathsf{LNI}_L(P)$ holds, $P' \in \mathit{states}\,[\![P]\!]$, $t \in H$ and $P' \xrightarrow{t} P_1, \ldots, P' \xrightarrow{t} P_n$ where the P_i are a complete enumeration of the t-reachable states from P', then

$$(P' \underset{X}{\parallel} Q) \setminus_H \approx_B (P_1 \underset{X}{\parallel} Q) \setminus_H \sqcap \ldots \sqcap (P_n \underset{X}{\parallel} Q) \setminus_H$$

Proof Follows directly from Theorem 5.3.6 and Lemma 6.3.9. □

The first compositional result we can show for local non-interference is the corresponding version of the equivalence between properties (i) and (ii) in Theorem 6.3.5. In other words, we can apply restrictions to a secure process' high-level events, or effectively add new obfuscating H labelled events, without making a secure process insecure. Similarly any process that allows such modifications and always remains secure must itself be secure.

Parallel composition over X may include a subset of L and will have the additional effect of restricting some low-level events. This should not cause any problems because it is uniformally applied. In other words, since $\Sigma Q \subseteq H$ the restriction of L labelled events is independent of the high-level process behaviour.

Theorem 6.3.11 $\mathsf{LNI}_L(P)$ holds if and only if $\mathsf{LNI}_L(P \underset{X}{\parallel} Q)$ holds $\forall\, X, Q$ with $\Sigma Q \subseteq H$.

Proof

($\Leftarrow$) $\mathsf{SLNI}_L(P \underset{H}{\parallel} Run_H) \Rightarrow \mathsf{SLNI}_L(P)$, since $P \underset{H}{\parallel} Run_H \equiv P$.

($\Rightarrow$) Let $P' \underset{X}{\parallel} Q'$ be a state of $P \underset{X}{\parallel} Q$.

(Case 1) Either P' or Q' is unable to communicate an $\breve{H}$ labelled transition. Since P' and Q' are required to agree on such events, $P' \underset{X}{\parallel} Q'$ can have no $\breve{H}$ labelled transitions; hence the condition is not broken.

(Case 2) Both P' and Q' can communicate $t \in \breve{H}$, i.e. $\exists\, P_1, \ldots, P_n$ such that $P' \xrightarrow{t} P_1, \ldots, P' \xrightarrow{t} P_n$ and the P_i are a complete enumeration of the t-reachable states from P', and similarly $\exists\, Q_1, \ldots, Q_m$ such that $Q' \xrightarrow{t} Q_1, \ldots, Q' \xrightarrow{t} Q_m$. We need to show

$$(P' \underset{X}{\parallel} Q') \setminus_H \approx_B \sqcap_{i,j} (P_i \underset{X}{\parallel} Q_j) \setminus_H$$

Now $\Sigma Q \subseteq H$ so, by Lemma 5.3.4, $\forall\, i \bullet Q_i \setminus_H \approx_B Q' \setminus_H \approx_B \mathit{Stop}$ (†).

$$
\begin{array}{lll}
 & \bigsqcap_{i,j} (P_i \underset{X}{\parallel} Q_j) \setminus_H & \\
\approx_B & & [\text{ Lemma 6.3.1 }] \\
 & \bigsqcap_{i,j} P_i \setminus_H \underset{X}{\parallel} Q_j \setminus_H & \\
\approx_B & & [\ (\dagger)\ ] \\
 & \bigsqcap_{i,j} P_i \setminus_H \underset{X}{\parallel} Q' \setminus_H & \\
\approx_B & & [\ P \sqcap P \approx_B P\ ] \\
 & \bigsqcap_{i} P_i \setminus_H \underset{X}{\parallel} Q' \setminus_H & \\
\approx_B & & [\text{ Lemma 6.3.9 }] \\
 & (\bigsqcap_{i} P_i \setminus_H) \underset{X}{\parallel} Q' \setminus_H & \\
\approx_B & & [\ \mathsf{LNI}_L(P) \text{ holds }] \\
 & P' \setminus_H \underset{X}{\parallel} Q' \setminus_H & \\
\approx_B & & [\text{ Lemma 6.3.1 }] \\
 & (P' \underset{X}{\parallel} Q') \setminus_H &
\end{array}
$$

(Case 3) P' and/or Q' are able to communicate $t \in H \setminus \breve{H}$. We need to show that

$$
\begin{array}{rl}
(P' \underset{X}{\parallel} Q') \setminus_H \approx_B & (P_1 \underset{X}{\parallel} Q') \setminus_H \sqcap \ldots \sqcap (P_n \underset{X}{\parallel} Q') \setminus_H \\
\sqcap & (P' \underset{X}{\parallel} Q_1) \setminus_H \sqcap \ldots \sqcap (P' \underset{X}{\parallel} Q_m) \setminus_H
\end{array}
$$

From Corollary 6.3.10 we have $(P_1 \underset{X}{\parallel} Q') \setminus_H \sqcap \ldots \sqcap (P_n \underset{X}{\parallel} Q') \setminus_H$ is bisimilar to $(P' \underset{X}{\parallel} Q') \setminus_H$. We have $(P' \underset{X}{\parallel} Q_i) \setminus_H \approx_B (P' \underset{X}{\parallel} Q') \setminus_H$ for all i by (†). Hence the condition holds. □

The second compositional result shows how a low-level process can be composed with a process that satisfies local non-interference. It is not as strong as the result shown in Theorem 6.3.11 since we are no longer able to compose over an arbitrary alphabet. The two processes must syncronise over all events in the low-level process, otherwise it is possible for a low-level user to distinguish between the 'during' and 'after' types of nondeterminism discussed on page 87. This relates to the interleaving of such events and is discussed further in Section 6.4.

Lemma 6.3.12 Let $X = \breve{H} \cup L$ where $\breve{H} \subseteq H$. If $\Sigma Q \subseteq L$ then

$$(P_1 \underset{X}{\parallel} Q) \setminus_H \sqcap \ldots \sqcap (P_n \underset{X}{\parallel} Q) \setminus_H \approx_B (P_1 \setminus_H \sqcap \ldots \sqcap P_n \setminus_H) \underset{X}{\parallel} Q \setminus_H$$

Proof The bisimulation equivalence can be verified by considering the firing rules. The LHS can only perform a τ event which will take it to one of the $(P_i \underset{X}{\parallel} Q) \setminus_H$ states. Since $L \subseteq X$ the $Q \setminus_H$ process needs to syncronise with $(P_1 \setminus_H \sqcap \ldots \sqcap P_n \setminus_H)$ for all visible events. Hence the only available transitions on the RHS is a τ events to one of the $P_i \setminus_H \underset{X}{\parallel} Q \setminus_H$ states. Lemma 6.3.1 can now be used to show the equivalence. □

Theorem 6.3.13 Let $X = \breve{H} \cup L$ where $\breve{H} \subseteq H$. $\mathsf{LNI}_L(P)$ holds if and only if $\mathsf{LNI}_L(P \underset{X}{\parallel} Q)$ holds $\forall X, Q$ with $\Sigma Q \subseteq L$.

