Isabella Mastroeni

Abstract Non-Interference - An Abstract Interpretation-based approach to Secure Information Flow

Ph.D. Thesis

31 Marzo 2005

Università degli Studi di Verona
Dipartimento di Informatica

Advisor:
prof. Roberto Giacobazzi

Series N°: **TD-02-05**

Università di Verona
Dipartimento di Informatica
Strada le Grazie 15, 37134 Verona
Italy

Summary

In this thesis, we show how the framework of abstract interpretation can be used for certifying the security degree of programs. In particular, the idea consists in observing that, if we model attackers as abstract domains, then by transforming these domains, we can manipulate attackers, characterizing which is the most powerful attacker for which a program is secure. Therefore, the central notions used in this thesis are *abstract domain transformers* and *secure information flows* (also called non-interference). First of all, we introduce an algebra defining a framework where we can design, classify and compare abstract domain transformers. In particular, we show that the standard Cousot and Cousot theory of abstract interpretation, based on the so called adjoint-framework of Galois connections, can be directly applied to reason about abstract domain transformers, yet providing formal methodologies for the systematic design of these transformers. The key point is that most domain transformers can be viewed as suitable problems of achieving precision with respect to some given semantic feature of the programming language we want to analyze. This is exactly the philosophy that lead us to the design of the transformer providing the most powerful harmless attacker that cannot violate non-interference, i.e., the most powerful observer that is not able to disclose any confidential information. Indeed, the main subject of this thesis is the definition of the notion of *abstract non-interference*, obtained by parameterizing standard non-interference relatively to what an attacker is able to observe. This notion is what we need for characterizing the secrecy degree of programs in the lattice of abstract interpretations, by deriving the most powerful harmless attacker for any program. The definition of abstract non-interference is based on the semantics of programs. This means that we can enrich this notion simply by enriching the considered semantics. Note that, abstract non-interference is a weakening of the standard notion of non-interference, but it is not the first work with this aim, for this reason we compare abstract non-interference with two of the most related works: the PER model [106] and robust declassification [118].
In order to make the abstract non-interference certification more practical, we introduce a compositional proof system whose aim is to certify abstract non-

interference in programming languages, inductively on the syntax of the languages, following an *a la* Hoare deduction of secrecy. At this point, we show that abstract non-interference can be formalized as a completeness problem, in the standard framework of abstract interpretation. This allows us to characterize the derivation of the most concrete public observer, i.e., the harmless attacker, and the derivation of the most abstract private observable of the program (related to declassification) as adjoint domain transformers. This adjoint relation formalize the intuitive dualism between these two approaches introduced for weakening non-interference.
We conclude this thesis by showing how we can further enrich abstract non-interference by adding the observation of time, and by generalizing abstract non-interference in order to model the confinement problem also for computational systems that are not programming languages. Indeed, we introduce the notion of *timed abstract non-interference*, providing the appropriate setting for studying how properties, of private data, interfere during the execution of programs, with properties of the elapsed time. In this way, we obtain a notion of abstract non-interference that avoids timing channels, namely information transmissions due to the capability of attackers of observing time. Finally, we prove that abstract non-interference can be generalized in order to cope with many well known models of secrecy in sequential, concurrent and real-time systems and languages. This is achieved by factoring abstractions in order to identify sub-abstractions modeling the different properties of the system on which the notions of non-interference are based. In particular, an abstraction decides the model of the system used for defining non-interference, e.g., denotational semantics. A further abstraction decides which is the aspect of computation that is observable, e.g., the computations where private actions are avoided. Finally, the last abstraction considered characterizes which properties of the computation the attacker can observe. These three abstractions are composed for obtaining generalized abstract non-interference, and depending on how we choose them, we decide the notion of non-interference that we want to enforce.

Acknowledgments

The first person that I have to thank is my advisor Roberto Giacobazzi, who, in these years, has been a friend, but above all a precious guide, teaching me how to be independent, developing my ideas, and making me grow up as part of a group and as scientific researcher.
I have also to thank very much Alessandro Finezzo whose friendly help as system administrator of my pc has been really precious. I cannot forget to thank Maurizio Atzori, Debora Botturi, Giuseppe Di Modica, Andrea Girardi, Antoine Miné, Enrico Oliosi, Elodie-Jane Sims, Andrea Tessari for their sincere friendship, but also all my friends and colleagues who made more pleasant to spend my days in the department, among all Alessio, Arnaud, Damiano, Fausto, Giovanni, Giuditta, Luca, Mila, Rosalba, Roberto (Segala), Samir.
It would be impossible not to quote my PhD thesis referees Patrick Cousot and Chris Hankin, but also Sebastian Hunt, Massimo Merro, Francesco Ranzato and Dave Schmidt for their precious advices and comments about my studies.
Clearly, most of my greatest thanks go to my family and in particular to my husband Federico, who tolerated my ups and downs while I was working on this thesis.

Contents

Preface

The present thesis is composed by 11 chapters. Each chapter is provided with a brief introduction explaining its contents, and all the chapters about original work end with a discussion about the problems addressed in the chapter and the relations with existing works. Much of the contents of this thesis have been published. In particular, the Chapters 6,7,8,9 and 10 and the compression method provided in Chapter 3 have been developed together with Roberto Giacobazzi. While Chapter 3 is based on works developed also together with Francesco Ranzato.
Given the number of joint works, I will opt for the academic "we" instead of "I". Finally, I would like to add some comments on the contents of this thesis. In particular, I think that the presence of Chapter 3, with a lot of results that are not used in the rest of the thesis, needs some explanations. Indeed, the work for my PhD thesis started studying an algebra for abstract domain transformers. This work lead us to the theoretical results explained in the chapter, but these results needed a field of application. This has been one of the reasons that convinced us to work on an old idea: to model security policies by abstract interpretation. So, we started working on security, and in particular on non-interference in language-based security. The definition of abstract non-interference has been the first result of these studies, and represented for us, in the beginning, a very good application for our algebra since it is based on an abstract domain transformer. But our work has continued in this direction, and while on one side, abstract non-interference well fit in the algebra of domain transformers, since it models two well known problems in language-based security as adjoint transformers of abstract domains, on the other side it showed us new fields of research and new ideas, new questions have continued to arise in our mind, leading us to obtain the amount of work presented in this thesis and with central subject abstract non-interference. For these reasons the study of the algebra for abstract domain transformers is not the main subject of this thesis. But these are also the reasons that convinced us to present anyway the (unfortunately incomplete) algebra among the results of the thesis. Therefore, a reader interested only in understanding abstract non-interference and the domain transformers defined for characterizing attackers and

observables, is not obliged to read the whole chapter. In particular, he/she can read Chapter 3 only until page 47. All the other results presented, concern the study of the algebra for abstract domain transformers and are not directly used in the development of abstract non-interference.

Verona, March 31, 2005

1

Introduction

Fatti non foste a viver come bruti, ma per seguir virtute e canoscenza.
Dante Alighieri

Suppose that some source S sends you a program, whose task is that of improving your financial investments. Suppose that you store all the information about your investments on a data base on your personal computer and that this software is free, under the condition that it can automatically send a log-file, containing a summary of the usage you made of the software, to the developers of the program. You could wonder if this program is secure [106]. The problem is that this program can access to private information about financial investment, and can send (public) information to S. You could wonder how one can be sure that S is not catching private information (about financial investments) through the observation of public information (log files). This is a clear example where it is necessary to check if a program has only *secure information flows* of information. This is a typical problem in language-based security and it is often called *non-interference*. Non-interference, previously referred as confidential problem or strong dependency, requires that no information about confidential data can be disclosed by observing public information, while data are processed by programs.

1.1 Non-interference in language-based security

In the last decades, an important task of language based security is to protect confidentiality of data manipulated by computational systems. Namely, it is important to guarantee that no information, about confidential/private data, can be caught by an external viewer. In many fields, where protection of confidentiality is a critical problem, the standard way used to protect private data is access control: special privileges are required in order to read confidential data. Unfortunately, these methods allow to restrict accesses to data but cannot control propagation

of information. Namely, once the information is released from its container, it can be improperly transmitted without any further control. This means that the security mechanisms, such as signature, verification, and antivirus scanning, do not provide assurance that confidentiality is maintained during the whole execution of the checked program. This implies that, to ensure that confidentiality policies are satisfied, it becomes necessary to analyze how information flows within the executed program. In particular, if a user wishes to keep some data confidential, he might state a policy stipulating that no data visible to other users is affected by modifying confidential information. This policy allows programs to manipulate and modify private data, as long as visible outputs of those programs do not reveal information about these data. A policy of this sort is called *non-interference* policy [68], since it states that confidential data may *not interfere* with public data. Non-interference is also referred as *secrecy* [111], since confidential data are considered *private*, while all other data are public [39]. The difficulty of preventing a program P from leaking private information depends greatly on what kind of observations of P are possible [109]. If we can make *external observations* of P's running time, memory usage, and so on, then preventing leaks becomes very difficult. For example, P could modulate its running time in order to encode the private information. Furthermore, these modulations might depend on low level implementation details, such as caching behaviours. But this means that it is insufficient to prove confinement with respect to an abstract semantics, every implementation detail, that affects running time, must be addressed in the proof of confinement. If, instead, we can only make *internal observations* of P's behaviour, the confinement problem become more tractable [109]. Internal observations include the values of program variables, and everything is observable internally, e.g. time in real-time systems.

In order to understand how this problem can be formalized, we have to go back to the seminal paper [80], where the notion of *confinement problem* is introduced. Consider a *customer* program and a *service* (host) program, the customer would like to ensure that the service cannot access (read or modify) any of his data, except those information to which he explicitly grants access (said *public*). In other words, the confinement problem consists in *preventing the results of the computation from leaking even partial information about confidential inputs*. Clearly, if the public data depends, in some way, on the private ones, then confinement becomes a problem. This strict relation between the confinement problem and the dependencies among data allows to describe the confinement problem as a problem of *non-interference* [68] by using the notion of *strong dependency* introduced in [19]. In the latter, the transmission of information is defined by saying that *information is transmitted over a channel when variety is conveyed from the source to the destination*. Clearly, if we substitute source with private and destination with public, then we obtain the definition of insecure information flow. More formally speaking, Cohen in [19] says that information can be transmitted from a to b during the execution of a system S, if by suitably varying the initial value of a (exploring the

variety in a), the resulting value in b after S's execution will also vary (showing the variety is conveyed to b). The absence of strong dependency has been interpreted as non-interference in [68], where non-interference is defined as:

> *"One group of users [...] is noninterfering with another group of users if what the first group does [...] has no effect on what the second group of users can see".*

Therefore, we have that security, defined as presence of only secure information flows, is non-interference, which is absence of strong dependencies. These definitions are general and can be applied to different kind of computational systems. In general, the notion of non-interference is used to stipulate policies of non-interference whenever a user wishes to keep some data confidential.

1.2 The problem: Weakening non-interference

The limitation of the notion of non-interference described so far is that it is an extremely restrictive policy. Indeed, non-interference policies require that *any* change upon confidential data has not to be revealed through the observation of public data. There are at least two problems with this approach. On one side, many real systems are intended to leak some kind of information. On the other side, even if a system satisfies non-interference, some kind of tests could reject it as insecure. These observations address the problem of *weakening* the notion of non-interference both characterizing the information that *is allowed* to flow, and considering *weaker* attackers that cannot observe any property of public data.

Characterizing released information.

Real systems often intentionally leak confidential information, therefore it seems sensible to try to measure this leakage as best as possible. The first work on this direction is [19], where the notion of selective dependency is introduced, which consists in a weaker notion of dependency, and therefore of non-interference, that identifies what flows during the execution of programs. More recently, in literature we can find other works that attack this problem from different points of view.

In [17], Shannon's information theory is used to quantify the amount of information a program may leak and to analyze the way in which this depends on the distribution of inputs. In particular, the authors are interested in analysing how much an attacker may learn (about confidential information) by observing the input/output behaviour of a program. The basic idea is the all information in the output of a deterministic program has to come from the input, and what it is not provided by the low input has to be provided by the high input. Therefore, this work wants to investigate how much of the information carried by the high inputs to a program can be learned by observing of the low outputs, assuming that the low inputs are known.

Shannon's information theory is not the only approach, existing in literature, for quantifying information flow. Indeed in [84] the capacity of covert channels, i.e., the *information flow quantity*, is measured in terms of the number of high level behaviours that can be accurately distinguished from the low level point of view. The idea is that if there are N such distinguishable behaviours, then the high level user can use the system to encode an arbitrary number in the range $0, \ldots, N-1$ to send it to the low level user, in other words $\log_2 N$ bits of information are passed.

In literature, there exists another important, more qualitative, approach whose aim is to discover which is the information that flows in order to *declassify* it for guaranteeing non-interference. *Declassifying* information means downgrading the sensitivity of data in order to accommodate with (intentional) information leakage[1]. *Robust declassification* has been introduced in [118] as a systematic method to drive declassification by characterizing what information flows from confidential to public variables. In particular, the observational attacker's capability is modeled by using equivalence relations, and declassification of private data is obtained by manipulating these relations in a semantic-driven way.

Constraining attackers.

The standard notion of non-interference is based on the assumption that an attacker is able to observe public data, without any observational or complexity restriction. The idea is to characterize, in some way, which has to be the power of the attacker that can disclose certain confidential properties from a given program. From this point of view, one of the first approaches that offers a way for weakening the observational capability is the PER model, where the power of the attacker is modeled by equivalence relations [106]. However, some recent papers treat the problem of weakening the power of the attacker in a more specific way.

The notion of non-interference is based on the concept of *indistinguishability* of behaviours: In order to establish that there is no information flow between two objects A and B, it is sufficient to establish that, for any pair of behaviours of the system that differ only in A's object, B's observations cannot distinguish these two behaviours. This observation suggests that it is possible to weaken non-interference by *approximating* this indistinguishability relation [41]. In this paper, the authors replace the notion of indistinguishability by the notion of *similarity*. Therefore, two behaviours, though distinguishable, might still be considered as *effectively* non-interfering, provided that they are similar, i.e., their difference is below a threshold ϵ. A similarity relation can be defined by means of an appropriate notion of distance and provides information on how much two behaviours differ from each other. The power of the attacker is then measured since this quantitative measure of the difference between behaviours is related with the number of statistical tests needed to distinguish the two behaviours.

[1] Note that this is similar to the Cohen's notion of selective dependency [19].

As noted above, the standard notion of non-interference requires that the public output of the program do not contain *any* information (in the information-theoretic sense) about the confidential inputs. This corresponds to an *all-powerful* attacker who, in his quest to obtain confidential information, has no bounds on the resources (time and space) that it can use. Furthermore, in these definitions an "attacker" is represented by an arbitrary function, which does not even have to be a computable function; the attacker is permitted essentially arbitrary power [82]. The observation made in [82] is that, instead, realistic adversaries are bounded in the resources they can use. For this reason, the author provides a definition of secure information flow that corresponds to an adversary working in probabilistic polynomial time, together with a program analysis that allows to certify this kind of information flows.

1.3 The idea: Attackers as abstract interpretations

It is clear that, all the cited papers and works, can weaken non-interference only for some aspects: either by allowing some information flow, or constraining attackers. Our idea is to define a general framework where the notion of non-interference can be weakened both allowing declassification and weakening the power of the attacker, in the same model. First of all, we have to find a way for characterizing how much an attacker may learn from a program. The idea is to consider attackers as static program analyzers, that can (statically) analyze the input/output behavior of programs by "observing" *properties* of data. The goal is to automatically generate, from security polices, a certificate specifying that the given program has only secure information flows. This is statically achieved to provide programs with their appropriate certificates, characterizing the degree of secrecy of a program relatively to what an attacker can analyze about the input/output information flow.

In order to better understand how we mean to weaken the power of the attacker, consider the following program, written in a simple imperative language, where the **while**-statement iterates until x_1 is 0. Suppose x_1 is a secret variable and x_2 is a public variable:

$$\textbf{while } x_1 \textbf{ do } x_2 := x_2 + 2;\ x_1 := x_1 - 1 \textbf{ endw}$$

Clearly, in the standard sense of non-interference, there is a flow of information from x_1 to x_2, since, due to the **while**-statement, x_2 changes depending on the initial value of x_1. This represents the case where no restriction is considered on the power of the attacker. However, suppose that the attacker can observe only the parity of values (0 is even). It is worth noting that if x_2 is initially even, then it is still even independently from the execution of the **while**, and therefore from the initial value of x_1. Similarly if x_2 is initially odd then it remains odd independently from the execution of the while, i.e., from the value of x_1. This means that there's

no information flow concerning parity. In order to model these situations we need to weaken standard non-interference relatively to what an attacker can observe about program information flows. In this way, we are weakening the power of the attacker since we model attackers that can observe properties of values, instead of the values of public variables.
We said above that, in the same model, we want also to characterize the private information that flows in programs, due to the semantics and to the attacker's observational capability. In order to understand how we can characterize *what* flows, consider the following program fragment:

$$l := l * h^2$$

Suppose that the attacker can only observe the parity of the public variable l, then it is clear that if we are interested only in keeping private the sign of h, then the program is secure, since the only information disclosed, in this case, is its parity. In this expression, it is the semantics of the program that puts a *firewall* that hides the sign of h. Therefore, given the model of the attacker, we can characterize, not only *if* there is an information flow, but also *what* is flowing, when it turns out that the program is insecure.

Therefore, we exploit the notion of abstract non-interference by parameterizing standard non-interference relatively to what an attacker can observe, and to what the program's semantics should not release about confidential data. The idea is to consider attackers as static program analyzers whose task is to disclose properties of private data by statically analyzing public resources. In particular, this idea allows us to introduce a notion of non-interference which is relative to the attacker's observation of program execution. Hence, a program ensures secrecy with respect to a given property, which can be statically analyzed by the attacker, if that property on confidential data cannot be disclosed by analyzing public data. For instance, in the first example above, any attacker looking at parity is unable to disclose secrets about confidential data. In this sense the program is secret for parity, while it is not secret relatively to stronger attackers, able to observe more concrete properties of data such as how much a variable grows (e.g. by interval analysis [28]). Since static program analysis can be fully specified as the abstract interpretation of the semantics of the program [28], we can model *attackers as abstract interpretations*.

In this thesis, we apply standard techniques, used in abstract interpretation, in order to define a notion of non-interference, in language-based security, that weakens attackers and allows intentional release of confidential information. In particular, the attacker is modeled by two properties representing what it can observe about public input, and what it can observe about public output. Both these properties are abstract domains, since we suppose that the attacker's observation consists in a static analysis of public data, inputs and outputs. By using these properties, we can do a first weakening of non-interference, similar to what have been done with partial equivalence relations [106], where the public observa-

tions were modeled by equivalence relations. However, we note that we can further weaken this notion by allowing intentional releases of confidential information. For this reason we consider also a property on private data, modeling what, at least, has to be kept private in order to guarantee non-interference. Therefore, abstract non-interference is obtained by a step by step weakening of the standard notion of non-interference. The important aspect of modeling attackers as static program analyzers is that, in this way, we can inherit the whole theory of abstract interpretation for manipulating attackers. In other words, we have that, in abstract non-interference, by transforming abstract domains, we are transforming attackers. This suggested us to use the existing theory on abstract domain transformers for deriving attackers. In particular, the idea is to compare the security degree of programs in the lattice of abstract interpretations. This can be obtained by associating harmless attackers with programs, namely by associating each program with the most concrete observation of public outputs that cannot disclose any confidential property. Hence, we provide a systematic method for extracting, when possible, the most precise (viz. most concrete) attacker for which a program ensures secrecy. This is achieved by a fixpoint construction which simplifies abstract domains towards the most concrete domain for which the program is secret as regards the corresponding property. This "canonical" attacker represents, in the lattice of abstract interpretations, the relative security degree of a program: Any other strictly stronger attacker will violate secrecy. As we said above, abstract non-interference can be also used for modeling the intentional release of confidential information. In particular, we can derive a domain transformer that, given an attacker's model, characterizes which is the maximal amount of information that does flow during the execution of a program. This is important in *declassification*. Declassifying information means downgrading the sensitivity of data in order to accommodate with intentional information leakage. Indeed, if we know the maximal amount of information that flows, then we know which properties can be declassified in order to be sure that non-interference cannot be violated. In particular, if we declassify any more abstract property, then we cannot avoid confidential information leakages. Moreover, the algebra of domain transformers allows us to prove that these two transformations, providing the most concrete harmless attacker and the maximal amount of confidential information leaked, are adjoint functions, formalizing the intuitive dualism existing between these two notions.

Abstract non-interference is defined by using denotational semantics, which can be not very useful in practice. For this reason, we introduce a compositional proof-system for certifying abstract non-interference in programming languages, in a syntax-directed way. Certifying abstract non-interference means proving that the program satisfies an abstract non-interference constraint relatively to some given abstraction of its input/output, and we derive this certification inductively on the program's syntax, by defining a proof system. Assertions in the proof-system have the form of Hoare triples: $(\eta)P\ (\rho)$ where P is a program fragment and η and ρ

are abstractions of program's data. However, the interpretation of abstract non-interference assertions is rather different from partial correctness assertions (see [9]): $(\eta)P\ (\rho)$ means that P is unable to disclose secrets if input and output values of public variables are approximated respectively in η and ρ. Hence, abstract non-interference assertions specify the secrecy of a program relatively to a given model of an attacker and the proof-system specifies how these assertions can be composed in a syntax-directed *a la* Hoare deduction of secrecy. The proof-system provides a deeper insight in abstract non-interference, by specifying how assertions concerning secrecy compose with each other. This is essential for any static semantics for secrecy devoted to derive certificates specifying the secrecy degree of programs.

1.4 Abstract non-interference: A versatile notion

The semantic approach to non interference is interesting since it makes non-interference a problem of the semantics chosen. In other words, the kind of non-interference that we want to enforce depends on the semantics chosen for modeling a computational system. This observation has been the key point in order to make abstract non-interference a general notion. Indeed we noticed that, simply by considering the trace semantics instead of the denotational one, we are able to model non-interference in presence of attackers able to observe the public memory of the system during the whole computation, observing all the variations of the public memory. On the other hand, if we consider a non deterministic denotational semantics, then we can model the possibilistic non-interference for non-deterministic systems, without changing the definition of abstract non-interference. While, if we consider denotational semantics, modeling non termination of programs, then we obtain notion of non-interference avoiding termination channels, namely that guarantee the soundness of the definition even in presence of attackers able to observe non termination of programs. Finally, we observed that by considering a semantics measuring also the time elapsed during the computation, and by considering this time value as a public variable, then we model abstract non-interference that avoids timing channels, namely that guarantees the soundness of abstract non-interference even if attackers are able to observe the time elapsed during computations.

Anyway, we noticed that changing the semantics was not sufficient in order to cover notions of non-interference defined on systems different from imperative languages, such as process algebras and timed automata. This also because the given notion of non-interference is based on the notion of *variable*, that is significant only in programming languages, and uses denotational semantics, that cannot model all the computational systems. Moreover, in literature, there exist different notions of non-interference that better model the confinement problem in systems based on finite state automata. In particular, while when considering the semantics, we allow the private to interfere with the public as long as this interference is not visible observing the output, in process algebras, for example,

non-interference is not violated whenever private actions do not interfere with the sequence of public actions at all. For this reason we start considering computational systems modeled by their computational trees, embodying both their branching and linear nature. At this point, we define generalized abstract non-interference by using three abstractions: the first decides the semantic model on which we define non-interference, e.g, denotational semantics or finite state automata, the second decides what should be the maximal information observable by an attacker, e.g., all the computations with a fixed private input or all the computations where private actions are not executed, finally the third characterizes the observational capability of the attacker. This last abstraction is the one that we can transform in order to guarantee non-interference, and which can be used for characterizing the most concrete harmless attacker for the given notion of non-interference, in the given computational system.

1.5 Algebra of domain transformers

The most important aspect of defining non-interference by using abstract interpretation, is the possibility of *systematically* characterizing the secrecy degree of programs, and in general of computational systems, by *transforming* abstract domains, those abstract domains used for modeling the attackers. For this reason, is important to introduce the theory of abstract domain transformers, and the theory of how they can be designed. Standard abstract interpretation provides advanced methods for the calculational design of static analyzers (see [26] for a fully detailed example) from a formally defined semantics of the programming language and from some given specification of how semantics has to be approximated. However, no such methodologies, neither for deriving nor for classifying abstract domain transformers, are known to provide an analogous calculational design of domain operations. For this reason, in this thesis we study also an algebra of abstract domain transformers. This algebra provides us with the right tools and notions for defining and classifying abstract domain transformers. The interest in such an algebra is, clearly, not limited to the field of abstract non-interference treated in this thesis. It allows to lift abstract interpretation of one level up, from denotations to domains and therefore it is a general characterization that can be applied in many field of theoretical computer science. In recent years we observed a growing interest in systematic design methods for program analysis frameworks. This is mainly justified by the fact that the most successful static analyzers are parametric with respect to the property of interest, in order to easily handle the variety of possible analyses which can be designed and studied for programs. Moreover, automatic methods for tuning these analyses in accuracy and costs are needed in order to avoid reimplementation of analyses when these are modified.

Indeed one of the most fundamental facts of abstract interpretation is that most of its properties in approximating semantics, like precision, completeness,

and compositionality, which may involve complex operators, fixpoints etc., all depend upon the notion of *abstraction*, which is precisely and uniquely specified by the chosen domain of properties [31]. Central in the construction of an abstract interpretation is therefore the notion of *domain*. This is the case for instance in program analysis, in type inference and in comparative semantics, where the various abstract (approximate) semantics all correspond to suitable abstractions, namely domains. In the literature, most of well known operations for refining and simplifying domains are the result of either solutions to specific problems in refining or simplifying domains (viz. disjunctive completion [31, 34], complete refinements and kernels [65], reduced power [31] and Heyting completion [66]) or inherited directly from the basic structure of the lattice of abstract interpretations (viz. reduced product [31], complementation [23]). Therefore, we would like to study a general framework for the calculational design of abstract domain operations, in such a way that systematic modification methodologies can be designed to modify generic abstract interpreters, and moreover this framework would be the perfect context for defining and classifying new abstract domain transformers. The main idea to solve the problem of systematically design domain transformers is to use the same abstract interpretation framework but now lifted one level up: the object of discourse are domains instead of program state descriptions. The use of abstract interpretation in higher types, later called higher-order abstract interpretation[2] will show the potential of abstract interpretation methods for reasoning about abstract domain transformers.

We can show that the standard Cousot and Cousot theory of abstract interpretation, based on the so called adjoint-framework of Galois connections, can be directly applied to reason about abstract domain operations, yet providing formal methodologies for the systematic design of abstract domain transformers. In particular, most domain transformers can be viewed as suitable problems of achieving precision with respect to some given semantic feature of the programming language we want to analyze. This observation has indeed an intuitive justification: the goal of refining a domain is always that of improving precision with respect to some basic semantic operation (e.g., arithmetic operations, unification in logic programs, data structure operations in simple and higher-order types). Analogously, simplifying domains corresponds to the dual operation of reducing precision with respect to analogous semantic operations. What turns out, is that most well known operations for transforming domains can be interpreted in this way and that the relation between refinement and simplification on domains is indeed an instance of the same abstract interpretation framework lifted to higher types, i.e., where the objects of abstraction/concretization are abstract domains. In particular, we study how we can reverse abstract domain transformers, seen as closure operators. We show that there exists two ways for reversing domain transformers: either con-

[2] There is a fundamental distinction between this use of the term *higher-order* and the one used in [34]. In [34] abstract interpretation theory was in fact applied to higher-order programming languages.

sidering their right adjoint functions, or considering their left adjoint functions, when one or both of them exist. Depending on which inversion we can do, we can interpret the domain transformer in different but precise ways. In particular, if a refinement admits the left adjoint, then we call the refinement *shell*, in the sense that it adds elements in order to guarantee a given property, and its adjoint *core*, which erases elements in order to guarantee that the same property holds, exactly as it happens for the completeness transformers [65]. While, if a refinement admits the right adjoint then we call it an *expander*, and its inverse is a *compressor*, since it finds the most abstract domain with the same refinement. We show that it is always possible to make a transformer right reversible, while, even if we can characterize left reversible refinements, we don't have a method for making refinements left-reversible.
Moreover, in the context of abstract domain completeness, we show a systematic method for checking if an abstract domain completeness refinement admits the corresponding compressor. All these studies provide an *algebra* of abstract domain transformers, that for completeness transformers is even more specified and complex. Completeness transformers are considered specifically, since the domain transformers used in the following of the thesis are completeness ones, but also because, in a more general setting, it is possible to show that most of the well known domain transformers are indeed abstract domain completeness transformers.

State of the art.

Any formal method to compare or transform abstract interpretations is inherently based on corresponding methods to compare and transform abstract domains. A domain, at any level of abstraction, is a set of mathematical objects which represent the properties of interest about a dynamic system, partially ordered with respect to their relative degree of precision. In program analysis, for instance, the design of analyzers corresponds to study a particular abstract domain, and modifying domains corresponds to modify analyses. As recently proved (e.g., see [107] for a reconstruction of groundness analysis in logic programming), the design of a complex abstract domain is the result of a number of steps which can be in some cases ingegnerized by applying suitable domain transformers to simpler domains for the property of interest.

The foundation for a theory of abstract domains was fixed by Cousot and Cousot in [31]. In that work the authors gave the main structure of abstract domains enjoying *Galois connections* and some fundamental operators for systematically composing domains in order to achieve attribute independent and relational analyses (respectively the reduced product and power operations). Since then, a number of papers developed new domain operations and studied the impact of these operations in the design of abstract interpreters for specific program analysis and languages. These include Cousot and Cousot's *reduced product*, *disjunctive completion* and *reduced cardinal power* [31, 32, 34]; Nielson's *tensor product* [98]; Giacobazzi et al. *dependencies*, *dual-Moore-set completion*, *complete kernels and*

shells, *Heyting completion*, and *least disjunctive basis* in [62, 65, 66]; and Cortesi et al. *open product*, *pattern completion*, and *complementation* [23, 24]. The notion of *domain refinement* and *domain simplification*, introduced in [44, 61], provided the very first generalization of these ideas. Intuitively a refinement is any operator performing an action of refinement with respect to the standard ordering of precision, e.g. by adding information to domains; while simplifications and compressors perform the dual action of "weeding out" information from domains.

1.6 Structure of thesis

This thesis is structured as follows. In Chapter 2 we introduce the basic algebraic notions that we are going to use in the following of this thesis. In Chapter 3, we introduce the study of the algebra for abstract domain transformers. In particular, we show that the standard Cousot and Cousot theory of abstract interpretation, based on the so called adjoint-framework of Galois connections, can be directly applied to reason about abstract domain transformers, yet providing formal methodologies for the systematic design of these domain transformers. The key point in this chapter is that most domain transformers can be viewed as suitable problems of achieving precision with respect to some given semantic feature of the programming language we want to analyze. This is exactly the philosophy that leads us to the design of the transformer providing the most powerful harmless attacker in abstract non-interference. In Chapter 4 we introduce the computational systems that are considered in the following of this thesis and the possible semantics that can be used for modelling them. In Chapter 5, we provide an excursus on the different notions of non-interference, in different computer science fields, and we describe the main approaches studied (see [104] for a survey). In particular, we first provide a brief background about the notion of non-interference, and then we provide a background about the existing techniques used for *enforcing* non-interference. We conclude the chapter by describing some possible weakenings of the notion of non interference, existing in literature.
In Chapter 6, we introduce the notion of *abstract non-interference* by parameterizing standard non-interference relatively to what an attacker is able to observe. We use this notion, for characterizing the secrecy degree of programs in the lattice of abstract interpretations, by deriving the most powerful harmless attacker for any program. Moreover, we model in abstract non-interference also the intentional release of information and we derive an abstract domain transformers, characterizing the maximal amount of information that flows. We conclude the chapter by showing how we can enrich the notion simply by enriching the considered semantics, and by comparing abstract non-interference with two of the most related works: the PER model [106] and robust declassification [118]. In Chapter 7, we introduce a compositional proof system whose aim is to certify abstract non-interference in programming languages, inductively on the syntax of the languages, following an

a la Hoare deduction of secrecy.
In Chapter 8, we show that abstract non-interference can be formalized as a completeness problem, in the standard framework of abstract interpretation. This allows us to characterize the derivation of the most concrete public observer, i.e., harmless attacker (defined in Chapter 6), and the derivation of the most abstract private observable of the program (related to declassification) as adjoint domain transformers. This adjoint relation formalize the intuitive dualism between these two approaches for weakening non-interference.
In Chapter 9, we introduce the notion of *timed abstract non-interference*, providing the appropriate setting for studying how properties, of private data, interfere, during the execution of programs, with properties of the elapsed time. In this way we obtain an abstract non-interference that avoids timing channels, namely information transmissions due to the capability of attackers of observing time.
Finally, in Chapter 10, we prove that abstract non-interference introduced in Chapter 6 can be generalized in order to cope with many well-known models of secrecy in sequential, concurrent and real-time systems and languages. This is achieved by factoring abstractions in order to identify sub-abstractions modeling the different properties of the system on which the notions of non-interference are based.
The thesis ends with Chapter 11, where we sum up the work done and we briefly describe all the new ideas that we would like to exploit.