Proof Let $P' \underset{X}{\parallel} Q'$ be a state of $P \underset{X}{\parallel} Q$. Since $\Sigma Q \subseteq L$ the only case we need to consider is when P' can communicate $t \in \breve{H}$, i.e. $\exists P_1, \ldots, P_n$ such that $P' \xrightarrow{t} P_1, \ldots, P' \xrightarrow{t} P_n$ and the P_i are a complete enumeration of the t-reachable states from P'.

We need to show that

$$(P' \underset{X}{\parallel} Q') \setminus_H \approx_B (P_1 \underset{X}{\parallel} Q) \setminus_H \sqcap \ldots \sqcap (P_n \underset{X}{\parallel} Q) \setminus_H$$

This follows directly from Theorem 5.3.6 and Lemma 6.3.12. □

Stronger results can be shown for the failures/divergences version of local non-interference. The distributivity result in Lemma 6.3.1 can be extended, for non-divergent processes, over the nondeterministic choice of multiple parallel states. This result can then be used to show that the parallel composition of non-divergent LNI^{FD}processes also satisfies LNI^{FD}.

Lemma 6.3.14 For non-divergent processes P_i and Q_j

$$\sqcap_{i,j}(P_i \setminus_H \underset{X}{\parallel} Q_j \setminus_H) =_{\mathrm{FD}} (\sqcap_i P_i \setminus_H) \underset{X}{\parallel} (\sqcap_j Q_j \setminus_H)$$

Proof The failures of $P \underset{X}{\parallel} Q$ can be defined in terms of the failures of P and Q. For non-divergent processes

$$
\begin{aligned}
\mathit{failures}\,[\![P \underset{X}{\parallel} Q]\!] = \{ \quad & (u, Y \cup Z) \mid Y \setminus X = Z \setminus X \\
\wedge \quad & \exists\, s, t.(s, Y) \in \mathit{failures}\,[\![P]\!] \\
\wedge \quad & (t, Z) \in \mathit{failures}\,[\![Q]\!] \\
\wedge \quad & u \in s \underset{X}{\parallel} t\}
\end{aligned}
$$

The lemma follows by simple rearrangement of the failures. □

Theorem 6.3.15 For non-divergent P and Q, if $\mathsf{LNI}_L^{FD}(P)$ and $\mathsf{LNI}_L^{FD}(Q)$ hold then $\mathsf{LNI}_L^{FD}(P \underset{X}{\parallel} Q)$ holds.

Proof Let $P' \underset{X}{\parallel} Q'$ be a state of $P \underset{X}{\parallel} Q$.

(Case 1) Either P' or Q' is unable to communicate an $\breve{H}$ labelled transition. Since P' and Q' are required to agree on such events, $P' \underset{X}{\parallel} Q'$ can have no $\breve{H}$ labelled transitions; hence the condition is not broken.

(Case 2) Both P' and Q' can communicate $t \in \breve{H}$, i.e. $\exists\, P_1, \ldots, P_n$ such that $P' \xrightarrow{t} P_1, \ldots, P' \xrightarrow{t} P_n$ and the P_i are a complete enumeration of the t-reachable states from P', and similarly $\exists\, Q_1, \ldots, Q_m$ such that $Q' \xrightarrow{t} Q_1, \ldots, Q' \xrightarrow{t} Q_m$.

$$
\begin{aligned}
& \textstyle\sqcap_{i,j} (P_i \underset{X}{\parallel} Q_j) \setminus_H \\
=_{\mathrm{FD}} & \qquad\qquad\qquad\qquad [\text{ Lemma 6.3.1 }] \\
& \textstyle\sqcap_{i,j} P_i \setminus_H \underset{X}{\parallel} Q_j \setminus_H \\
=_{\mathrm{FD}} & \qquad\qquad\qquad\qquad [\text{ Lemma 6.3.14 }] \\
& (\textstyle\sqcap_i P_i \setminus_H) \underset{X}{\parallel} (\textstyle\sqcap_j Q_j \setminus_H) \\
=_{\mathrm{FD}} & \qquad\qquad\qquad\qquad [\ \mathsf{LNI}_L^{FD}(P) \text{ and } \mathsf{LNI}_L^{FD}(Q) \text{ hold }] \\
& P' \setminus_H \underset{X}{\parallel} Q' \setminus_H \\
=_{\mathrm{FD}} & \qquad\qquad\qquad\qquad [\text{ Lemma 6.3.1 }] \\
& (P' \underset{X}{\parallel} Q') \setminus_H
\end{aligned}
$$

(Case 3) P' and/or Q' are able to communicate $t \in H \setminus \breve{H}$. We need to show that

$$(P' \underset{X}{\parallel} Q') \setminus_H =_{\mathrm{FD}} \quad (P_1 \underset{X}{\parallel} Q') \setminus_H \sqcap \ldots \sqcap (P_n \underset{X}{\parallel} Q') \setminus_H$$
$$\sqcap \quad (P' \underset{X}{\parallel} Q_1) \setminus_H \sqcap \ldots \sqcap (P' \underset{X}{\parallel} Q_m) \setminus_H$$

Using analogous reasoning to case 2 we can show that

$$(P_1 \underset{X}{\parallel} Q') \setminus_H \sqcap \ldots \sqcap (P_n \underset{X}{\parallel} Q') \setminus_H$$

is equivalent to $(P' \underset{X}{\parallel} Q') \setminus_H$. Similarly for the second term. The condition then follows. $\square$

6.4 Separability

If two systems communicate in different alphabets then there are no events for their parallel composition to synchronise on and no opportunity for one process to interfere with the other. If one of these processes represents the behaviours of a low-level user communicating in L and the second process the behaviours of a high-level user communicating in H, a process constructed as their composition would be expected to be secure. A user observing the system through L (or H) would observe the corresponding component process.

Separability is concerned with systems that can be decomposed into a high-level and a low-level process; such systems could be built as the composition of these processes, which would make them secure. It originated with Rushby [Rus81b] as a way of resolving problems in the verification of kernelized systems. The idea was that "the environment provided by a separation kernel should be indistinguishable from that of an isolated machine". This theory was corrected slightly by Jacob in [Jac90b] and discussed for covert channels in [Jac90a] (where [Rus81c] is also referenced).

Separability can be simply put into a CSP context by using the interleaving operator. We can also defined a stronger condition [RW95] which requires the component processes to be deterministic.

Definition 6.4.1 Let P be a process whose alphabet is partitioned by $\{H, L\}$. P is said to be *separable* with respect to $\{H, L\}$ if there exists processes P_H and P_L, where $\Sigma P_H = H$ and $\Sigma P_L = L$, such that

$$P = P_H \mathbin{|||} P_L$$

Definition 6.4.2 Let P be a process whose alphabet is partitioned by $\{H, L\}$. P is said to be *strongly separable* with respect to $\{H, L\}$ if there exists deterministic processes P_H and P_L, where $\Sigma P_H = H$ and $\Sigma P_L = L$, such that

$$P = P_H \mathbin{|||} P_L.$$

One immediate problem with separability as a condition is that it is symmetric, whereas we are interested in an asymmetric information flow property. Weaker asymmetric versions of the condition exist, such as semi-separability [Wul97] and the formulation given in [ZL97]. Such conditions allow P_H to communicate in H and L and rely on the two processes synchronising over L labelled events. Some results on the compositions of such processes are given in Theorem 6.3.5 and Theorem 6.3.13.

The interpretation of Definition 6.4.1 depends on how we are interpreting interleaving and equivalence. If we are using the full CSP semantics separability is not a sufficient condition for security because the condition only requires semantic equivalence to the interleaved processes. Nondeterminism within such a system can cause it to be insecure. If we take the definition to be using a bisimulation type equivalence, we can show that separability is sufficient to show strong local non-interference, since it shows a system built as the interleaving of two processes.

The results of Section 6.3 apply to interleaving.

Lemma 6.4.3 $(P \mathbin{|||} Q) \setminus_H \approx_B (P \setminus_H) \mathbin{|||} (Q \setminus_H)$.

Proof This is a proved by Lemma 6.3.1 when $X = \{\}$. $\square$

Theorem 6.4.4 If $\mathsf{SLNI}_L(P)$ and $\mathsf{SLNI}_L(Q)$ hold then $\mathsf{SLNI}_L(P \mathbin{|||} Q)$ holds.

Proof This is proved by Theorem 6.3.3 when $X = \{\}$. $\square$

Example 6.4.5 The processes $Chaos_L$ and $Chaos_H$ satisfy $\mathsf{SLNI}_L(P)$ (by Theorems 5.3.4 and 5.3.2 respectively) and hence by Theorem 6.4.4 so does $Chaos_L ||| Chaos_H$[4].

(End of Example)

A process is separable if it can be equivalently expressed as the interleaving of a high-level and a low-level process. It is shown in [Wul97, Proposition 3.3.2] that a separable process, despite appearing to have the potential for being represented by several high-level and low-level processes, can only be the interleaving of the lazy abstractions with respect to L and H respectively.

§Proposition 6.4.6 If process P is separable, then $\mathcal{L}_H(P)$ describes the behaviour in alphabet L, and $\mathcal{L}_L(P)$ describes the behaviour in alphabet H. In other words

$$P = \mathcal{L}_H(P) ||| \mathcal{L}_L(P)$$

If P is a separable process it does not follow that any transition system representing P satisfies either of the local non-interference conditions. In particular, the LTS generated from the CSP firing rules (applied to P) is not necessarily secure. Separability does however give us that there exists at least one secure LTS (and corresponding CSP definition), namely the interleaving of the high and low lazy abstractions. The proof of this is by construction.

Corollary 6.4.7 If process P is separable there exists a semantically equivalent transition system Q such that $\mathsf{SLNI}_L(Q)$ holds.

Proof $P = \mathcal{L}_L(P) ||| \mathcal{L}_H(P)$ by Proposition 6.4.6. Let P_H and P_L be any processes that are semantically equivalent to $\mathcal{L}_H(P)$ and $\mathcal{L}_L(P)$ respectively. $Q = P_H ||| P_L$ satisfies $\mathsf{SLNI}_L(Q)$ by Theorem 6.4.4. □

Example 6.4.8 The process $Q = Chaos_{L\cup H} \sqcap P$ does not satisfy SLNI_L if $\mathsf{SLNI}_L(P)$ is not satisfied (Theorem 5.3.6). Now $Chaos_{L\cup H}$ is separable since

[4]This result holds when the transition system of the interleaved process is formed according to the firing rules. It is certainly not necessarily true for a process that is equivalent under the usual CSP semantics. For more details see section 7.1.

it is semantically equivalent to $Q' = \mathit{Chaos}_L \mid\mid\mid \mathit{Chaos}_H$; Q' and Q are also semantically equivalent. As shown in Example 6.4.5, $\mathsf{SLNI}_L(Q')$ holds for all transition systems representing Chaos_L and Chaos_H, when generating according to the firing rules.

(End of Example)

For a general separable P we have shown it is not necessarily the case that P itself, as opposed to a semantically equivalent P', is secure. If P is strongly separable not only is P secure, but so is every semantically equivalent P'. This follows because strong separability implies determinism and the local non-interference conditions agree with the lazy security condition for such processes.

§Corollary 6.4.9 Process P is strongly separable with respect to partition $\{H, L\}$ if and only if $\mathcal{L}_H(P)$ and $\mathcal{L}_L(P)$ are deterministic.

Corollary 6.4.10 If process P is strongly separable with respect to partition $\{H, L\}$ then $\mathsf{SLNI}_L(P)$ holds.

Proof Since P is strong separable with respect to partition $\{H, L\}$, by Corollary 6.4.9, $\mathcal{L}_H(P)$ and $\mathcal{L}_L(P)$ are deterministic. By Proposition 6.4.6, $P = \mathcal{L}_H(P) \mid\mid\mid \mathcal{L}_L(P)$. Hence, since the alphabets are disjoint, P is deterministic. Therefore, by Theorems 6.1.6 and 3.2.17, since $\mathcal{L}_H(P)$ is deterministic $\mathsf{SLNI}_L(P)$ holds. □

The equivalent version of Theorem 6.4.4 does not hold if SLNI is replaced by LNI[5]. The local non-interference version of the theorem fails in those cases where there is a state σ which has multiple transitions labelled with the same event $t \in H$ and the states these transitions lead to, $\sigma_1, \ldots, \sigma_n$, are not all equivalent. In these situations there is a nondeterminism present in $\sigma \setminus_H$; this is preserved after a t event by the nondeterminism arising from the multiple t labelled transitions.