2

Basic Notions

I am ignorant of absolute truth. But I am humble before my ignorance,
and therein lies my honour and my reward.
KHALIL GIBRAN

In this chapter, we introduce the basic algebraic notions that we are going to use in this thesis. In particular, we describe the mathematical background, recalling the notions of sets, relations and functions [79]. Afterwards, we also give a brief description of the Scott hierarchy of ordered structures [67]. Moreover, we recall the notion of fixpoint together with its constructive characterization [110, 30]. We introduce abstract interpretation [28, 31], providing the fundamental characterizations of abstract domain existing in literature. We describe the notions of sound and complete abstractions, and we define the domain transformers that make abstract domains complete [65].

2.1 Mathematical background

2.1.1 Sets

A *set* is a collection of objects (or elements). Typical examples of sets are natural numbers $\mathbb{N}$, integer numbers $\mathbb{Z}$, and rational numbers $\mathbb{Q}$. The notation $x \in A$, where A is a set, denotes that the object x is an element of A, namely we say that x belongs to A. The notation $A \subseteq B$ means that A is subset of B, namely that each element of A belongs also to B. For example, we write that $\mathbb{N} \subseteq \mathbb{Z}$. If A and B are two sets, we say that they are equal, i.e., $A = B$, if A is subset of B and viceversa, i.e., if $A \subseteq B$ and $B \subseteq A$. We denote $A \subset B$ (or $A \subsetneq B$) the relation of properly contained, namely $A \subseteq B$, but there exists an element in B that is not in A. For example, we have that $\mathbb{N} \subsetneq \mathbb{Z}$. In general, given two sets A and B, if it exists an element in A [in B] such that it is not in B [in A] then we write

$A \neq B$. The empty set, $\varnothing$, is the set without any element. This means that for each element x we have $x \notin \varnothing$, and for each set A we have $\varnothing \subseteq A$.
Let A and B be two sets. The set $A \cup B$ (*union*) is the set of all the objects belonging to A or to B: $A \cup B \stackrel{\text{def}}{=} \{\, x \,|\, x \in A \ \vee\ x \in B \,\}$.
The set $A \cap B$ (*intersection*) is the set of all the objects belonging both to A and to B: $A \cap B \stackrel{\text{def}}{=} \{\, x \,|\, x \in A \ \wedge\ x \in B \,\}$. The set $A \setminus B$ is the set of all the elements of A that are not in B: $A \setminus B \stackrel{\text{def}}{=} \{\, x \,|\, x \in A \ \wedge\ x \notin B \,\}$. We say that two sets are *disjoint* if their intersection is the empty set. Let A be a set, we define the *powerset* of A, i.e., $\wp(A)$, as the collection of all the subsets of A: $\wp(A) \stackrel{\text{def}}{=} \{\, X \,|\, X \subseteq A \,\}$.

Relations

Let us see how it is possible to relate elements of sets. Let X be any set, and consider two elements $a, b \in X$, we call *ordered pair* the element $\langle a, b\rangle$ such that $\langle a, b\rangle \neq \langle b, a\rangle$. For each $n \geq 2$ we define the ordered n-tuple of n objects $a_0, a_1, \ldots, a_{n-1}$ by $\langle \ldots \langle \langle a_0, a_1\rangle, a_2\rangle, \ldots\rangle$, denoted by $\langle a_0, a_1, \ldots, a_n\rangle$.

Definition 2.1 (Cartesian product). *Let $\{A_i\}_{i<n}$ be n sets. We define the cartesian product of the n sets A_i as the set*

$$A_0 \times A_1 \times \ldots A_n \stackrel{def}{=} \{\, \langle a_0, a_1, \ldots, a_n\rangle \,|\, \forall i < n.\, a_i \in A_i \,\}$$

For any set S and for each $n \in \mathbb{N}$, $n \geq 1$, S^n denotes the n-th cartesian self product of S. A generic tuple in S^n is denoted by σ, σ_i denotes its i-th component, and $\sigma[y/i]$ denotes the tuple obtained from σ by replacing σ_i with y. In general, a *relation* between the elements of a set A and the elements of a set B is a subset of the cartesian product $A \times B$. In the particular case when $A = B$, a subset of $A \times A$ is said *binary relation* on A. Let us see the possible properties of a relation, and the main kinds of relation existing.

Definition 2.2 (Equivalence relation). *We say that a binary relation $\mathcal{R}$ on A is an equivalence relation on A if the following conditions on $\mathcal{R}$ hold:*

(i) $\forall x \in A\,.\,\langle x, x\rangle \in \mathcal{R}$ *(reflexivity);*
(ii) $\forall x, y \in A\,.\,\langle x, y\rangle \in \mathcal{R} \ \Rightarrow\ \langle y, x\rangle \in \mathcal{R}$ *(symmetry);*
(iii) $\forall x, y, z \in A\,.\,\langle x, y\rangle \in \mathcal{R} \ \wedge\ \langle y, z\rangle \in \mathcal{R} \ \Rightarrow\ \langle x, z\rangle \in \mathcal{R}$ *(transitivity).*

In the following, whenever $\langle x, y\rangle \in \mathcal{R}$ we will also write $x\mathcal{R}y$. If $\mathcal{R}$ fails the reflexive condition then it is said *partial equivalence relation* (PER).
If A is a set with an equivalence relation $\mathcal{R}$, we consider, for each $x \in A$, the subset A_x of A containing all the elements $y \in A$ such that $y\mathcal{R}x$. These sets are called *equivalence classes* of A as regards the relation $\mathcal{R}$, and usually they are denoted by $[x]_{\mathcal{R}}$, with $x \in A$.

Definition 2.3 (Partition). *Let X be any set. $\mathcal{P} \subseteq \wp(X)$ is a partition of X if*

- $\bigcup_{P \in \mathcal{P}} P = X$;

- *For all $P_1, P_2 \in \mathcal{P}$ either $P_1 = P_2$ or $P_1 \cap P_2 = \varnothing$.*

An equivalence relation induces a partition of the set on which it is defined, and the elements of the partition are its equivalence classes. Each element of an equivalence class can be used for uniquely representing the class. For this reason, in the following, we sometimes identify partitions with equivalence relations.
Another important kind of relation is the order relation. It allows to order the objects of a set in different ways.

Definition 2.4 (Partial order). *Let P be a set. A partial order on P is a binary relation $\leq$ such that, $\forall x, y, z \in P$:*

(i) $x \leq x$ (reflexivity),
(ii) $x \leq y \ \wedge \ y \leq x \ \Rightarrow \ x = y$ (antisymmetry),
(iii) $x \leq y \ \wedge \ y \leq z \ \Rightarrow \ x \leq z$ (transitivity).

An order relation $\mathcal{R}$ on P is said to be *strict* if it is *irreflexive*, i.e., $\forall x \in P$ we have $\langle x, x\rangle \notin \mathcal{R}$. Clearly a non reflexive relation is not necessarily irreflexive. An order relation on X is said *linear* order relation if every two elements of X are comparable in the relation.

Definition 2.5 (Wellfounded relation). *Let $\mathcal{R}$ be a binary relation on a set X. We say that $\mathcal{R}$ is well-founded, if for every nonempty set $Y \subseteq X$ there exists $z \in Y$ such that $\langle y, z\rangle \notin \mathcal{R}$ for any $y \in Y \setminus \{z\}$. The relation $\mathcal{R}$ is* strictly *well-founded if it is well-founded and irreflexive.*

A linear, well-founded partial order is called *well-order*. In particular, a partial order relation $\leq$ on a set X is well-founded if for every nonempty $Y \subseteq X$, the poset $\langle Y, \leq\rangle$ has a minimal element.

Functions

We now introduce a particular class of relations on sets. Let A and B be two sets. A *function* (or map) f from A to B is a relation between A and B satisfying the following condition: for each $a \in A$ there exists exactly one $b \in B$ such that $\langle a, b\rangle \in f$, in this case we write $b = f(a)$. Usually, such a function is denoted as $f : A \to B$ where A is called the *domain* of the function and B *co-domain* of f. We will denote by $a \mapsto f(a)$ the fact that f associates with a the value $f(a)$. If there exists an element $a \in A$ such that $f(a)$ is not defined, then we say that f is *partial*, otherwise f is said to be *total*. The set $f(X) \stackrel{\text{def}}{=} \left\{ \ f(x) \,\middle|\, x \in X \ \right\}$, is the *image* of $X \subseteq A$ under f. The image of the whole domain is called *range* of f. Instead, the set $f^{-1}(X) \stackrel{\text{def}}{=} \left\{ \ y \in A \,\middle|\, f(y) \in X \ \right\}$ is the the *inverse image* of $X \subseteq B$ under f. Let us see some fundamental properties of functions.

Definition 2.6. *Let $f : A \to B$ be a function. We say that f is* one-to-one *or* injective *if for each $a_1, a_2 \in A$ we have that $f(a_1) = f(a_2)$ implies $a_1 = a_2$. We say that f is* onto *or* surjective *if $f(A) = B$.*

Hence, a one-to-one function maps distinct elements in distinct elements, while an onto function takes all the elements of the co-domain.
In general, if between two sets there exists a function that is both one-to-one and onto, then the function is called *bijection*, and the two sets are *isomorphic*. Finally, a particular kind of function is the *identity* map, i.e., $\iota_A : A \to A$, which associates, with each element of A, the element itself.
Given two functions $f : A \to B$ and $g : B \to C$, their *composition* is the function $g \circ f : A \to C$ which associates, with each element $a \in A$, the element $(g \circ f)(a) \stackrel{\text{def}}{=} g(f(a)) \in C$. Sometimes, in sake of simplicity, we will denote by gf the composition $g \circ f$.
Moreover, we will denote by $\lambda x.f(x)$ the function f on the variable x. Consider two sets S and T, and a natural number $n \in \mathbb{N}$, if $f : S \to T$ then $\langle f, ..., f\rangle : S^n \to T^n$ denotes the componentwise extension of f, i.e., $\langle f, ..., f\rangle = \lambda\sigma.\langle f(\sigma_1), ..., f(\sigma_n)\rangle$.

Cardinalities, ordinals and induction

In this section, we describe how it is possible to *measure* the number of elements contained in a set. The idea is to *count*, i.e, to associate with each element a natural number. Let us see, how we can associate with each natural number a set [79].

Definition 2.7. *For each natural number let us define recursively the following set:*

$$\begin{cases} 0 \stackrel{def}{=} \emptyset \\ k+1 \stackrel{def}{=} k \cup \{k\} \end{cases}$$

In this way we can define $\omega \stackrel{\text{def}}{=} \left\{ \; n \,\middle|\, n \in \mathbb{N} \; \right\}$. Let us consider some properties of these sets.

Definition 2.8. *Let X be any set.*

(a)$\forall n \in \omega$ we define $\cup^n X$ as

$$\begin{cases} \cup^0 X = X \\ \cup^{k+1} X = \bigcup(\cup^k X) \end{cases}$$

(b) The transitive closure *of X is defined as $TC(X) = \bigcup \left\{ \; \cup^n X \,\middle|\, n \in \omega \; \right\}$.*
(c) A set is transitive *if $TC(X) = X$.*

Example 2.9. Consider $TC(3)$, where $3 = \{0, 1, 2\}$.

$$\begin{aligned} &\cup^0 3 = \{0, 1, 2\} = \{\emptyset, \{0\}, \{0, 1\}\} \\ &\cup^1 3 = \bigcup\{\emptyset, \{0\}, \{0, 1\}\} = \{0, 1\} = \{\emptyset, \{0\}\} \\ &\cup^2 3 = \bigcup\{\emptyset, \{0\}\} = \{0\} \\ &\vdots \end{aligned}$$

Hence, $TC(3) = \{0, 1, 2\} = 3$, namely 3 is transitive. Let us consider now $X = \{\{0\}, \{1\}, \{2\}\}$.

$$\begin{aligned} &\cup^0 X = X \\ &\cup^1 X = \bigcup\{\{0\}, \{1\}, \{2\}\} = \{0, 1, 2\} \\ &\vdots \end{aligned}$$

Namely $TC(X) = X \cup \{0, 1, 2\} \neq X$, which means that X is not transitive.

We say that two sets A and B are *equipotent*, $A \approx B$, if there exists a one-to-one function from A onto B. Hence, a set A is finite if $A \approx n$, for some $n \in \mathbb{N}$, while A is a *numerable* infinite set if there exists a one-to one function $f : \omega \to A$.
The relation $\approx$ is an equivalence relation, whose equivalence classes are called *cardinalities*, denoted by $|\cdot|$. Namely $|X|$ is the collection of all the sets Y such that $Y \approx X$. In particular there exists $n \in \mathbb{N}$ such that $n \approx X$, which identifies the cardinality. In all the numerable sets, namely with at most the cardinality of natural numbers, we can use a technique, called *induction*, for proving or defining properties.

Principle 2.10 (Induction) *A property p holds for all $n \in \mathbb{N}$, i.e., $p(n)$ true for all $n \in \mathbb{N}$, iff*

1. *$p(0)$ is true;*
2. *For each $n \in \mathbb{N}$, $n > 0$, we have that $p(n)$ true implies $p(n + 1)$ also true.*

Now we extend the previous notions in order to cope also with sets with a number of elements greater than ω, i.e., with *non-numerable* sets.

Definition 2.11 (ordinal). *A set α is an ordinal if α is transitive and strictly well-ordered by $\in$.*

In the following, $\mathbb{O}$ will denote the class of ordinals. Give two ordinals α and β, we say that $\alpha \leq \beta$ if $\alpha \in \beta$, and therefore $\alpha \subseteq \beta$. For each ordinal α, its successor $S(\alpha) = \alpha \cup \{\alpha\}$ is an ordinal. Hence, α can be a finite ordinal only if it is a natural number, and the first infinite ordinal is ω. In particular ω is the smallest ordinal α such that

$$\begin{cases} \alpha \neq 0 \\ \forall \beta \in \alpha \,.\, S(\beta) \in \alpha \end{cases}$$

Let us define the *least upper bound* (see Def. 2.18) of a set of ordinals.

Theorem 2.12. *Let $A \subseteq \mathbb{O}$, then*

- *$\bigcup A \in \mathbb{O}$;*
- *If $\forall \alpha \in A \,.\, \exists \beta \in A \,.\, \alpha \leq \beta$, then $\bigcup A$ is the smallest ordinal that exceeds all ordinals in A.*

Each ordinal α is called *successor ordinal* if there exists an ordinal β such that $\alpha = S(\beta)$. An ordinal which is not a successor ordinal is called a *limit ordinal*.
Now we extend the induction technique to non-numerable (transfinite) sets.

Principle 2.13 (Transfinite induction) *A property p holds for each ordinal α if*

1. *$p(0)$ holds;*
2. *For each ordinal β, if $p(\beta)$ is true then also $p(S(\beta))$ is true;*
3. *For each limit ordinal γ, if we have that $\forall \eta < \gamma$ we have that $p(\eta)$ holds, then also $p(\gamma)$ holds.*

See [79] for a complete description of the arithmetic on ordinals.

2.1.2 Algebraic ordered structures

A set is a collection of objects without any kind of structure. We would like to work with structures that embody the relations existing among their objects.

Poset

Let us consider first, the structures obtained combining sets with order relations.

Definition 2.14 (poset). *A set P with an order relation $\leq$ is said (partial) ordered set, denoted by $\langle P, \leq_P \rangle$, and it is called poset.*

Depending on the relation existing between the elements of a poset, we can classify the structure in different ways. In particular, if, in a poset P, all the pairs of elements are in the relation $\leq_P$, then we say that P is a set with a *total* order and P is said *chain*.

Definition 2.15 (Chain). *A P poset is a chain if, for each $x, y \in P$, we have $x \leq_P y$ or $y \leq_P x$.*

A chain is a totally ordered set. A set X is an *anti-chain* if for each pair of elements $x, y \in P \setminus \{\bot, \top\}$, we have $x \not\leq_P y$ and $y \not\leq_P x$. A typical example of partial ordered set is the powerset $\wp(X)$ of any set X. This poset is composed by all the subsets of X and it is ordered by inclusion. This is a partial order since not all the elements are in this order, e.g., $\{a, b\} \not\subseteq \{b, c\}$ and $\{a, b\} \not\supseteq \{b, c\}$, for each $a, b, c \in X$. Typical examples of totally ordered sets are the sets of numbers with the classical order relation. Typical examples of anti-chains are the flat domains.

At this point we can think of inverting the order in a poset P. Namely, given a generic poset P, we can build a new poset P^δ (*dual* of P) defining $x \leq_{P^\delta} y$ iff $y \leq_P x$. This definition leads to the following principle:

Principle 2.16 (Duality) *Given any theorem Φ, true for all the posets, its dual, Φ^δ, holds for all the posets.*

On a poset $\langle P, \leq_P \rangle$ we can define two families of sets depending on the order $\leq_P$.

Definition 2.17. *Let $\langle P, \leq_P \rangle$ be a poset. We say that $Q \subseteq P$ is an* ideal *of P if we have that $\forall x \in Q, y \in P \,.\, y \leq_P x \;\Rightarrow\; y \in Q$. We say that Q is a* filter *if it is the dual of an ideal.*

These two sets can be build starting from a subset of P. In particular, the *filter closure* of a set $Q \subseteq P$, is the set $\uparrow Q \stackrel{\text{def}}{=} \{\, y \in P \mid \exists x \in Q \,.\, x \leq_P y \,\}$, where $\langle P, \leq_P \rangle$ is a poset. The set $\downarrow Q$, *ideal closure*, is dually defined. We use the simplified notation $\uparrow x$ and $\downarrow x$ for denoting $\uparrow \{x\}$ and $\downarrow \{x\}$.

Definition 2.18. *Let $\langle P, \leq_P \rangle$ be a poset, and let $X \subseteq P$. We say that a is an* upper bound *of X if $\forall x \in X \,.\, x \leq_P a$, a is said* maximal *if it also belongs to X.*
If the set of upper bounds has the smallest element m, then we call this element least upper bound *of X (shortly* lub*) and we write $m = \bigvee X$.*
If the lub belongs to P then it is said maximum *(or top) and it is usually denoted by $\top$.*

Dually, we define the notions of lower bound, minimal element, greatest lower bound (*glb*) of a set X, denoted by $\bigwedge X$, and minimum (or bottom), denoted by $\bot$. In the following, we will denote by $max(X)$, the set of all the maximal elements in X, and by $min(X)$ the set of all the minimal ones.
We can note that if a poset has top (or bottom), then it is necessary unique for the antisymmetry property of the order relation. In general, we denote by $x \wedge y$ and $x \vee y$, respectively the elements $\bigwedge\{x, y\}$ and $\bigvee\{x, y\}$.

Lattices and cpo

Let us consider some more complex ordered structures.

Definition 2.19 (Directed set). *Let P be a poset. We say that P is a directed set if each non-empty finite subset of P has least upper bound in P.*

For example a chain is always a directed set. Let us define a complete partial ordered set in the following way.

Definition 2.20 (cpo). *A complete partial order on directed sets (*dcpo *or* cpo*) is a poset $\langle P, \leq_P \rangle$ such that $\bot \in P$ and for each directed set D in P we have $\bigvee D \in P$.*

Every finite poset is a cpo. The set of all the natural numbers, with the usual order, is a cpo only if we add the limit element ω, since, without it, we don't have, in the set, the least upper bound of all the natural numbers.

Proposition 2.21. *A poset $\langle P, \leq_P \rangle$ is a cpo iff each chain in P has least upper bound.*

Let us introduce the important notion of lattice.

Definition 2.22. *Let $\langle P, \leq_P \rangle$ be a poset. P is a* semi-lattice *as regards $\vee_P$ [$\wedge_P$] if $\forall x, y \in P \,.\, x \vee_P y \in P$ [$\forall x, y \in P \,.\, x \wedge_P y \in P$]. It is a* lattice *if it is a semi-lattice as regards both $\vee_P$ and $\wedge_P$. The lattice is said to be* complete *if $\forall S \subseteq P \,.\, \bigvee_P S \in P$ and $\bigwedge_P S \in P$.*

The lattice described in Def. 2.22 is denoted by $\langle P, \leq_P, \vee_P, \wedge_P, \top_P, \bot_P \rangle$. A typical example of complete lattice is the powerset of X where the glb operation is the intersection, while the lub operation is the union of sets. Let us consider now other characterizations of complete lattices.

Theorem 2.23. *Let P be a poset, $P \neq \varnothing$. The following assertions are equivalent:*

(i) P is a complete lattice;
(ii) For each subset S of P we have that $\bigwedge_P S$ exists in P;
(iii) P has the $\top$ element and for each non-empty S of P we have $\bigwedge_P S$ exists in P.

In the following, we will call *domain* a generic ordered structure [1]. Note that, if P is a complete lattice, then a cartesian product P^n (called *direct product*) is still a complete lattice under the canonical componentwise ordering induced from P. The following particular kind of complete lattice is very important when speaking of abstract interpretation

Definition 2.24 (Moore family). *Let L be a complete lattice. $X \subseteq L$ is a Moore family of L if $X = \mathcal{M}(X) \stackrel{def}{=} \left\{ \bigwedge S \,\middle|\, S \subseteq X \right\}$, where $\bigwedge \varnothing = \top \in \mathcal{M}(X)$.*

In general, for each $X \subseteq L$, $\mathcal{M}(X)$ is called *Moore closure* of X in L, i.e., $\mathcal{M}(X)$ is the smallest (w.r.t. inclusion) subset of L which contains X and which is a Moore family of L. Let us define the notion of *irreducible* elements of a lattice. In a complete semi-lattice, an element is irreducible if it cannot be obtained as glb [lub] of two other elements.

Definition 2.25 (Meet-irreducible). *An element $p \in L$, $p \neq \top$, with L being a lattice, is called meet-irreducible if $p = x \wedge_L y$ implies $p = x$ or $p = y$.*

In the following we will denote by $Mirr(L)$ the set of meet-irreducible elements in a lattice L. The *join-irreducible* elements are dually defined and are denoted by $Jirr(L)$. We say that a lattice L is *meet-generated* by its meet-irreducibles iff $L = \mathcal{M}(Mirr(L))$, namely if the Moore closure of these elements generates each element of the lattice. Dually, we can define a *join-generated* lattice. The following proposition shows how it is possible to characterize the meet-irreducible elements of a powerset.

Proposition 2.26. *Let A be a set, and $X \subset A$, then X is meet-irreducible in $\wp(A)$ iff there exists $a \in A$ such that $X = A \smallsetminus \{a\}$.*

[1] Recall that, for Scott, a domain is a ω-dcpo, namely a domain in which the limits of directed sets with numerable cardinality exist.

Corollary 2.27. $\wp(A) = \mathcal{M}(Mirr(\wp(A)))$

Let us define now the set of atoms [co-atoms] of a lattice.

Definition 2.28 (Atom). *Let P be a poset with bottom $\bot$. We say that $a \in P$, $a \neq \bot$, is an atom of P if for each $x \in P$, with $x \neq \bot$, we have that $\bot \leq_P x \leq_P a$ implies $x = a$.*

The notion of *co-atom* is dually defined. A lattice is called *atomistic* when it is join-generated by its atoms. A lattice is *co-atomistic* if it is meet-generated by its co-atoms. Therefore, if a lattice is co-atomistic, then the set of its *Mirr* coincides with the set of its co-atoms.

Definition 2.29 (ACC lattice). *Let P be a poset. We say that P satisfies the ascending chain condition (ACC) if for each $x_1 \leq x_2 \leq \ldots \leq x_n \leq \ldots$, increasing sequence of elements of P, there exists $k \in \mathbb{N}$ such that $x_k = x_{k+1} = \ldots$.*

Dually, we can define a *DCC* lattice as a lattice without infinite descending chains. At this point, we can give the following characterizations of complete lattices.

Theorem 2.30. *Let P be a poset*

(i) If P has the bottom, $\bot$, and it is ACC then it is a complete lattice;
(ii) If P is both ACC and DCC then P is a complete lattice.

Functions on domains

Let us consider functions defined on ordered structures.

Definition 2.31 (Monotone function). *Let $\langle P, \leq_P \rangle$ and $(Q, \leq_Q)$ be two posets. A function $f : P \to Q$ is monotone (or* order-preserving*) if for each $x, y \in P$ such that $x \leq_P y$ we have that $f(x) \leq_Q f(y)$.*

Namely, a monotone function preserves the order between two structures. The function is an *order-embedding* when $x \leq_P y$ iff $f(x) \leq_Q f(y)$. The set of all the monotone functions between two posets is denoted by $(P \xrightarrow{\text{m}} Q, \sqsubseteq)$, it is ordered *pointwise*, i.e., $f \sqsubseteq g$ if, for each $x \in P$ we have $f(x) \leq_Q g(x)$), and it is called *space of the monotone functions* between P and Q. The following kind of functions is very important in the study of semantics of programs.

Definition 2.32 ((Scott-)continuous function). *Let P and Q be two cpo. A function $f : P \to Q$ is (Scott-)continuous if it is monotone and for each directed set D of P, we have $f(\bigvee_P D) = \bigvee_Q f(D)$.*

This means that a function is continuous if it preserves the limits of directed sets. Moreover, by Proposition 2.21 [87], this notion can be equivalently formalized on chains instead of on directed sets, namely $f \in P \to Q$ is continuous iff f preserves *lub*'s of (non-empty) chains. We denote the space of continuous functions between P and Q, ordered pointwise, as $(P \xrightarrow{\text{c}} Q, \sqsubseteq)$. A function that preserves the bottom

element, $\bot$, is called *strict*. We denote the space of the strictly continuous functions between P and Q, as $P \xrightarrow{\bot} Q$. We can define the *co-continuous* functions dually. Now we introduce another important condition on functions which is additivity ([32]).

Definition 2.33 ((Completely) addittive function). *Let P and Q be two cpo. We say that a function $f : P \to Q$ is (completely) additive if for each subset X of P, we have that $f(\bigvee_P X) = \bigvee_Q f(X)$.*

Therefore, an additive function preserves limits (lub's) of all subsets of P (empty-set included). This means that all the additive functions are also continuous. The space of additive functions, ordered pointwise, is denoted by $(P \xrightarrow{\text{a}} Q, \sqsubseteq)$. We can dually define *co-additive* functions, whose space is denoted by $(P \xrightarrow{\text{coa}} Q, \sqsubseteq)$.

Definition 2.34 (Join-uniform function). *[63] Let L be a complete lattice. A function $f : L \to L$ is* join-uniform *if for all $Y \subseteq C$ we have that the implication $(\exists \bar{x} \in Y.\, \forall y \in Y.\, f(y) = f(\bar{x})) \Rightarrow (f(\bigvee Y) = f(\bar{x}))$ holds.*

In other terms, f is join-uniform if it is additive for any family of elements for which f is constant. *Meet-uniformity* is dually defined.

Scott hierarchy

In the following, we will introduce some classical characterizations of complete lattices. First of all, we have to define an operation on elements of a complete lattice: the complement.

Definition 2.35 (Complement). *Let P be a poset with $\bot$ and $\top$. For each $x \in P$ we say that $y \in P$ is the complement of x if it satisfies the following condition:*

$$(C) \quad x \wedge y = \bot \text{ and } x \vee y = \top$$

A lattice where each element has a complement is called *complemented* lattice.

Definition 2.36 (Boolean algebra). *A boolean algebra (shortly Ba) is a complemented and* distributive *lattice P, namely such that for each $x, y, z \in P$:*

$$(D) \quad x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z).$$

A *complete boolean algebra* (shortly cBa) is a complete lattice that satisfies the conditions (C) and (D), in Def. 2.35 and 2.36, respectively. It is well known that (D) implies its dual condition, indeed each boolean algebra is isomorphic to its dual. Another notion of complementation is the pseudo-complement.

Definition 2.37 (Pseudo-complement). *Let L be a lattice, and let $x \in L$. We say that x^* is the pseudo-complement of x if the following condition holds:*

$$(PC) \quad x \wedge x^* = \bot \;\; e \;\; \forall y \in L\,.\, x \wedge y = \bot \;\Rightarrow\; y \leq x^*$$

Hence, the pseudo-complement of an element x is the greatest element, whose glb with x is the bottom ($\bot$). Note that the pseudo-complement has no condition on the lub operation. A lattice is *pseudo-complemented* if each element of the lattice has a pseudo-complement.
A lattice L is *relatively pseudo-complemented* if each pair of elements $x, y \in L$ has relative pseudo-complement $x * y$: $x \wedge x * y \leq_L y$, and, if for each $z \in L$ we have $x \wedge z \leq_L y$, then $z \leq_L x * y$. Birkhoff [14] proved that each relatively pseudo-complemented lattice is also distributive.

Definition 2.38 (Complete Heyting algebra). *A complete Heyting algebra (shortly cHa), is a complete lattice P completely distributive, namely that satisfies the following condition:*

$$(CD) \quad x \wedge \bigvee Y = \bigvee \left\{ \, x \wedge y \,\middle|\, y \in Y \, \right\}$$

for each $x \in P$ and $Y \subseteq P$.

A Heyting algebra is a lattice, possibly not complete, which satisfies the property (CD). This property can be also formalized in the following way:

Definition 2.39 (Heyting algebra). *A Heyting algebra (shortly Ha) is a lattice H such that*

$$\forall a, b \in H \, . \, \bigvee \left\{ \, x \in H \,\middle|\, a \wedge x \leq b \, \right\} \in H$$

Fixpoint theory

Let us consider the notion of *fixpoint*.

Definition 2.40 (Fixpoint). *Let P be a poset, and let $f : P \to P$ be a function on P. Then $x \in P$ is a fixpoint of f if $f(x) = x$. We will denote by* $\mathrm{Fix(f)} \stackrel{def}{=} \left\{ \, x \in P \,\middle|\, f(x) = x \, \right\}$ *the set of all the fixpoints of f.*

If $\leq_P$ is the partial order on P, we denote by *lfp(f)* the least fixpoint of f, namely the unique element $x \in \mathrm{Fix(f)}$ such that $\forall y \in \mathrm{Fix(f)} \, . x \leq_P y$. Dually we can define the notion of greatest fixpoint of f, i.e., *gfp(f)*.

Definition 2.41. *In the same hypothesis of Def. 2.40 we say that $x \in P$ is a* pre-fixpoint *of f if $x \leq_p f(x)$. We define* $\mathrm{Pre(f)} \stackrel{def}{=} \left\{ \, x \in P \,\middle|\, x \leq_P f(x) \, \right\}$. *We say that x is a* post-fixpoint *of f if $f(x) \leq_p x$, and we define the set* $\mathrm{Post(f)} \stackrel{def}{=} \left\{ \, x \in P \,\middle|\, f(x) \leq_P x \, \right\}$.

At this point we can conclude that

$$\mathit{Fix(f)} = \mathit{Pre(f)} \cap \mathit{Post(f)}$$

We are interested in constructive characterizations of fixpoints, in particular of the least and of the greatest fixpoint. For this reason, we recall the Tarski theorem [110].

Definition 2.42 (Upper transfinite iteration sequence). *Consider the complete lattice $\langle L, \leq_L, \vee, \wedge, \top, \bot \rangle$. Let $x \in L$ and let $f : L \to L$ be a monotone function. We define the upper transfinite iteration sequence of f, $\{f^\alpha(x)\}_{\alpha \in \mathbb{O}}$, starting from x, as*

$$\begin{cases} f^0(x) = x \\ f^{\alpha+1}(x) = f(f^\alpha(x)), \ \alpha \in \mathbb{O} \\ f^\beta(x) = \bigvee_{\alpha \leq \beta} f^\alpha(x), \ \textit{for each limit ordinal } \beta \end{cases}$$

Dually we can define the lower transfinite iteration sequence, where we use the operator *inf*. When all the iterations of f exist and are well defined, we say that the function is *iterable*.

Definition 2.43 (Stationary transfinite sequence). *Given a function f, we say that the sequence $\langle f^\alpha(x), \alpha \in \mathbb{O} \rangle$ is stationary if there exists an ordinal $\epsilon \in \mathbb{O}$ such that $\forall \beta \geq \epsilon$ we have $f^\beta(x) = f^\epsilon(x)$; in this case the limit of the sequence is $f^\epsilon(x)$, and ϵ is said* closure ordinal. *We denote as* luis(f)(x) *the limit of the upper stationary sequence of f starting from x. Analogously we denote by* llis(f)(x) *the limit of the lower stationary sequence of f starting from x.*

Note that, if f is continuous, then this sequence has limit with a finite closure ordinal. The following is the Tarski theorem.