When a process is interleaved with $\sigma \setminus_H$ it can take place before or after the nondeterminism has been resolved. When we interleave with the nondeterministic choice of the $\sigma_i \setminus_H$ states, the nondeterminism is always

[5]It does for LNI^{FD}.

resolved before the interleaved process has the opportunity to communicate. This is because the interleaving is done with the σ_i states and the nondeterministic choice of the resultant states is then taken. In other words, the local non-interference security condition can fail because

$$\begin{array}{ll}
& (P' \mid\mid\mid Q') \setminus_H \\
\approx_B & \hfill [\text{ Lemma 6.4.3 }] \\
& (P' \setminus_H) \mid\mid\mid (Q' \setminus_H) \\
\approx_B & \hfill [\ \mathsf{LNI}_L(P') \text{ holds }] \\
& (\sigma_1 \setminus_H \sqcap \ \ldots \sqcap \sigma_n \setminus_H) \mid\mid\mid (Q' \setminus_H)
\end{array}$$

is not necessarily equivalent to

$$(\sigma_1 \mid\mid\mid Q') \setminus_H \sqcap \ldots \sqcap (\sigma_n \mid\mid\mid Q') \setminus_H$$

This distinction is the different between the 'during' and 'after' types of nondeterministic choice discussed on page 87.

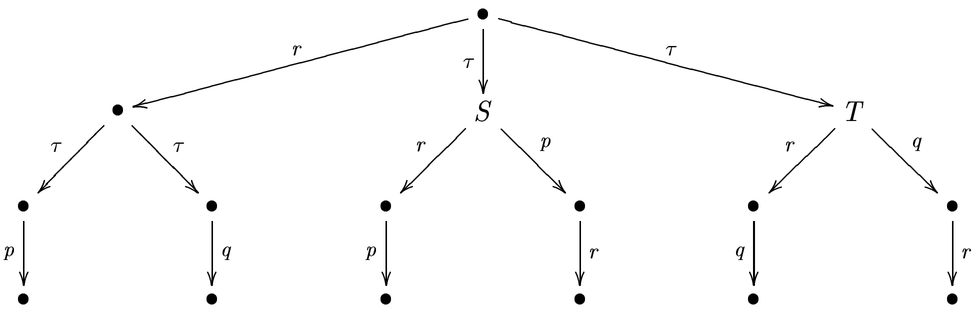
Example 6.4.11 Let $L = \{p, q, r\}$, $H = \{h\}$ and consider the processes

$$\begin{array}{rcl}
P_1 & = & p.0 \\
P_2 & = & q.0 \\
P & = & \tau.p.0 + \tau.q.0 + h.P_1 + h.P_2 \\
Q & = & r.0
\end{array}$$

$\mathsf{LNI}_L(P)$ holds since $P \setminus_H = \tau.p.0 + \tau.q.0 = \tau.P_1 \setminus_H + \tau.P_2 \setminus_H$ (see Example 5.1.4) and $\mathsf{LNI}_L(Q)$ holds since Q only communicates in L (Theorem 5.3.2). For $\mathsf{LNI}_L(P \mid\mid\mid Q)$ to hold we required

$$\tau.(P_1 \mid\mid\mid Q) \setminus_H + \tau.(P_2 \mid\mid\mid Q) \setminus_H \approx_B (P \mid\mid\mid Q) \setminus_H$$

Let $S = p.r.0 + r.p.0$ and $T = q.r.0 + r.q.0$. We have



$$
\begin{aligned}
(P \mid\mid\mid Q) \setminus_H &\approx_B P \setminus_H \mid\mid\mid Q \setminus_H \\
&\approx_B \tau.S + \tau.T + r.(\tau.p.0 + \tau.q.0)
\end{aligned}
$$

whereas

$$\tau.(P_1 \mid\mid\mid Q) \setminus_H + \tau.(P_2 \mid\mid\mid Q) \setminus_H = \tau.S + \tau.T$$

Hence $\mathsf{LNI}_L(P \mid\mid\mid Q)$ does not hold. The low-level user can distinguish between the states before and after an h event and can thus deduce when such an event has, or has not, occurred.

(End of Example)

Another case that needs to be considered is when the two processes being interleaved can communicate the same $t \in H$ labelled transition. For example, if $P \mid\mid\mid Q \xrightarrow{t} P' \mid\mid\mid Q$ and $P \mid\mid\mid Q \xrightarrow{t} P \mid\mid\mid Q'$, it is necessary to consider $(P' \setminus_H \mid\mid\mid Q \setminus_H) \sqcap (P \setminus_H \mid\mid\mid Q' \setminus_H)$. In general, if P never has a nondeterminism created by an H labelled event or the resulting states of such nondeterminism are equivalent, and if the same holds for Q, then $\mathsf{LNI}_L(P) \Rightarrow \mathsf{SLNI}_L(P)$ and no problem will occur. In other words, this case adds no new problems to the ones already covered since it is only an issue when either P or Q has multiple t labelled transitions.

The cases that cause the problems for local non-interference are when a nondeterminism that is visible to the low-level user is duplicated by the nondeterminism generated by the occurrence of duplicated $t \in H$ labelled transitions from a given state. An alternative approach to local non-interference, as discussed on page 96, is to rewrite the process P as $\mathcal{R}_H(P)$ (by replacing duplicate labelled transitions with a single transition followed by a nondeterministic choice) and then seeing whether the $\mathsf{SLNI}_L(\mathcal{R}_H(P))$ holds.

Theorem 6.4.12 $\mathsf{LNI}_L(P) \wedge \mathsf{LNI}_L(Q) \Rightarrow \mathsf{SLNI}_L(\mathcal{R}_H(P) \mid\mid\mid \mathcal{R}_H(Q))$.

Proof Follows from Theorem 5.2.11 and Theorem 6.4.4. □

Since the problem will exist whenever we have such high-level and low-level nondeterminisms interacting, it is interesting to look at which processes do not exhibit it. The problem is removed if neither process has such high-level nondeterminisms resolving low-level ones; this does not prevent processes that are nondeterministic and communicate in both H and L. In particular it will also be true when the individual processes communicate in only one of these alphabets. This is the requirement for separability and so the fact that local non-interference is not preserved by the interleaving of arbitrary secure processes is of lesser importance.

Theorem 6.4.13 If $\Sigma P = L$ and $\Sigma Q = H$ then $\mathsf{LNI}_L(P \mid\mid\mid Q)$ holds.