Theorem 2.44. *[110] Let $\langle L, \leq_L, \vee, \wedge, \top, \bot \rangle$ be a complete lattice and $f : L \to L$ be a monotone function, then the set of all the fixpoints of f is a complete lattice such that the least and the greatest fixpoints are, respectively*

$$lfp(f) = \bigwedge \text{Post(f)} \quad e \quad gfp(f) = \bigvee \text{Pre(f)}$$

If f is continuous, then

$$lfp(f) = \bigvee_{n \leq \omega} f^n(\bot)$$

This theorem tells us that, if we have a continuous function on a complete lattice, then its least fixpoint is the limit (that is obtained with at most ω steps) of the iteration sequence obtained starting from the bottom of the lattice. Dually, starting from the top of the lattice, we can find the greatest fixpoint of a co-continuous function.

Theorem 2.45. *Let $f : L \to L$ be a co-continuous function, defined on the complete lattice $\langle L, \leq_L, \vee, \wedge, \top, \bot \rangle$, then*

$$gfp(f) = \bigwedge_{n \leq \omega} f^n(\top)$$

Therefore, this characterization of least [greatest] fixpoint requires continuity [co-continuity] of the considered function. This makes this characterization not-constructive since, for example, when the function f is continuous, then we have a

method for finding the least fixpoint, but not for finding the greatest one. In [30] it is described a constructive characterization of the Tarski theorem, which relax the continuity condition, but requires a transfinite iteration. This allows to constructively characterize the complete lattice of the fixpoints of a function defined on L. Note that, $\langle \mathrm{Fix}(f), \leq_L \rangle \subseteq \langle L, \leq_L \rangle$, where the operations, $\vee, \wedge, \top, \bot$ are the same as in L.

Theorem 2.46 ([30]). *In the hypotheses of Def. 2.42, the set of the fixpoint of f is a non-empty complete lattice ordered by $\leq_L$, with bottom* $\mathrm{luis(f)}(\bot)$ *and top* $\mathrm{llis(f)}(\top)$.

Therefore, the least and the greatest fixpoint of a monotone function can be found as limits of the transfinite iteration sequences, respectively, lower and upper, starting respectively from the bottom and from the top of the lattice. In particular, if the function is continuous [co-continuous] the least [greatest] fixpoint has a finite closure ordinal.

2.2 Abstract Interpretation

2.2.1 Abstract domains individually

In this section, we recall the Cousot and Cousot's definition of abstract domain. Let C be a complete lattice or cpo, playing the role of concrete domain, and A be the poset of abstract objects ordered by their precision: $x \leq_A y$ if x is more precise than y in describing a given computational property. The standard abstract interpretation framework is funded on the adjoint relation between the abstract and the concrete domain [28].

Galois connections

Let us introduce the typical tools used for defining abstract interpretations.

Definition 2.47 (Galois connection). *Let $(A, \leq_A)$ and $(C, \leq_C)$ be two posets, $\alpha : C \to A$ and $\gamma : A \to C$. We say that $\langle A, \alpha, \gamma, C \rangle$ is a Galois connection (or* adjunction*), shortly GC, if*

- *For each $a \in A$ and for each $c \in C$ we have that $\alpha(c) \leq_A a \;\Leftrightarrow\; c \leq_C \gamma(a)$.*

In this case we write GC

$$(C, \leq_C) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \leq_A).$$

When $\langle A, \alpha, \gamma, C \rangle$ is a Galois connection, then α [γ] is called *left adjoint* [*right adjoint*] of γ [α]. In general C is called *concrete domain*, A is called *abstract domain*, α is also called *abstraction* and γ is also called *concretization*. Moreover, the fact that the two functions are monotone means that, both the processes, of abstraction

and of concretization, preserve the relative precision relation. Namely, if a concrete element contains a greater amount of information if compared with another one, then also its abstraction has to contain a greater (or equal) amount of information. On the other hand, the condition $x \leq \gamma(y) \;\Leftrightarrow\; \alpha(x) \leq y$ [32] guarantees the existence of the best approximation $\alpha(x)$ of x
Galois connections have two important properties:

- The function $\gamma\alpha$ is *extensive*, i.e., $\forall c \in C \,.\, c \leq_C \gamma\alpha(c)$.
- The function $\alpha\gamma$ is *reductive*, i.e., $\forall a \in A \,.\, \alpha\gamma(a) \leq_A a$.

Therefore, we can observe that

$$\forall a \in A \,.\, \gamma\alpha\gamma(a) = \gamma(a)$$
$$\forall c \in C \,.\, \alpha\gamma\alpha(c) = \alpha(c)$$

The following results are proved in [32].

Proposition 2.48. *If* $(C, \leq_C) \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} (A, \leq_A)$ *and* $(C, \leq_C) \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} (A, \leq_A)$*, then* $\alpha_1 = \alpha_2$ *iff* $\gamma_1 = \gamma_2$.

The consequence of this fact is that we represent a Galois connection simply by giving its left adjoint or its right adjoint. In particular, each adjoint can be uniquely determined by using the other one in the following way:

Proposition 2.49. *If* $(C, \leq_C) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \leq_A)$*, then for each* $c \in C$ *we can define the function* $\alpha(c) = \bigwedge \left\{ \; a \in A \,\middle|\, c \leq_C \gamma(a) \; \right\}$. *While, for each* $a \in A$*, we can define the function* $\gamma(a) = \bigvee \left\{ \; c \in C \,\middle|\, \alpha(c) \leq_A a \; \right\}$.

This means that, α maps each element $c \in C$ in the smallest element in A whose image by γ is greater than c, as regards $\leq_C$. Viceversa, γ maps each element $a \in A$ in the greatest element in C whose image by α is lower than a, as regards $\leq_A$.

Theorem 2.50. $(C, \leq_C) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \leq_A)$ *iff* α *is an additive map iff* γ *is a co-additive map.*

This means that, whenever we have an additive [co-additive] function between two domains, we have a Galois connection between the two domains. More precisely, whenever we have an additive function f, we can always build a Galois connection by considering its right adjoint $f^+ \stackrel{\text{def}}{=} \lambda x. \bigvee \left\{ \; y \,\middle|\, f(y) \leq x \; \right\}$. On the other, hand whenever we have a co-additive function f, we can always build a Galois connection by considering its left adjoint $f^- \stackrel{\text{def}}{=} \lambda x. \bigwedge \left\{ \; y \,\middle|\, x \leq f(y) \; \right\}$. In this conditions, $(f^+)^- = (f^-)^+ = f$.

Definition 2.51 (Galois insertion). *If* $(C, \leq_C) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \leq_A)$ *is such that* $\alpha\gamma = \iota_A$*, then we have a Galois insertion, GI, and we write* $(C, \leq_C) \underset{\alpha}{\overset{\gamma}{\leftrightarrows\!\!\!\!\!\rightarrow}} (A, \leq_A)$.

Proposition 2.52. *Let* $(C, \leq_C) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \leq_A)$*, then the following assertions are equivalent:*

1. $(C, \leq_C) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \leq_A)$;
2. α *is onto;*
3. γ *is one-to-one.*

Analogously, if $\gamma\alpha = \iota_c$, we say that the connection is a *Galois projection* and we denote it as $(C, \leq_C) \underset{\alpha}{\overset{\gamma}{\twoheadleftarrow}} (A, \leq_A)$. In this case, it is possible to prove that α is one-to-one and that γ is onto. Note that a Galois insertion induces an order structure from the domain C to the domain A.

Proposition 2.53. *If* $\langle C, \leq_C \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle A, \leq_A \rangle$*, and the concrete domain* C *is a complete lattice, then also* A *is a complete lattice.*

We can always obtain a Galois insertion starting from a Galois connection. This process is called *reduction*. This consists in collecting together all the elements $a \in A$, that have the same image under γ. The following proposition shows how we can compose Galois connections.

Proposition 2.54. *If* $(C, \leq_C) \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} (B, \leq_B)$ *and* $(B, \leq_B) \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} (A, \leq_A)$*, then we can define a Galois connection in the following way:* $(C, \leq_C) \underset{\alpha_2\alpha_1}{\overset{\gamma_1\gamma_2}{\leftrightarrows}} (A, \leq_A)$.

Closure operators

Let us introduce the notion of closure operator.

Definition 2.55 (Upper closure operator). *A function* $\rho : P \to P$ *on a poset* $\langle P, \leq_P \rangle$*, is an upper closure operator (shortly* upper closure *or uco) if it satisfies the following conditions:*

(i) $\forall x \in P \,.\, x \leq_P \rho(x)$ *(Extensivity);*
(ii) $\forall x, y \in P \,.\, (x \leq_P y \;\Rightarrow\; \rho(x) \leq_P \rho(y))$ *(Monotonicity);*
(iii) $\forall x \in P \,.\, \rho\rho(x) = \rho(x)$ *(Idempotence).*

Viceversa, if ρ is reductive, i.e., $\forall x \in P \,.\, x \geq_P \rho(x)$, then it is called *lower closure* or lco.
Given a Galois connection $(C, \leq_C) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \leq_A)$, we can prove that the map $\gamma\alpha$ is an upper closure on C while $\alpha\gamma$ is a lower closure on A. This means that, while in the abstraction process it is allowed to lose information, this is not possible in the concretization process, hence we can say that, if $c \in C$, then $\alpha(c)$ is the most precise abstract element that represents c. Viceversa, the upper closure operators identifies Galois insertions. Moreover, Ward, in [115], provides a characterization of Galois insertions in terms of Moore families.

Theorem 2.56. *Let* A *and* C *two lattices, let* $\alpha : C \to A$ *and* $\gamma : A \to C$*. The following assertions are equivalent:*

- $C \underset{\gamma}{\overset{\alpha}{\twoheadleftarrow}} A$;
- A *is isomorphic to a Moore family of* C;

- *If ρ is an upper closure on C, and there exists an isomorphism $\iota : \rho(C) \to A$ (and therefore $\iota^{-1} : A \to \rho(C)$), then $C \underset{\iota\circ\rho}{\overset{\iota^{-1}}{\leftrightarrows}} A$.*

This means that each Galois insertion, and therefore each abstraction of C, can be, uniquely, associated with an upper closure operator on C. Hence, it is possible to describe abstract domains on C in terms of both Galois insertions and upper closure operators [31]. In particular, the formulation of abstract domains through upper closures is particularly convenient when reasoning about the properties of abstract domains independently from the representation of their objects, i.e., independently from the name of objects in A [31].
Note that $\langle \rho(C), \leq \rangle$, which is an abstraction of C by Theorem 2.56, is a complete meet sub-semi-lattice of C (i.e., $\wedge$ is its *glb*), but, in general, it is not a complete sub-lattice of C, since the *lub* in $\rho(C)$ — defined by $\lambda Y \subseteq \rho(C).\, \rho(\vee Y)$ — might be different from that in C. $\rho(C)$ is a complete sub-lattice of C (later called *disjunctive abstraction*) iff ρ is additive. Dual results hold for lower closures.

2.2.2 Abstract domains collectively

The *lattice $\mathfrak{L}_C$ of abstract interpretations* of C (cf. [28, Section 7] and [31, Section 8]), is the complete lattice of all possible abstract domains (modulo isomorphic representation of their objects) of the concrete domain C, and it is isomorphic to the lattice $uco(C)$ of all the upper closure operators on C. Indeed, if C is a complete lattice or a cpo, then

$$\langle uco(C), \sqsubseteq, \sqcup, \sqcap, \lambda x.\top, \lambda x.x \rangle$$

is a complete lattice [100, 115], where for every $\rho, \eta \in uco(C)$, $\{\rho_i\}_{i\in I} \subseteq uco(C)$ and $x \in C$:

- $\rho \sqsubseteq \eta$ iff $\forall y \in C.\, \rho(y) \leq \eta(y)$ iff $\eta(C) \subseteq \rho(C)$;
- $(\sqcap_{i\in I}\rho_i)(x) = \wedge_{i\in I}\rho_i(x)$;
- $(\sqcup_{i\in I}\rho_i)(x) = x \;\Leftrightarrow\; \forall i \in I.\, \rho_i(x) = x$;
- $\lambda x.\top$ is the top element, whereas $\lambda x.x$ is the bottom element.

The pointwise ordering on $uco(C)$ corresponds precisely to the standard ordering used to compare abstract domains with regard to their precision: A_1 is more precise than A_2 (i.e., A_2 is an abstraction of A_1) iff $A_1 \sqsubseteq A_2$ in $uco(C)$. Least upper bounds and greatest lower bounds, on $uco(C)$, have the following reading as operators on domains. Let $\{A_i\}_{i\in I} \subseteq uco(C)$: (i) $\sqcup_{i\in I}A_i$ is the most concrete among the domains in $\mathfrak{L}_C$ which are abstractions of all the A_i's, i.e., $\sqcup_{i\in I}A_i$ is the *least* (w.r.t. $\sqsubseteq$) *common abstraction* of all the A_i's; (ii) $\sqcap_{i\in I}A_i$ is (isomorphic to) the well-known *reduced product* (basically cartesian product plus reduction) of all the A_i's, or, equivalently, it is the most abstract among the domains in $\mathfrak{L}_C$ which are more concrete than every A_i. Let us remark that $\sqcap_{i\in I}A_i = \mathcal{M}(\cup_{i\in I}A_i)$.
Let us see now some properties of closure operators.

Proposition 2.57. *Let $\rho, \eta \in uco(C)$ and $Y \subseteq C$*

1. $\rho(\bigwedge \rho(Y)) = \bigwedge \rho(Y)$
2. $\rho(\bigvee Y) = \rho(\bigvee \rho(Y))$
3. $\eta \sqsubseteq \rho \;\Leftrightarrow\; \eta \circ \rho = \rho \;\Leftrightarrow\; \rho \circ \eta = \rho$
4. *([94])* $\rho \circ \eta \in uco(C) \;\Leftrightarrow\; \rho \circ \eta = \eta \circ \rho = \rho \sqcup \eta$

Some of the most important operations on upper closure operators are: Reduced product $\sqcap$ [23,31], $\sqcup$, *pseudo-complement* $\ominus$ [23,45,59] and *reduced (relative) power* $\longrightarrow$ [31, 64, 66].

Reduced product

A simple method used for composing closure is reduced product, which allows to build modular closures starting form simpler ones. We saw before that

$$(\sqcap_{i \in I} \rho_i)(x) = \bigwedge_{i \in I} \rho_i(x)$$

namely, the reduced product is the smallest Moore family containing the set of all the involved closures. In Fig. 2.1 we have two closures, *Sign* and *Par*, on $\wp(\mathbb{Z})$, which respectively consider the sign and the parity of integer numbers. In particular, $0+$ represents all the non-negative integers, $0-$ all the non-positive integers, *ev* all the even integers and *od* all the odd integers. By reduced product we obtain the smallest, namely the most abstract, closure which contains the cartesian product of the two closures.

Least upper bound of closures

The other operation in $uco(C)$ is the least upper bound:

$$(\sqcup_{i \in I} \rho_i)(C) = \bigcap_{i \in I} \rho_i(C)$$

Let us see an example in Fig. 2.2. In this case, we have two closures on $\wp(\mathbb{Z})$. The closure that we obtain as result is the set intersection of the two closures; the greatest, namely the most concrete, closure contained in both the closures.

2.2.3 Equivalence relations vs Closure operators

In this section, we recall that there exists an isomorphism between equivalence relations and upper closure operators [101], in particular, for each equivalence relation on the domain C, $\mathtt{R} \subseteq C \times C$, we can define an upper closure operator on $\wp(C)$, $Clo^{\mathtt{R}} \in uco(\wp(C))$, and viceversa, from each upper closure operator $\eta \in uco(\wp(C))$ we can define an equivalence relation $Rel^{\eta} \subseteq C \times C$.