Proof For all $Q' \in \mathit{states}\,[\![Q]\!]$ if $Q' \xrightarrow{t} \sigma_1, \ldots, Q' \xrightarrow{t} \sigma_n$ we have $Q' \setminus_H$ and $\sigma_i \setminus_H$ are bisimilar to *Stop*. Hence, if $P' \mid\mid\mid Q' \in \mathit{states}\,[\![P \mid\mid\mid Q]\!]$, it is easy to show that $(P' \mid\mid\mid Q') \setminus_H \approx_B (P' \mid\mid\mid \sigma_1) \setminus_H \sqcap \ldots \sqcap \; (P' \mid\mid\mid \sigma_n) \setminus_H$ as required. $\Box$

6.5 Implementation

The automatic verification of the security conditions is necessary because the state space rapidly becomes too large to handle manually, even for small systems. Furthermore, automatic tools help in the developmental process, by allowing a large number of systems to be tested quickly and accurately. Tools exist that can be used to check several of the existing security conditions. CSP has been adapted for automatic tools [Sca98]; the FDR tool [FSEL] provides a way to checking determinism directly and so test the lazy security condition. CCS processes are served by the *Concurrency Workbench* [CPS93] which in turn has been adapted to the *Security Checker* for BNDC [FG95, FG97].

Strong local non-interference can be tested be looking at every pair of states connected by an H labelled transition and testing whether they are bisimilar. This is not the best way to do it since it requires many bisimilarity tests and there is potentially a great deal of overlap in the calculations being performed. Instead it is better to calculate the maximal autobisimulation with respect to H and then check the necessary pairs of states. The autobisimulation effectively 'colours' every state and two states are bisimilar if and only if they share the same colour. Hence we can quickly test whether the condition holds for a given pair of states.

To verify LNI we use Theorem 5.2.11 and apply the strong local non-interference algorithm to the rewritten transition system.

Computing the maximal autobisimulation of the transition system is done in two stages. By Theorem 2.2.10 we can calculate the maximal weak autobisimulation of $\mathcal{A}_1 = \langle S_1, T_1, \alpha_1, \beta_1, \lambda_1 \rangle$ by creating the transition system $\mathcal{A}_2$, where $\stackrel{\hat{\mu}}{\Longrightarrow}$ paths are replaced by $\stackrel{\mu}{\longrightarrow}$ transitions, and then calculating the maximal strong autobisimulation. Now $\stackrel{\hat{\tau}}{\Longrightarrow}$ means $(\stackrel{\tau}{\longrightarrow})^*$, hence to generate the τ paths we require the reflexive, transitive closure of the τ transitions[6]. It is then simple to calculate the relational product $(\stackrel{\tau}{\longrightarrow})^* \stackrel{\mu}{\longrightarrow} (\stackrel{\tau}{\longrightarrow})^*$; i.e. $\stackrel{\hat{\mu}}{\Longrightarrow}$ where $\mu \in L$.

The maximal strong autobisimulation can be calculated by a partitioning algorithm. The states that are bisimilar to each other are kept in the same partition. At each iteration these partitions, which may contain states that are not bisimilar, are refined into further partitions as we make more distinctions between its members. Ultimately every partition will only contain bisimilar states.

Consider the operator Ψ from $S_1 \times S_1$ to itself where

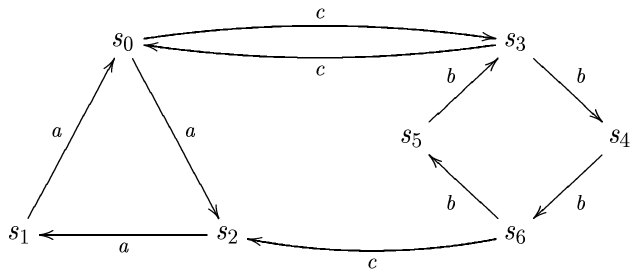
$$\begin{aligned}\Psi(R) = \{(P_1, P_2) \quad | \quad & \forall\, t \in \Sigma \setminus H \bullet \\ & \forall\, \sigma \bullet (P_1 \stackrel{t}{\longrightarrow} \sigma \Rightarrow \exists\, \sigma' \bullet (P_2 \stackrel{t}{\longrightarrow} \sigma' \wedge (\sigma, \sigma') \in R)) \ \wedge \\ & \forall\, \sigma \bullet (P_2 \stackrel{t}{\longrightarrow} \sigma \Rightarrow \exists\, \sigma' \bullet (P_1 \stackrel{t}{\longrightarrow} \sigma' \wedge (\sigma, \sigma') \in R))\}\end{aligned}$$

It follows from this definition that Ψ is monotonic, under inclusion, on the complete lattice of $S_1 \times S_1$ and R is a bisimulation if and only if $R \subseteq \Psi(R)$. Hence, $\Psi(\bigcap R_i) = \bigcap \Psi(R_i)$ with the maximal fixpoint $R_\Psi = \bigcap \Psi^i(S_1 \times S_1)$. In other words we compute the limit of the sequence (R_i) where $R_0 = S_1 \times S_1$ and $R_{i+1} = \Psi(R_i)$. This has a polynomial time complexity[7]; a crude upper bound is $O(\mid S_1 \mid . \mid T_2 \mid^2)$ since there are most $\mid S_1 \mid$ iterations where for every transition we need to check for a corresponding transition.

[6] Standard algorithms for this can be found, for example, in [Ski98].

[7] This method can be improved upon. Efficient algorithms for strong bisimulation are discussed in [Fer89] and [PT87], and for weak bisimulation directly in [KS90].

Example 6.5.1 Consider the transition system:



Initially all the states are in the same equivalence class. The first application of Ψ groups together the states with identically labelled transitions. So $\Psi(S_1 \times S_1)$ contains four equivalence classes: $\{s_0\}, \{s_1, s_2\}, \{s_3, s_6\}$ and $\{s_5, s_4\}$. The second application splits up $\{s_1, s_2\}$ and $\{s_3, s_6\}$, and the third application splits up $\{s_5, s_4\}$.

(End of Example)

Once we have calculated the maximal autobisimulation R_Ψ, strong local bisimulation can be checked by looking at all the pairs of states, in the original transition system, that have an H labelled transition between them. If $\sigma \xrightarrow{t} \sigma'$ where $t \in H$, we require $(\sigma, \sigma') \in R_\Psi$.

Chapter 7

Conclusions

For low-deterministic processes the idea of security, or lack of information flow, seems to be well understood. There are various reasons for believing that, when the absence of high-level communications implies the process behaves deterministically, we can correctly identify secure and insecure processes. Firstly the conditions we have for different models are equivalent for deterministic processes, with the different untimed models of concurrency agreeing for such processes. Secondly different security formulations, including those based on deterministic processes and some (such as invariance, BNDC and local non-interference) which deal with nondeterministic processes, are equivalent for low-deterministic processes. Finally experience with the determinism based conditions and a wide variety of processes has produced the expected results.

Nondeterministic processes appear to present many more difficulties for specifying security properties. There are a large number of conditions in the literature and a great variance in what constitutes both secure and insecure processes. Part of this problem is the difficult in modelling nondeterminism; there are several different ways of viewing nondeterminism and the processes that we class as secure will change depending on which view we take. It is also much harder to encapsulate what we really want to mean by 'secure'. The idea of an H labelled transition taking us to a state we may have reached without it, while still necessary, is not sufficient.

Studying processes as transition systems permits us to deal with the individual possibilities / transitions. Process algebras such as CSP can abstract away too many of the details of nondeterminism; it is this under-specification of nondeterminism that is usually so desirable and analysing potential information flow is one of the few areas where this appears to be the case [RRF99]. This makes it difficult, if not impossible, to directly determine facts that seem important in determining whether a process is secure, as in the $Chaos \sqcap LEAK$ examples in Section 3.3. This suggests that the standard denotational models of CSP are not sufficient when looking at information flow in nondeterministic systems and it is necessary to consider processes using operational models such as transition systems.

Existing nondeterministic conditions appear to classify some insecure processes as secure. The conditions fall into two categories: those that agree with the determinism based conditions and those that do not. The non-deducibility, non-deducibility on strategies, (strong) nondeterministic non-interference and NDC conditions all fail to correctly identify the process $P = l \rightarrow P \,\square\, h \rightarrow h \rightarrow P$ as insecure, even though they correctly identify $P = h \rightarrow l \rightarrow Stop$. This is not due to an eager interpretation of high-level events, otherwise both processes would be incorrectly identified, but subtler flaws in the conditions. The remaining conditions, which have been shown to agree with the lazy security condition, fail for alternate reasons.

We have introduced the new local non-interference conditions and shown that they correspond with our expectations for low-deterministic processes. For nondeterministic processes the conditions are strict, relying on a possibilistic interpretation of nondeterminism and preventing the high-level user from resolving, or re-resolving, any internal choices. In particular this requires an operational, transition system description of a process, rather than a behavioural one, with a literal interpretation that permits no automatic refinement.

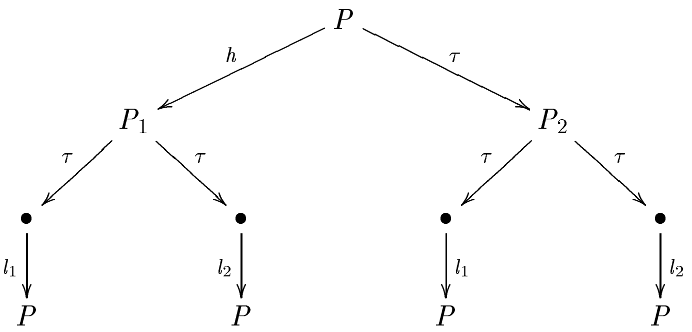
7.1 Refinement Paradox

The security properties for nondeterministic systems suffer from the *refinement paradox*. There are some process that are considered secure, which have refinements that are insecure. This seems undesirable, and counter-intuitive, since a refinement of a process can be thought of as one of the ways the process can behave. [It is perfectly reasonable to expect insecure processes to be refined by secure ones. The process $P \sqcap Q$, where P is a secure process and Q is an insecure one, can be refined by P.]

The determinism based conditions avoid the refinement paradox since they are closed under refinement. This follows since deterministic processes are maximal under refinement. For non-divergent, deterministic processes we have shown that the local non-interference conditions agree with the lazy security condition, so we know the refinement paradox does not affect such processes.

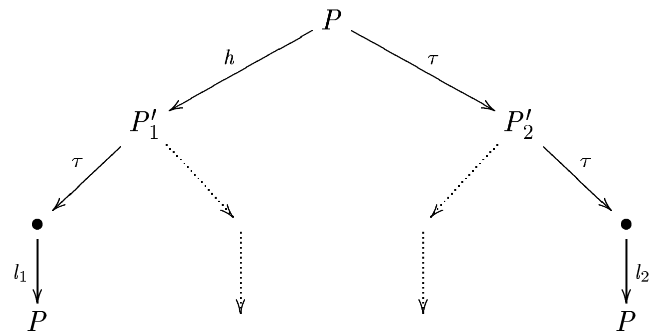
We do not need a complicated process to exhibit the refinement paradox. Consider the process

$$P = (h \rightarrow (l_1 \rightarrow P \sqcap l_2 \rightarrow P)) \triangleright (l_1 \rightarrow P \sqcap l_2 \rightarrow P)$$



This is a process it seems reasonable to class as secure since, independent of the high-level user's behaviour, it nondeterministically chooses an l_1 event or l_2 event. Whenever the high-level user communicates an h event the system is taken into a state that is equivalent to the one that would have been reached had the event not been communicated. The occurrence of an l_1 leading from state P_1 is indistinguishable to an l_1 leading from state P_2; similarly for l_2. This process satisfies both BNDC and SLNI.

This process can be refined, using the standard CSP semantic models, by $P' = h.\tau.l_2.P' + \tau.\tau.l_1.P'$, by deciding how the internal choices at P_1 and P_2 are resolved.



This process is clearly insecure. The low-level user can deduce an h event has occurred if an l_1 event can be communicated; similarly an h event cannot have occurred if an l_2 event is available. This process satisfies neither BNDC or SLNI.

One cause of information flow in nondeterministic process, highlighted in Chapters 4 and 5, is when the high-level user resolves some nondeterminism in the system. Some forms of refinement also resolve nondeterminism; this can remove states equivalent to states that other events, possibly under the control of the high-level user, can still place the system in. This can cause some (potentially) secure processes to have refinements that are insecure.

When we refined P to P' two nondeterministic choices were resolved: the one at P_1 and the one at P_2. The security of P relies on the fact that P_1 and P_2 are equivalent, but this is no longer true of the relative states P'_1 and P'_2 of P'. This refinement problem does not arise in deterministic systems, since there are no observable nondeterminisms to resolve.

If we want to retain CSP refinement we cannot prevent this problem without seriously restricting the nondeterministic processes we count as secure. It would be necessary, for example, to have a condition that ensures for all states S where an H labelled event is available, all refinements of the low-level system view after the high-level event are equivalent to all refinements of the low-level system view without the event. This would limit any low-level nondeterminism to before the high-level user is able to act. This is because any nondeterminism once H labelled events can occur, might be refined differently depending on whether or not these actions do occur.

Under a possibilistic interpretation of nondeterminism this type of refinement is not permitted because it changes the available nondeterministic choices. For a security condition to hold after a refinement we need to ensure that indistinguishable states remain so after refinement. In other words any refinement of nondeterministic choice would have to be the same over all indistinguishable choices. Such a refinement exists for the view of nondeterminism presented by Lowe in [Low99]. If $P \sqsubseteq Q$ then, for every way the nondeterminism can be resolved in Q, it must be possible to resolve the nondeterminism in P so the two processes behave the same. This would prevent P be refined by P' above if the two nondeterministic choices had the same labelling.