Consider an upper closure operator $\eta \in uco(\wp(C))$, we can define a corresponding equivalence relation $Rel^{\eta} \subseteq C \times C$, in the following way:

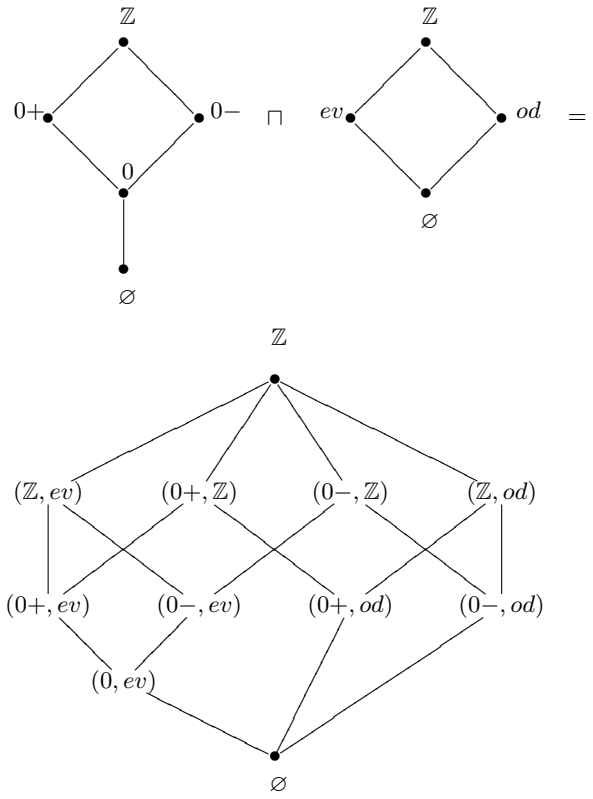


Fig. 2.1. Example of reduced product

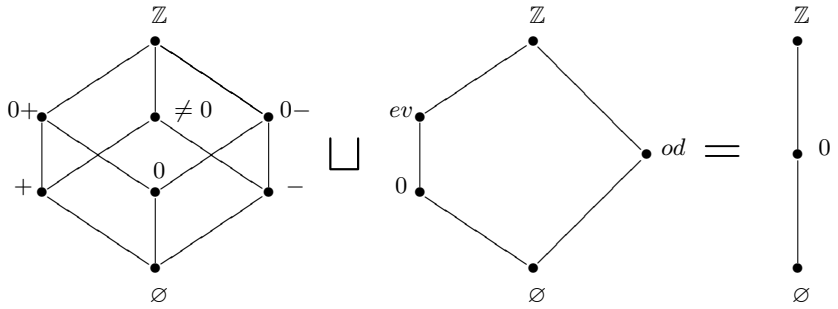


Fig. 2.2. Example of least upper bound of closures

$$\forall x, y \in C \,.\, x \;\; \mathit{Rel}^{\eta}\; y \;\Leftrightarrow\; \eta(\{x\}) = \eta(\{y\})$$

Proving that Rel^{η} is an equivalence relation is immediate and doesn't depend on the fact that η is an uco, but only on the fact that it is a function.

Consider now an equivalence relation $\mathtt{R} \subseteq C \times C$, we can define a corresponding upper closure operator $\mathit{Clo}^{\mathtt{R}} \in \mathit{uco}(\wp(C))$, in the following way:

$$\forall x \in C\,.\ \mathit{Clo}^{\mathtt{R}}(\{x\}) = [x]_{\mathtt{R}}$$
$$\forall X \subseteq C\,.\ \mathit{Clo}^{\mathtt{R}}(X) = \bigcup_{x \in X} [x]_{\mathtt{R}}$$

Namely, $\mathit{Clo}^{\mathtt{R}}$ is obtained by disjunctive completion of the equivalence classes induced by $\mathtt{R}$. Proving that $\mathit{Clo}^{\mathtt{R}}$ is an upper closure operator is immediate. In particular idempotence derives directly from the fact that $\mathtt{R}$ is an equivalence relation. In [101] the closure $\mathit{Clo}^{\mathtt{R}}$, as defined above, is identified as the most concrete upper closure operator inducing the same partition as the relation $\mathtt{R}$, and it is called *partitioning*. Let $\pi_{\mathtt{R}}$ the partition induced by $\mathtt{R}$, then we can rewrite the characterization of $\mathit{Clo}^{\mathtt{R}}$ in terms of abstract domain transformers:

$$\mathit{Clo}^{\mathtt{R}} = \bigtriangledown(\mathcal{M}(\pi_{\mathtt{R}})) = \bigtriangledown(\pi_{\mathtt{R}} \cup \{C, \varnothing\})$$

Given a generic closure η, we can always associate with it the partitioning closure, denoted by $\mathcal{P}(\eta)$, corresponding to the partition Rel^{η}, by using the isomorphism described before. Moreover, we have that $\eta = \mathcal{P}(\eta)$ iff η is closed under $\vee$ and $\neg$ [102], therefore the following result holds.

Proposition 2.58. *An upper closure operator $\eta \in \mathit{uco}(\wp(C))$ is partitioning, i.e., $\eta = \mathcal{P}(\eta)$, iff it is complemented, namely if $\forall X \in \eta.\ \overline{X} \stackrel{\mathit{def}}{=} C \smallsetminus X \in \eta$.*

Indeed, an upper closure operator η is always closed under glb, intersection in this context, therefore whenever it is closed also under complementation, we have that it is surely disjunctive, for the well known De Morgan laws. In the following we have an example of partitioning closure associated with a partition.

Example 2.59. Consider $\Sigma = \{1, 2, 3, 4\}$ and the partition $\pi = \{\{1\}, \{2, 3\}, \{4\}\}$, then the closure η with fix points $\{\varnothing, \{1\}, \{4\}, \{123\}, \Sigma\}$ induces exactly π as partition of states, but the (unique) most concrete closure that induces π is $\mathit{Clo}^{\pi} = \bigtriangledown(\{\varnothing, \{1\}, \{2, 3\}, \{4\}\}, \Sigma)$, which is exactly the closure on the right in Fig. 2.3.

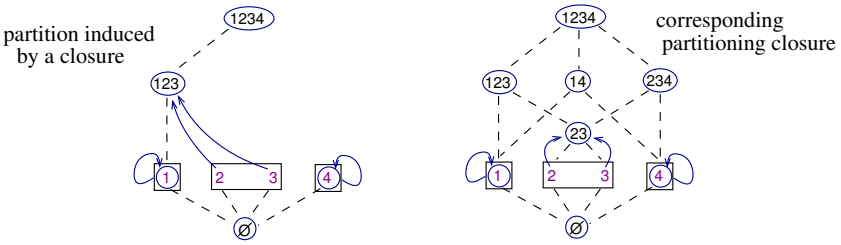


Fig. 2.3. A partitioning closure.

2.2.4 Abstract domain soundness and completeness

In this section, we introduce the notions of soundness and completeness of an abstract domain as regards a given concrete function or operation. We said, in the previous section, that the abstraction process introduces a loss of information, therefore it may be possible that what is computed in the concrete, does not coincide with what is computed in the abstract domain. Anyway, it is important that what holds in the concrete, holds also in the abstract process. On the contrary, we accept that in the abstract we have something more. This means that generally we require the *soundness* of the abstract domain, while we can relax the *completeness* requirement.

Soundness

In abstract interpretation there are two equivalent ways to express *soundness* of abstractions [31].

Definition 2.60. *Let $\langle A, \alpha, \gamma, C\rangle$ be a GI, let $f : C \to C$ be a function on the concrete domain, and let $f^\sharp : A \to A$ be its abstraction. We say that the abstraction is* sound *if*

$$\forall x \in C \,.\, \alpha \circ f(x) \leq_A f^\sharp \circ \alpha(x) \text{ or, equivalently } \forall x \in C \,.\, f \circ \gamma(x) \leq_A \gamma \circ f^\sharp(x)$$

This means that the abstract domain describes everything is described by the concrete, and possibly much more. In Fig. 2.4 we have a graphical representation of soundness. In particular, in Fig. 2.4(a) is represented the condition $\alpha \circ f(x) \leq_A f^\sharp \circ \alpha(x)$, which compares the computational processes in the abstract domain. In Fig. 2.4(b) is represented the condition $f \circ \gamma(x) \leq_C \gamma \circ f^\sharp(x)$, which compares the computational processes in the concrete domain.
We can note that there exists a particular abstract function $f^\sharp$ that guarantees soundness for f.

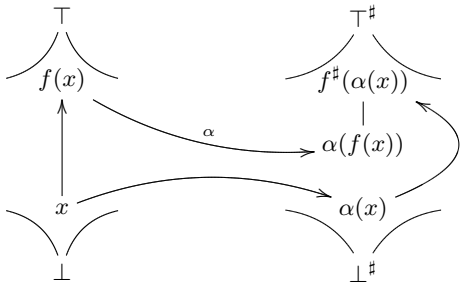
Theorem 2.61.

$$\forall x \in C \,.\, \alpha \circ f(x) \leq_A f^\sharp \circ \alpha(x) \;\Leftrightarrow \forall x \in C \,.\, \alpha \circ f \circ \gamma(x) \leq_A f^\sharp(x)$$

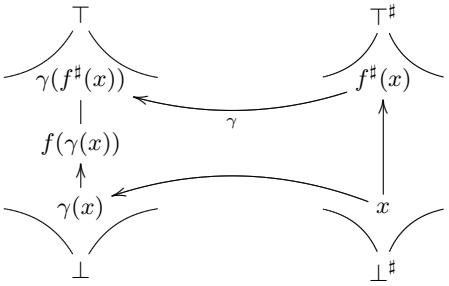
As a consequence, we have that $\alpha \circ f \circ \gamma : A \to A$ is the best correct approximation (*bca*) in A of $f : C \to C$ [31].

Completeness

Let us see in which conditions, the concrete and the abstract processes of calculus preserve the same precision. In particular, as we have done for soundness, we can think of comparing the two computational processes both in the abstract and in the concrete domain. But, while the two different characterizations of soundness are equivalent, they are not equivalent for completeness, namely when equality is required. The most known notion of completeness compares the computational results in the abstract domain and it is formalized as follows [31]:



(a)



(b)

Fig. 2.4. Soundness condition

Definition 2.62 (Complete abstraction). *Let $\langle A, \alpha, \gamma, C\rangle$ be a GI, and let $f : C \to C$ be a concrete function. We say that $f^\sharp : A \to A$ is complete for f, on the abstract domain $\alpha(C)$, if $\alpha \circ f = f^\sharp \circ \alpha$.*

This completeness require that in the abstract domain, the abstract and the concrete computations provide the same result. In the following we call this notion of completeness, *backward* completeness ($\mathcal{B}$-completeness). In Fig. 2.5 we have the graphical representation of this condition. This condition is not always true, for each possible abstraction α, but if there exists a complete function for f in $\alpha(C)$, then $\alpha \circ f \circ \gamma$ is also complete, and viceversa [31]. This means that it is possible to define a complete function for f in α, iff $\alpha \circ f \circ \gamma$ is complete. Moreover, since γ can be defined in terms of α, we can omit it, speaking of abstraction α complete for the concrete function f [65].

Lemma 2.63. *The function $\alpha \circ f \circ \gamma : A \to A$, is complete for f if and only if $\gamma \circ \alpha \circ f \circ \gamma \circ \alpha : C \to C$ is such that $\gamma \circ \alpha \circ f \circ \gamma \circ \alpha = \gamma \circ \alpha \circ f$.*

If we consider $\rho = \gamma \circ \alpha$, this condition can be rewritten as

$$\rho \circ f \circ \rho = \rho \circ f. \tag{2.1}$$

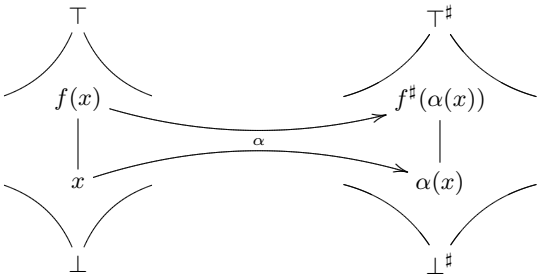


Fig. 2.5. Backward completeness condition

On the other hand, if we decide to compare the two computations in the concrete domain, then we obtain another notion of completeness that we call *forward* completeness ($\mathcal{F}$-completeness). In this case completeness holds iff $f \circ\gamma(x) = \gamma \circ f^\sharp(x)$. In Fig. 2.6 we have the graphical representation of this condition. As before, we

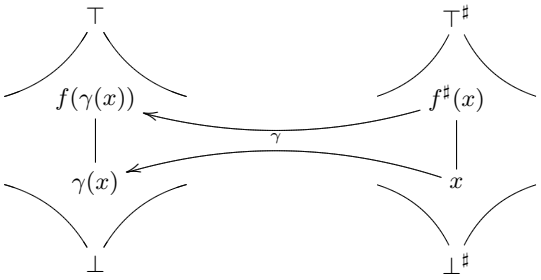


Fig. 2.6. Forward completeness condition

can make this definition independent from the concretization and from the specific abstract function, by considering the corresponding closure operator and the bca of the concrete function f. In this way we can formalize forward completeness as

$$\rho \circ f \circ \rho = f \circ \rho. \tag{2.2}$$

While $\mathcal{B}$-completeness is well known in abstract interpretation, and corresponds to the standard notion of completeness [65, 95], the notion of $\mathcal{F}$-completeness is less known. $\mathcal{B}$-completeness means that the domain ρ is expressive enough such that no loss of precision is accumulated by abstracting in ρ the arguments of f. Conversely, $\mathcal{F}$-completeness means that no loss of precision is accumulated by approximating the result of the function f computed in ρ. Clearly, when ρ is both $\mathcal{B}$ and $\mathcal{F}$ complete for f, then ρ is a morphism: $\rho \circ f = f \circ \rho$.

Completeness shells and core

In this section, we introduce a family of domain transformers that make an abstract domain complete. The problem of making abstract domains $\mathcal{B}$-complete has been solved in [65]. These results have been extended to $\mathcal{F}$-completeness in [60]. The key point in this construction is that there exists an either $\mathcal{B}$ or $\mathcal{F}$ complete abstract function $f^\sharp$ in an abstract domain A iff the best correct approximation $\alpha \circ f \circ \gamma$ of f in A is respectively either $\mathcal{B}$ or $\mathcal{F}$ complete. This means that both $\mathcal{F}$ and $\mathcal{B}$ completeness are properties of the underlying abstract domain A relatively to the concrete function f. These transformations are defined in terms of a function $f : C \to C$ on the concrete domain and they transform an abstract domain A, i.e., a closure operator, in order to make it complete for the given function f adding or erasing the smallest possible amount of information. We can obtain these transformers in two ways: by finding the most abstract domain that contains A and which is complete (*complete shell* of A); or by finding the most concrete domain contained in A and which is complete for f (*complete core* of A). These two methods are provided in [65] for backward completeness. The generalization to forward completeness is straightforward [60]. In a more general setting consider the function $f : C_1 \to C_2$ on the complete lattices C_1 and C_2, $\rho \in uco(C_2)$, and $\eta \in uco(C_1)$. $\langle \rho, \eta \rangle$ is a pair of $\mathcal{B}[\mathcal{F}]$-complete abstractions for f if $\rho \circ f = \rho \circ f \circ \eta$ $[f \circ \eta = \rho \circ f \circ \eta]$. Consider the following sets

$$\mathcal{F}(C_1, C_2, f) \stackrel{\text{def}}{=} \left\{ \langle \rho, \eta \rangle \,\middle|\, f \circ \eta = \rho \circ f \circ \eta \right\}$$
$$\mathcal{B}(C_1, C_2, f) \stackrel{\text{def}}{=} \left\{ \langle \rho, \eta \rangle \,\middle|\, \rho \circ f = \rho \circ f \circ \eta \right\}.$$

In particular, when $C_1 = C_2$, we write respectively $\mathcal{F}_R(C, f)$ and $\mathcal{B}_R(C, f)$. At this point, we consider domain transformations that allow to minimally transform any abstract domain A, not complete for f, in order to get completeness.

Definition 2.64. *Let $f : C \xrightarrow{m} C$ with C being a complete lattice. Let's define*

$$R_f^{\mathcal{B}} \stackrel{def}{=} \lambda\rho.\mathcal{M}(\textstyle\bigcup_{y \in \rho} \max(f^{-1}(\downarrow y))) \qquad C_f^{\mathcal{B}} \stackrel{def}{=} \lambda\rho.\left\{ y \in C \,\middle|\, \max(f^{-1}(\downarrow y)) \subseteq \rho \right\}$$

$$R_f^{\mathcal{F}} \stackrel{def}{=} \lambda\rho.\mathcal{M}(f(\rho)) \qquad C_f^{\mathcal{F}} \stackrel{def}{=} \lambda\rho.\left\{ y \in C \,\middle|\, f(y) \subseteq \rho \right\}$$

All these functions, $R_f^{\mathcal{B}}$ and $C_f^{\mathcal{B}}$, $R_f^{\mathcal{F}}$ and $C_f^{\mathcal{F}}$, are defined on $uco(C)$.

It is clear that $R_f^{\mathcal{B}}$ is monotone on $uco(C)$, because f is monotone in the complete lattice $\langle \wp(C), \subseteq \rangle$. Moreover, by definition, $R_f^{\mathcal{B}}(X) \sqsubseteq X$. The idea is that the inverse image of f contains all the elements that make a domain backward complete for f. On the other hand, also $R_f^{\mathcal{F}}$ is clearly monotone, and the idea is that the image of f contains all the elements that make a domain forward complete. Analogous reasonings can be done for the other functions.

Consider first the cases when $C_1 = C_2$, and $\eta = \rho$, then we want to find the shell and the core in $\mathcal{B}(C, f) \stackrel{\text{def}}{=} \left\{ \rho \,\middle|\, \rho \circ f = \rho \circ f \circ \rho \right\}$. Note that $\rho \in \mathcal{B}(C, f)$ iff $\rho \sqsubseteq R_f^{\mathcal{B}}(\rho)$, this is important because allows to build the complete domain as fixpoint. Then the following result holds.

Theorem 2.65. *[65] Consider $\rho \in uco(C)$ and suppose that it is not backward complete as regards the concrete function f. The backward complete shell of ρ is*

$$\mathcal{R}^{\mathcal{B}}_f(\rho) = gfp^{\sqsubseteq}_{\rho} \lambda\varphi.\rho \sqcap R^{\mathcal{B}}_f(\varphi)$$

While the backward complete core of ρ is

$$\mathcal{C}^{\mathcal{B}}_f(\rho) = lfp^{\sqsubseteq}_{\rho} \lambda\varphi.\rho \sqcup C^{\mathcal{B}}_f(\varphi)$$

where $\lambda\varphi.\rho \sqcup C^{\mathcal{B}}_f(\varphi) : uco(C) \to uco(C)$ and $\lambda\varphi.\rho \sqcap R^{\mathcal{B}}_f(\varphi) : uco(C) \to uco(C)$.

On the other hand, as far as the forward completeness is concerned, we look for shells and cores in the set $\mathcal{F}(C, f) \stackrel{\text{def}}{=} \{\ \rho \mid f \circ\rho = \rho \circ f \circ\rho\ \}$. Note that $\rho \in \mathcal{F}(C, f)$ iff $\rho \sqsubseteq R^{\mathcal{F}}_f(\rho)$, this is important because allows to build the complete domain as fixpoint, in particular we have the following result.

Theorem 2.66. *Consider $\rho \in uco(C)$ and suppose that it is not forward complete as regards the concrete function f. Then the forward complete shell of ρ is*

$$\mathcal{R}^{\mathcal{F}}_f(\rho) = gfp^{\sqsubseteq}_{\rho} \lambda\varphi.\rho \sqcap R^{\mathcal{F}}_f(\varphi)$$

While the forward complete core of ρ is

$$\mathcal{C}^{\mathcal{F}}_f(\rho) = lfp^{\sqsubseteq}_{\rho} \lambda\varphi.\rho \sqcup C^{\mathcal{F}}_f(\varphi)$$

where $\lambda\varphi.\rho \sqcup C^{\mathcal{F}}_f(\varphi) : uco(C) \longrightarrow uco(C)$ and $\lambda\varphi.\rho \sqcap R^{\mathcal{F}}_f(\varphi) : uco(C) \longrightarrow uco(C)$.

Consider $\ell \in \{\mathcal{B}, \mathcal{F}\}$. Note that $\mathcal{R}^{\ell}_f \in lco(uco(C))$ and $\mathcal{C}^{\ell}_f \in uco(uco(C))$ (see [65]). It is worth noting that ℓ-complete cores and shells are adjoint abstract domain transformers, i.e., adjoint functions on the lattice of abstract interpretations.

Theorem 2.67. *Let C_1, C_2 be complete lattices and $f : \wp(C_1) \xrightarrow{m} \wp(C_2)$. For any $\eta \in uco(C_1)$ and $\rho \in uco(C_2)$: $\mathcal{C}^{\ell}_f(\eta) \sqsubseteq \rho \Leftrightarrow \eta \sqsubseteq \mathcal{R}^{\ell}_f(\rho)$.*

In particular, it is possible to show that $C^{\ell}_f(\eta) \sqsubseteq \rho \Leftrightarrow \eta \sqsubseteq R^{\ell}_f(\rho)$.

Remark 2.68. If the function f is additive, then we can show that

$$max\{\ x \mid f(x) \leq y\ \} = \bigvee \{\ x \mid f(x) \leq y\ \} = f^{+},$$

namely f admits the right adjoint f^{+} [60]. This means that in this case we have

$$\mathcal{B}\text{-completeness for } f \quad \Leftrightarrow \quad \mathcal{F}\text{-completeness for } f^{+}$$

In [65] this notion of completeness is called *absolute*, while the most general case, in which we consider the sets $\mathcal{F}_R$ and $\mathcal{B}_R$ introduced before, is called *relative* completeness. At this point, we would like to characterize the shell and the core also for relative completeness [65]. By extending the observations made in [65] we have that the only interesting cases are the calculus of the *backward/forward*

complete core of ρ relative to η, and of the *backward/forward complete shell of η relative to ρ*. Let $\ell \in \{\mathcal{F}, \mathcal{B}\}$, we have:

$$\mathcal{C}_f^{\ell,\eta}(\rho) = \rho \sqcup C_f^\ell(\eta) \;\; \ell\text{-Complete core of } \rho \text{ relative to } \eta;$$
$$\mathcal{R}_f^{\ell,\rho}(\eta) = \eta \sqcap R_f^\ell(\rho) \; \ell\text{-Complete shell of } \eta \text{ relative to } \rho.$$

Finally, note that by the definitions of the functions C_f^ℓ and R_f^ℓ, we can prove the following relation:

$$\mathcal{C}_f^{\ell,\eta}(\rho) = \rho \;\Leftrightarrow\; \eta = \mathcal{R}_f^{\ell,\rho}(\eta) \tag{2.3}$$

Indeed, $\rho \sqcup C_f^\ell(\eta) = \rho$ iff $C_f^\ell(\eta) \sqsubseteq \rho$, $\eta \sqcap R_f^\ell(\rho) = \eta$ iff $R_f^\ell(\rho) \sqsupseteq \eta$, and we have seen above that $C_f^\ell(\eta) \sqsubseteq \rho$ iff $R_f^\ell(\rho) \sqsupseteq \eta$.

Examples of completeness transformers

Most domain refinements can be specified, as $\mathcal{F}$-complete refinements with respect to a given semantic operation. Intuitively, a domain refinement adds the functionalities of a given semantic operation of interest, namely the direct image of a function f. This corresponds to saying that a domain refinement can be specified as solution of a $\mathcal{F}$-completeness problem. In the following we consider as example the three basic operations for domain refinements originally introduced in [31], namely disjunctive completion, reduced product and reduced power of domains. A number of other operations can be specified in the same way as $\mathcal{F}$-completeness problems with respect to a given semantic operation. Clearly, whenever the operation f is additive, these characterizations all have an equivalent formulation in terms of standard completeness.

Reduced product and pattern completion:

Reduced product corresponds to the *glb* operation in $uco(C)$. Clearly any domain, being a Moore family, is $\mathcal{F}$-complete with respect to the binary and infinitary $\wedge$ operation. Therefore, it is immediate that the least $\mathcal{F}$-complete domain with respect to $\wedge$, that contains the domains A and B, is indeed $A \sqcap B$. A similar observation holds for patterns completion, like refinements introduced in [24], which are basically reduced product where one argument is fixed. The following result characterizes the pattern completion $\lambda X.\, A \sqcap X$ as a $\mathcal{F}$-completeness problem.

Theorem 2.69. *Let C be a complete lattice and $A, B \in uco(C)$. B is $\mathcal{F}$-complete for the function $\lambda X.\, \{a \wedge x \mid x \in X,\; a \in A\}$ iff $B \sqsubseteq A$.*

Disjunctive completion:

It is well known that a domain is disjunctive whenever the corresponding closure operation ρ is additive. Weaker forms of disjunctive completion are possible by allowing finite additivity. Therefore, the disjunctive completion can be formulated as follow [31, 61, 77]

$$\bigvee(X) = \bigsqcup \{\ Y \in uco(C) \,|\, Y \sqsubseteq X \ \wedge\ Y \text{ additive}\ \}$$

The following result characterizes the disjunctive completion as a $\mathcal{F}$-completeness problem with respect to disjunction:

Theorem 2.70. *Let C be a complete lattice and $\rho \in uco(C)$.*

1. *ρ is finitely disjunctive iff ρ is $\mathcal{F}$-complete for $\lambda X.\ \{\bigvee S \mid S \subseteq_{fin} X\}$;*
2. *ρ is disjunctive iff ρ is $\mathcal{F}$-complete for $\lambda X.\ \{\bigvee S \mid S \subseteq X\}$.*

Therefore, in both cases, the disjunctive completion $\bigvee(A)$ can be obtained by solving the recursive equation $X = A \sqcap \mathcal{R}^{\mathcal{F}}_{\vee}(X)$. A constructive characterization of this solution can be obtained for co-continuous lattices.

Proposition 2.71. *If C is a co-continuous lattice, then $\mathcal{R}^{\mathcal{F}}_{\vee}$ is co-continuous in $uco(C)$.*

Reduced power and Heyting completion:

The reduced power operation in [31] takes two input domains X and Y and returns the domain of monotone functions from X to Y, denoted $A \to B$. A logical model for this construction was studied in [66], under the hypothesis that C is a complete Heyting algebra. If $A, B \in uco(C)$ then

$$A \to B = \mathcal{M}(\{\ a \to b \,|\, a \in A, b \in B\ \})$$

where $a \to b = \vee \{\ x \,|\, a \wedge x \leq b\ \}$ [66].

Theorem 2.72. *Let C be a complete lattice and $A \in uco(C)$. A is $\mathcal{F}$-complete for $\to$ if and only if $\forall a, b \in A.\ a \to b \in A$.*

As observed in [66], the function $\lambda A.\ gfp(\lambda X.\ A \sqcap X \to X)$ is a domain refinement, which provides the Heyting completion of A. This operation can be defined as a $\mathcal{F}$-completeness problem: $gfp(\lambda X.\ A \sqcap X \to X)$ is the most abstract solution of the recursive domain equation $X = A \sqcap \mathcal{R}^{\mathcal{F}}_{f}(X)$ where the function is $f = \lambda X.\ \{x \to y \mid x, y \in X\}$.

3

A Geometry of Abstract Domain Transformers

Wisdom begins with wonder.
SOCRATE

Standard abstract interpretation provides advanced methods for the calculational design of static analyzers (see [26] for a fully detailed example) from a formally defined semantics of the programming language and from some given specifications on how semantics has to be approximated. An abstract interpreter can, therefore, be derived automatically from these two specifications. However, no such methodologies are known to provide an analogous calculational design of domain operations. Most of well known operations for refining and simplifying domains are in fact the result of either solutions to specific problems in refining or simplifying domains (viz. disjunctive completion [31,34], complete refinements and kernels [65], reduced power [31] and Heyting completion [66]) or inherited directly from the basic structure of the lattice of abstract interpretations (viz. reduced product [31], complementation [23]). The problem we want to attack in this chapter is that of studying a general framework for the calculational design of abstract domain operations, in such a way that systematic modification methodologies can be designed to modify generic abstract domains.

The main idea, for solving the problem of systematically design domain transformers, is to use the same abstract interpretation framework, but now lifted one level up: the object of discourse are domains instead of program state descriptions. A theory for the use of abstract interpretation to reason about abstract domain transformers is the main contribution of this chapter. The use of abstract interpretation in higher types, later called higher-order abstract interpretation will show the potentiality of abstract interpretation methods to fully design versatile abstract domain transformers.

In this chapter, we show that the standard Cousot and Cousot theory of abstract interpretation, based on the so called adjoint-framework of Galois connections, can be directly applied to reason about abstract domain operations, yet providing formal methodologies for the systematic design of abstract domain transformers. We first show that most domain transformers can be viewed as suitable problems of achieving precision with respect to some given semantic feature of the programming language we want to analyze. This observation has indeed an intuitive justification: the goal of refining a domain is always that of improving precision with respect to some basic semantic operation (e.g., arithmetic operations, unification in logic programs, data structure operations in simple and higher-order types). Analogously, simplifying domains corresponds to the dual operation of reducing precision with respect to analogous semantic operations. We show that most well known operations for transforming domains can be interpreted in this way and that the relation between refinement and simplification on domains is indeed an instance of the same abstract interpretation framework lifted to higher types, i.e., where the objects of abstraction/concretization are abstract domains.

3.1 Abstract interpretation in higher types

The notion of abstract domain refinement and simplification has been introduced in [44, 61] as a generalization of most well-known operations for transforming abstract domains. In this section, we consider these notions as instances of a more general pattern where abstract domain transformers have the same structure of abstract domains. In the sake of simplicity we consider unary functions only, even if all the following results can be easily generalized to generic n-ary functions. A domain transformer τ is any operator on $uco(C)$. If τ is monotone then it preserves the relative precision of the transformed domains. Following [61] we distinguish between:

Domain refinements: A domain refinement $\tau : uco(C) \to uco(C)$ is a function refining domains, i.e., $X \subseteq \tau(X)$.

Domain simplifications: A domain simplification $\tau : uco(C) \to uco(C)$ is a function simplifying domains, i.e., $\tau(X) \subseteq X$.

Monotone refinements and simplifications can be associated with closure operators: If τ is a monotone refinement or simplification then $\lambda x.\ gfp(\lambda y.\ x \sqcap \tau(y))$ and $\lambda x.\ lfp(\lambda y.\ x \sqcup \tau(y))$ are the corresponding idempotent refinements and simplifications [29]. Therefore, monotone refinements and simplifications may have the same structure of abstract domains, as closure operators on $uco(C)$, resp. $\tau \in lco(uco(C))$ and $\tau \in uco(uco(C))$. This key observation will be the basis in order to lift standard abstract interpretation in higher types, i.e., from a theory for approximating computational objects, such as semantics, to a theory for transforming abstract domains and domain transformers. We prove that Cousot

and Cousot's Galois connection based abstract interpretation theory is perfectly adequate to develop a theory of abstract domain transformers providing these transformations with the same calculational design techniques which are known for standard abstract interpretation.

As we recalled in Section 2.2.1, standard Galois connection-based abstract interpretation is based on the notion of adjointness. The following result, due to Janowitz [15, 76], characterizes the structure of adjoint closure operators.

Theorem 3.1 ([76]). *Let $\langle \tau, \tau^+ \rangle$ and $\langle \tau^-, \tau \rangle$ be pairs of adjoint closure operators on C. Then:*

$$(1)\quad \tau \in lco(C) \;\Leftrightarrow\; \tau^- \in uco(C) \;\Leftrightarrow\; \begin{cases} \tau^- \circ \tau = \tau \\ \tau \circ \tau^- = \tau^- \end{cases}$$

$$(2)\quad \tau \in lco(C) \;\Leftrightarrow\; \tau^+ \in uco(C) \;\Leftrightarrow\; \begin{cases} \tau^+ \circ \tau = \tau^+ \\ \tau \circ \tau^+ = \tau \end{cases}$$

Note that, saying that $\langle \tau, \tau^+ \rangle$ is a pair of adjoint operators, means that τ is additive, and that $\tau^+ = \lambda x.\ \bigvee \{\ y \,|\, \tau(y) \leq x\ \}$, while saying that $\langle \tau^-, \tau \rangle$ is a pair of adjoint operators, means that τ is co-additive, and that $\tau^- = \lambda x.\ \bigwedge \{\ y \,|\, x \leq \tau(y)\ \}$. Stated in our terms, this result says that any (either right or left) adjoint of a refinement (simplification) is a simplification (refinement). This means that for any refinement (simplification) we may have two possible simplifications (refinements) corresponding to either the right or the left adjoint, when one or both of them exist.

Let $\tau \in lco(uco(C))$ be a domain refinement. By Th. 3.1, if τ^- exists then:

$$\tau^-(\tau(X)) = \tau(X) \text{ and } \tau(\tau^-(X)) = \tau^-(X).$$

This means that τ^- is a simplification such that both τ and τ^- have the same sets of fixpoints, namely τ^- reduces any abstract domain X until the reduced domain Y satisfies $\tau(Y) = Y$. We call τ^- the *core* of τ and τ the *shell* of τ^-.