7.2 Assumptions

The local non-interference conditions make various assumptions, or rather choices about how the system is modelled and observed. The possibilistic nature of nondeterminism and the literal interpretation of transition systems are discussed in the preceding section. The assumptions about the abilities of the users is discussed in Chapter 5 and is based on the ideals given in the introduction; the low-level user should be given as much opportunity to detect information flow and the high-level user should have as much opportunity to cause it.

These assumptions err on the side of caution. It seems far more acceptable to be rejecting secure processes as insecure, than accepting insecure processes as secure. The first case makes the system designer more cautious, by potentially making it harder to design a secure process. The second case means we might fail in our aim of identifying undesirable information flow.

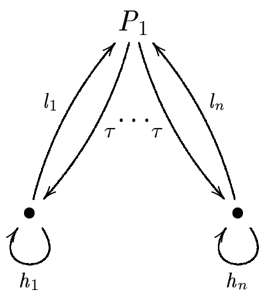
The second motivating property for local non-interference, in Section 5.1, was that the low-level user should not be able to deduce the behaviour of the high-level user, but that the results of events outside the high-level user's control, even if they affect which H labelled events are available, can be deduced. Part of the motivation for this is that we require an asymmetric property, so the low-level user should be able to influence the available H labelled events. The other reason, implicitly given by this and other properties, is that information flow can only be caused by the low-level user deducing information about high-level behaviour, or the high-level user influencing low-level behaviours.

The availability of H labelled events due to internal choices is not classed as an information flow, even when the low-level user can deduce which high-level behaviours have become available. In other words, the low-level user is allowed to make deductions on information that has been passed from the system to the high-level user. What cannot be deduced in a secure system is that the high-level user has received this information or what is subsequently done with it.

When the system makes a nondeterministic choice, then makes the result available to both the high-level user and the low-level user, there is no way we can tell whether the information was meant to be secure. This is very different to resolutions based on the high-level user's behaviour because such behaviour is supposed to be secret.

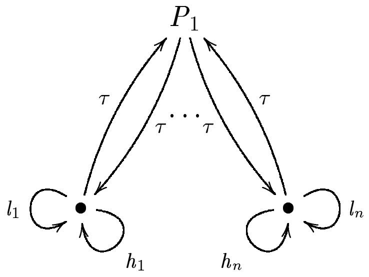
Example 7.2.1 Let $L = \{l_1, \ldots, l_n\}$ and $H = \{h_1, \ldots, h_n\}$. Consider the process

$$\begin{aligned} P_1 = \quad & \mu X.(l_1 \to P_1 \;\Box\; h_1 \to X) \\ \sqcap \quad & \ldots \\ \sqcap \quad & \mu X.(l_n \to P_1 \;\Box\; h_n \to X) \end{aligned}$$



P_1 is a process that make an internal choice, gives the high-level user the opportunity the see how the choice was resolved, and guarantees the low-level user can see the resolution of the nondeterminism before another internal choice is made. P_1 satisfies the local non-interference conditions since the behaviour of the high-level user can have no effect on the low-level user. The only conditions that P_1 does not satisfy are non-deducibility and non-deducibility on strategies.

We can construct a similar process which also satisfies the non-deducibility conditions. Consider the process P_2 where the recursions are replaced by $\mu X.(l_i.X + h_i.X + \tau.P_2)$



P_2 no longer guarantees the low-level user can see how the choice was resolved, although any observations by the low-level user are a subset of the internal choices. Any sequence of resolutions that are observed by either user might also be observed by the alternate user.

(End of Example)

We can construct a process, as in Example 7.2.1, where the system nondeterministically generates a high-level key and this information can be deduced by the low-level user. The local non-interference conditions allow this since we are concerned with what the high-level user does rather than what the high-level user is told. Nevertheless it is not possible for the high-level user to use this information within a process that satisfies the conditions. The use of a key, for example to XOR with some confidential bit stream, will fail; the result of the computation would have to be the resolution of a nondeterminism and this would cause the process to be insecure.

Bibliography

[All91] P. G. Allen. "A Comparison of Non-Interference and Non-Deducibility using CSP", *Proc. 1991 IEEE Computer Security Workshop*, 43–54.

[Arn94] A. Arnold. "Finite Transition Systems", Prentice Hall (1994).

[CPS93] R. Cleaveland, J. Parrow and B. Steffen. "The Concurrency Workbench: A Semantics-Based Tool for the Verification of Concurernt Systems", *ACM Transactions on Programming Languages and Systems* **15:1** (1993), 36–72.

[Fei80] R. Feiertag. "A Technique for Proving Specifications are Multilevel Secure", *Technical Report, SRI Report CSL-109* (1980).

[Fer89] J. Fernandez. "An Implementation of an Efficient Algorithm for Bisimulation Equivalence", *Science of Computer Programming* **13** (1989/90), 219–236.

[FG93a] R. Focardi, R. Gorrieri. "A Classification of Security Properties (extended abstract)", *Technical Report UBLCS-93-21* (October 1993).

[FG93b] R. Focardi, R. Gorrieri. "An Information Flow Security Property for CCS", *Proceedings of Second North American Process Algebra Workshop (NAPAW'93)* (August 1993).

[FG94] R. Focardi, R. Gorrieri. "A Classification of Security Properties", *Journal of Computer Security* **3:1** (1994/5) 5–33.

[FG95] R. Focardi, R. Gorrieri. "The Security Checker: A Semantics-based Tool for the Verification of Security Properties", *Proceedings of the Eigth IEEE Computer Security Foundations Workshop* (June 1995), 60–69.

[FG97] R.Focardi, R. Gorrieri. "The Compositional Security Checker: A Tool for the Verification of Information Flow Security Properties", *IEEE Transactions on Software Engineering* **23:9** (September 1997).

[FLR77] R. J. Feiertag, K. N. Levitt, L. Robinson. "Proving Multilevel Security of a System Design", *Proceedings, Sixth ACM Symposium on Operating Systems Principles* (1977), 57–56.

[Foc96] R. Focardi. "Comparing Two Information Flow Security Properties", *Proceedings of 9th IEEE Computer Security Foundations Workshop (CSFW'96)* (June 1996) 116–122.

[For97] R. Forster. "Non-interference Properties for Nondeterministic Systems" *Dissertation for transfer to DPhil status, Oxford University Computing Laboratory* (1997).

[FSEL] Formal Systems (Europe) Limited. "Failures-Divergences Refinement: User Manual and Tutorial", *Available by FTP from Oxford University.*

[GM82] J. A. Goguen, J. Meseguer. "Security Policies and Security Models", *IEEE Symposium on Security and Privacy* (1982), 11–20.

[GM84] J. A. Goguen, J. Meseguer. "Unwinding and Inference Control", *IEEE Symposium on Security and Privacy* (1984), 75–86.

[Gra92] J. Graham-Cumming. "The Formal Development of Secure Systems", *DPhil, Oxford University* (1992).

[Hoa85] C. A. R. Hoare. "Communicating Sequential Processes", Prentice Hall (1985).

[Jac90a] J. Jacob. "Separability and the Detection of Hidden Channels", *Information Processing Letters* **34** (1990), 27–29.

[Jac90b] J. Jacob. “Specifying Security Properties”, *Developments in Concurrency and Communication* (C. A. R. Hoare, editor) (1990), Addison Wesley, 221–237.

[KS90] P. Kanellakis, S. A. Smolka. “CCS Expressions, Finite State Processes, and Three Problems of Equivalence”, *Information and Computation* **86** (1990), 43–68.

[KW91] P. A. Karger and J. C. Wray. “Storage Channels in Disk Arm Optimization”, *Proceedings of the 1991 Symposium on Research in Security and Privacy* (1991), 52–61.

[Low99] G. Lowe. “Defining Information Flow”, *Technical Report No. 1999/3, Department of Mathematics & Computer Science, Leicester University* (1999).

[McC87] D. McCullough. “Specifications for Multi-Level Security and a Hook-Up Property”, *1987 IEEE Symposium on Security and Privacy* (1987), 161–166.

[McC88] D. McCullough. “Noninterference and the Composibility of Security Properties”, *1988 IEEE Symposium on Security and Privacy* (1988).

[McC90] D. McCullough. “A Hookup Theorem for Multilevel Security”, *IEEE Transactions of Software Engineering* **16:6** (June 1990), 563–568.

[McL90] J. McLean. “Security Models and Information Flow”, *1990 IEEE Symposium on Security and Privacy* (1990), 180–187.

[Mill87] J. K. Millen. “Covert Channel Capacity”, *1987 IEEE Symposium on Security and Privacy* (1987), 60–66.

[Mill90] J. .K. Millen. “Hookup Security for Synchronous Machines”, *1990 IEEE Computer Security Foundations Workshop* (1990), 84–90.

[Miln80] R. Milner. “A Calculus of Communicating Systems”, *LNCS 92*, Springer-Verlag (1980).

[Miln89] R. Milner. “Communication and Concurrency”, Prentice Hall (1989).

[Par81] D. Park. "Concurrency and Automata on Infinite Sequences", *5th GI Conference on Theoretical Computer Science, LNCS 104* Springer-Verlag (1981), 167–183.

[Pin95] S. Pinsky. "Absorbing Covers and Intransitive Non-interference", *1995 IEEE Symposium on Security and Privacy* (1995), 102–113.

[PT87] R. Paige, R. E. Tarjan. "Three Partition Refinement Algorithms", *SIAM J. Comput.* **16:6** (December 1987), 973–989.

[RG99] A. W. Roscoe, M. H. Goldsmith. "What is Intransitive Noninterference?", *12th IEEE Computer Security Foundation Workshop* (28–30 June 1999), 228-238.

[RGG95] A. W. Roscoe, P. H. B. Gardiner, M. H. Goldsmith, J. R. Hulance, D. M. Jackson, J. B. Scattergood. "Hierarchical Compression for Model-Checking CSP *or* How to Check 10^{20} Dining Philosophers for Deadlock", *TACAS (Denmark)* (May 1995), 187–200.

[Ros94] A. W. Roscoe. "Model-checking CSP", *A Classical Mind: Essays in Honour of C.A.R. Hoare* (A. W. Roscoe, editor), Prentice Hall (1994), 353–378.

[Ros95a] A. W. Roscoe. "CSP and Determinism in Security Modelling", *1995 IEEE Symposium on Security and Privacy* (1995), 114–127.

[Ros98] A. W. Roscoe. "The Theory and Practice of Concurrency", Prentice Hall (1998).

[RRF99] A. W. Roscoe, G. M. Reed and R. Forster. "The Successes and Failures of Behavioural Models", *Symposium in Celebration of the work of C. A. R. Hoare* (1999).

[RS99] P. Y. A. Ryan, S. A. Schneider. "Process Algebra and Non-interference", *12th IEEE Computer Security Foundation Workshop* (28–30 June 1999), 214–227.

[RW95] A. W. Roscoe, L. Wulf. "Composing and Decomposing Systems under Security Properties", *Proc. 8th IEEE Computer Security Foundations Workshop* (1995), 9–15.

[RWW94] A. W. Roscoe, J. C. P. Woodcock, L. Wulf. "Non-interference through Determinism" *ESORICS 94, LNCS 875* Springer (1994), 33–53.

[Rus81a] J. M. Rushby. "Proof of Separability: A Verification Technique for a Class of Security Kernels", *Techincal Report, Computing Laboratory, University of Newcastle upon Tyne* (1981).

[Rus81b] J. M. Rushby. "The Design and Verification of Secure Systems", *ACM Operating Systems Rev.* **15:5**, 12–21.

[Rus81c] J. M. Rushby. "Verification of Secure Systems", *Internal Report SSM/9, Computing Laboratory, Univeristy of Newcastle upon Tyne* (1981).

[Rya91] P. Y. A. Ryan. "A CSP Formulation of Non-Interference and Unwinding", *Cipher* (Winter 1991), 19–30.

[Sca98] B. Scattergood. "Tools for CSP and Timed CSP", *DPhil, Oxford University* (1998).

[Sch77] M. Schaefer, B. Gold, R. Linde and J. Scheid. "Program confinement in KVM/370", *Proceedings of the 1977 ACM Annual Conference* (October 1977), 303–410.

[Ski98] S. Skiena. "The Algorithm Design Manual", Springer (1998).

[Sut86] D. Sutherland. "A Model of Information", *Proceedings of the 9th National Computer Security Conference* (September 1986), 175–183.

[TG88] C. Tsai and V. D. Gligor. "A Bandwidth Computation Model for Covert Storage Channels and its Applications", *1988 IEEE Symposium on Security and Privacy* (1988), 108–121.

[WJ90] J. T. Wittbold, D. M. Johnson. "Information Flow in Nondeterministic Systems", *Proceedings of the 1990 Symposium on Research in Security and Privacy* (1990), 144–161.

[Wra91] J. C. Wray. "An Analysis of Covert Timing Channels", *Proceedings of the 1991 Symposium on Research in Security and Privacy* (1991), 2–7.

[Wul97] L. Wulf. "Interaction and Security in Distributed Computing", *DPhil, Oxford University* (1997).

[ZL97] A. Zakinthinos and E. S. Lee. "A General Theory of Security Properties", *Proceedings of the 1997 Symposium on Research in Security and Privacy* (1997).