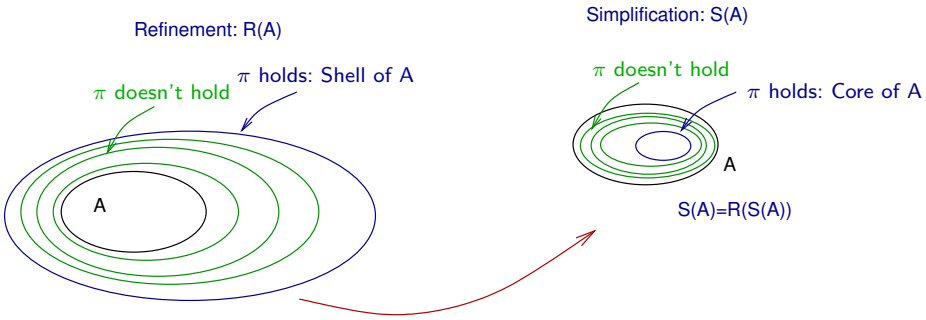


Fig. 3.1. Shells vs cores.

The following proposition, provides a further characterization of the adjoint closure operators as shell/core.

Proposition 3.2. *Let $\tau \in lco(C)$ and $\eta \in uco(C)$. If $\langle \tau^-, \tau \rangle$ and $\langle \eta, \eta^+ \rangle$ are pairs of adjoint operators, then*

$$\tau^- = \lambda x.\ \bigwedge \{\ \tau(y) \,|\, \tau(y) \geq x\ \}$$
$$\eta^+ = \lambda x.\ \bigvee \{\ \eta(y) \,|\, x \leq \eta(y)\ \}$$

Proof. Let us prove that $\bigwedge \{\ \tau(y) \,|\, \tau(y) \geq x\ \} = \bigwedge \{\ y \,|\, \tau(y) \geq x\ \}$ (the proof for η can be obtained by duality). We suppose that $\langle \tau^-, \tau \rangle$ is a pair of adjoint functions, namely that τ is co-additive. We prove that the two implications of equality separately. Note that

$$\begin{array}{l} \{\ \tau(y) \,|\, \tau(y) \geq x\ \} \subseteq \{\ y \,|\, \tau(y) \geq x\ \} \\ \qquad \Rightarrow\ \bigwedge \{\ \tau(y) \,|\, \tau(y) \geq x\ \} \geq \bigwedge \{\ y \,|\, \tau(y) \geq x\ \}. \end{array}$$

On the other hand, τ is co-additive, therefore

$$\begin{array}{l} \tau(\bigwedge \{\ y \,|\, \tau(y) \geq x\ \}) = \bigwedge \{\ \tau(y) \,|\, \tau(y) \geq x\ \} \geq x \\ \qquad \Rightarrow\ \tau(\bigwedge \{\ y \,|\, \tau(y) \geq x\ \}) \in \{\ \tau(y) \,|\, \tau(y) \geq x\ \} \end{array}$$

This means, since τ is reductive, that

$$\bigwedge \{\ y \,|\, \tau(y) \geq x\ \} \geq \tau(\bigwedge \{\ y \,|\, \tau(y) \geq x\ \}) \geq \bigwedge \{\ \tau(y) \,|\, \tau(y) \geq x\ \}.$$

In this way we proved the equality.

In particular, if $\pi \subseteq uco(C)$ is a $\bigsqcup$-closed property of abstract domains (where $\sqcup$ is the greatest lower bound of closures). It is clear that, if $C \in \pi$, then

$$\mathcal{R}_\pi \stackrel{\text{def}}{=} \lambda X \in uco(C).\ \bigsqcup \{\ Y \,|\, Y \in \pi \cap \downarrow X\ \} \in lco(uco(C))$$

$\mathcal{R}_\pi(X)$ is the most abstract domain refining X such that π holds (see Fig. 3.1). It is clear that $\mathcal{R}_\pi^-$ exists iff π is a complete sublattice of $uco(C)$. In this case, both $\mathcal{R}_\pi$ and $\mathcal{R}_\pi^-$ have the same set of fixpoints which is π, and

$$\mathcal{R}_\pi^- \stackrel{\text{def}}{=} \lambda X.\ \bigsqcap \{\ Y \,|\, Y \in \pi \cap \uparrow X\ \} \in uco(uco(C)).$$

The interpretation of the right adjoint of a refinement τ, when it exists, is quite different. By Th. 3.1, if τ^+ exists, then:

$$\tau^+(\tau(X)) = \tau^+(X) \text{ and } \tau(\tau^+(X)) = \tau(X).$$

In this case $\tau^+(X)$ is not a fixpoint of τ. Instead, it returns the most abstract domain whose precision can be lifted to that of X by refinement. The following proposition, provides a further characterization of the adjoint closure operators as expander/compressor.

Proposition 3.3. *Let $\tau \in lco(C)$ and $\eta \in uco(C)$. If $\langle \tau, \tau^+ \rangle$ and $\langle \eta^-, \eta \rangle$ are pairs of adjoint operators, then:*

$$\tau^+ = \lambda x.\ \bigvee \{\ y \,|\, \tau(y) = \tau(x)\ \}$$
$$\eta^- = \lambda x.\ \bigwedge \{\ y \,|\, \eta(y) = \eta(x)\ \}$$

Proof. Let us prove the result for τ, the other case is obtained by duality. Since we suppose that $\langle \tau, \tau^+ \rangle$ is a pair of adjoint functions, we are supposing that τ is additive. We prove the two implication of equality separately. First of all note that, since τ is reductive, i.e., $\tau(x) \leq x$, then

$$\{\ y \,|\, \tau(y) = \tau(x)\ \} \subseteq \{\ y \,|\, \tau(y) \leq x\ \}$$
$$\Rightarrow\ \bigvee \{\ y \,|\, \tau(y) = \tau(x)\ \} \leq \bigvee \{\ y \,|\, \tau(y) \leq x\ \}$$

On the other hand, since $\tau(x) \in \{\ \tau(y) \,|\, \tau(y) \leq x\ \}$, by additivity of τ, we have that

$$\tau(\bigvee \{\ y \,|\, \tau(y) \leq x\ \}) = \bigvee \{\ \tau(y) \,|\, \tau(y) \leq x\ \} \geq \tau(x)$$

Moreover, for each y, such that $\tau(y) \leq x$, we have $\tau(y) \leq \tau(x)$, by idempotence of τ, therefore

$$\tau(x) \geq \bigvee \{\ \tau(y) \,|\, \tau(y) \leq x\ \} \Rightarrow \tau(x) \geq \tau(\bigvee \{\ y \,|\, \tau(y) \leq x\ \})$$

Hence, $\tau(\bigvee \{\ y \,|\, \tau(y) \leq x\ \}) = \tau(x)$, i.e., $\bigvee \{\ y \,|\, \tau(y) \leq x\ \} \in \{\ y \,|\, \tau(y) = \tau(x)\ \}$, which implies

$$\bigvee \{\ y \,|\, \tau(y) = \tau(x)\ \} \geq \bigvee \{\ y \,|\, \tau(y) \leq x\ \}.$$

In this way we proved the equality.

Note that, τ^+ reduces any abstract domain X such that $\tau(X) = X$, towards the most abstract domain Y such that $\tau(Y) = X$. We call τ^+ the *compressor* of τ and τ the *expander* of τ^+. In this case, if $\pi \subseteq uco(C)$ is a $\bigsqcap$-closed property of abstract domains (where $\sqcap$ is the reduced product of closures) and $\{\top\} \in \pi$ then

$$\mathcal{R}_\pi \stackrel{\text{def}}{=} \lambda X.\ \bigsqcup \{\ Y \,|\, Y \in \pi \cap \downarrow X\ \} \in lco(uco(C)).$$

Let $\pi^\neg(X) \stackrel{\text{def}}{=} \{Y \mid \mathcal{R}_\pi(Y) = \mathcal{R}_\pi(X)\}$. If $\mathcal{R}_\pi$ is co-additive, we have

$$\mathcal{R}^+_\pi \stackrel{\text{def}}{=} \lambda X.\ \bigsqcup \{\ Y \,|\, Y \in \pi^\neg(X) \cap \uparrow X\ \} \in uco(uco(C))$$

The idea here is that $\mathcal{R}^+_\pi(X)$ refines the domain towards the domain which includes X and all those elements that $\mathcal{R}_\pi$ would weed out (see Fig. 3.2).

3.2 Reversible transformers

When a transformer (simplification or refinement) admits an adjoint (right or left), then we say that the refinement is *reversible*. Clearly, not all the domain

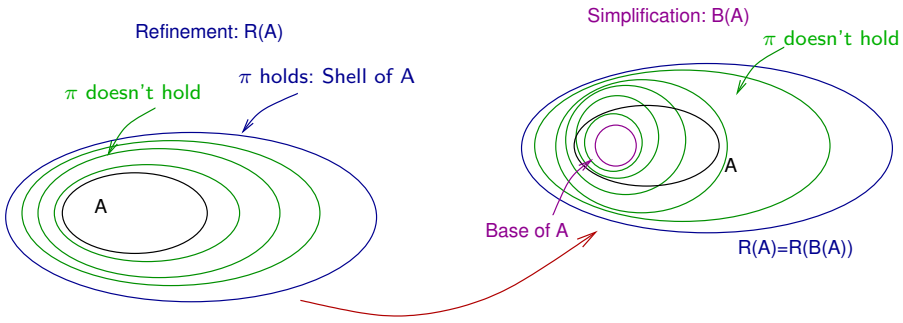


Fig. 3.2. Expander vs compressors.

transformers are reversible, because not all closures are either additive or co-additive functions. However, adjointness can be weakened by considering only those properties that make a transformer reversible, either as a pair shell/core or expander/compressor. In the following, we describe the properties of invertible refinements since the properties of invertible simplifications can be derived by duality as shown above.

3.2.1 Shell vs core

By Prop. 3.2, the relation between the shell τ and the core τ^- is characterized by the fact that $\tau^-(X)$ isolates the most concrete domain which is contained in X and which is a fixpoint of τ:

$$\tau^-(x) = \bigwedge \{\ \tau(y) \,|\, x \leq \tau(y)\ \}$$

While $\tau^- \circ \tau = \tau$ always holds for any $\tau \in \mathit{lco}(C)$, the key property, which characterizes the pair shell/core, is $\tau \circ \tau^- = \tau^-$ and it holds iff $\langle \tau^-, \tau \rangle$ is a pair of adjoint functions.

Theorem 3.4. *Let $\tau \in \mathit{lco}(C)$. Then $\tau \circ \tau^- = \tau^-$ holds iff τ is co-additive.*

Proof. We prove that $\forall x \in C.\ \tau(\bigwedge \{\ \tau(y) \,|\, \tau(y) \geq x\ \}) = \bigwedge \{\ \tau(y) \,|\, \tau(y) \geq x\ \}$ iff τ is co-additive. If τ is co-additive then

$$\tau(\bigwedge \{\ \tau(y) \,|\, \tau(y) \geq x\ \}) = \bigwedge \{\ \tau\tau(y) \,|\, \tau(y) \geq x\ \} = \bigwedge \{\ \tau(y) \,|\, \tau(y) \geq x\ \}$$

where the last equality holds by idempotence of τ.
Suppose, towards a contradiction, that the equality holds and that τ is not co-additive. If τ is not co-additive, then there exists $Z \subseteq \tau(C)$ such that $\tau(\bigwedge Z) \neq \bigwedge Z$. Let's prove that $\bigwedge Z = \bigwedge \{\ \tau(y) \,|\, \tau(y) \geq \bigwedge Z\ \}$. Note that $\bigwedge Z \leq \bigwedge \{\ \tau(y) \,|\, \tau(y) \geq \bigwedge Z\ \}$. On the other hand, it is worth noting that

$$\tau(y) \in Z \ \Rightarrow\ \tau(y) \geq \bigwedge Z \ \Rightarrow\ \tau(y) \in \{\ \tau(y) \,|\, \tau(y) \geq \bigwedge Z\ \}$$

Namely, $Z \subseteq \{\ \tau(y) \,|\, \tau(y) \geq \bigwedge Z\ \}$, therefore $\bigwedge Z \geq \bigwedge \{\ \tau(y) \,|\, \tau(y) \geq \bigwedge Z\ \}$. So we have the equality. Therefore we can conclude that

$$\tau(\bigwedge \{\ \tau(y) \,|\, \tau(y) \geq \bigwedge Z\ \}) = \tau(\bigwedge Z) \neq \bigwedge Z = \bigwedge \{\ \tau(y) \,|\, \tau(y) \geq \bigwedge Z\ \}$$

This means that the relation between shells and cores holds only in the standard adjoint framework.

3.2.2 Complete shell vs core

We've just introduced the general notion of core. In Sect. 2.2.4 we introduced the completeness core, which erases all the elements that make an abstract domain incomplete. We want to characterize, in the completeness framework, when a completeness core exists, namely for which conditions on f the completeness shell for f is reversible.
The following result characterizes the existence of $\mathcal{F}$ cores for a function $f : C \to C$.

Theorem 3.5. *Let C be a complete lattice and $f : C \to C$ be a monotone function. Then $\mathcal{R}^{\mathcal{F}}_f \in lco(uco(C))$ is co-additive iff f is co-additive.*

Proof. First of all note that, $\mathcal{R}^{\mathcal{F}}_f = \lambda X.\, \mathcal{M}(f(X))$ is co-additive iff the following equations hold:

$$\begin{aligned}\mathcal{M}(f(\sqcap_i X_i)) &= \sqcap_i \mathcal{M}(f(X_i)) = \mathcal{M}(\cup_i \mathcal{M}(f(X_i))) = \mathcal{M}(\cup_i f(X_i)) \\ &= \mathcal{M}(f(\cup_i X_i))\end{aligned}$$

This means that $\mathcal{R}^{\mathcal{F}}_f$ is co-additive iff $\mathcal{M}(f(\sqcap_i X_i)) = \mathcal{M}(f(\cup_i X_i))$. In particular, if we consider $\{x_i\}_{i\in\mathbb{N}} \subseteq C$, then we have $\mathcal{M}(f(\sqcap_i\{\top, x_i\})) = \mathcal{M}(f(\cup_i\{\top, x_i\}))$. Hence, being $f(\bigwedge_i x_i) \in \mathcal{M}(f(\sqcap_i\{\top, x_i\}))$, we have $f(\bigwedge_i x_i) \in \mathcal{M}(f(\cup_i\{\top, x_i\}))$. It is worth noting that the least element in $\mathcal{M}(f(\cup_i\{\top, x_i\}))$ is $\bigwedge_i f(x_i)$, therefore $f(\bigwedge_i x_i) \geq \bigwedge_i f(x_i)$. On the other hand,by monotonicity of f, we have that $f(\bigwedge_i x_i) \leq f(x_i)$ for each $i \in \mathbb{N}$, and therefore $f(\bigwedge_i x_i) \leq \bigwedge_i f(x_i)$, by well-known properties of greatest lower bound. So we have the equality and the co-additivity of f. On the other hand, if f is co-additive then it is simple to prove that also $\mathcal{R}^{\mathcal{F}}_f$ is co-additive.

In the following, we show some example of $\mathcal{F}$-complete cores associated with the standard domain refinements in [31].

Product core:

Reduced product always admits a corresponding core. We consider the core associated with the domain refinement $\lambda X.\, A \sqcap X$, with $A \in uco(C)$. In this case, by Theorem 3.1, the corresponding core is defined as follows:

$$\mathcal{C}^{\mathcal{F}}_{\wedge} = \lambda X.\ \sqcap \{\ Y \,|\, A \sqcap Y \subseteq X\ \} = \begin{cases} \{\top\} & X \not\sqsupseteq A \\ X & \text{otherwise} \end{cases}$$

Disjunctive core:

Let C be a Heyting algebra, i.e., a completely distributive lattice. This ensures that the semantic operation $\vee$ is co-additive. Then, by Theorem 3.5, the disjunctive completion $\curlyvee$ admits core $\mathcal{C}_{\vee}^{\mathcal{F}}$ which is, by Theorem 3.1 defined as follows:

$$\mathcal{C}_{\vee}^{\mathcal{F}} = \lambda X. \bigsqcap \{\, Y \,|\, \curlyvee(Y) \subseteq X \,\}.$$

Intuitively, $\mathcal{C}_{\vee}^{\mathcal{F}}(X)$ returns the most concrete disjunctive portion of X, namely the largest disjunctive domain contained in X.

3.2.3 Expander vs compressor

By Prop. 3.3, the relation between the expander τ and its compressor τ^+ is characterized by the fact that $\tau^+(x)$ is the most abstract domain which allows us to reconstruct $\tau(x)$ by refinement:

$$\tau^+(x) = \bigvee \{\, y \,|\, \tau(x) = \tau(y) \,\}$$

While $\tau^+ \circ \tau = \tau^+$ always holds for any $\tau \in \mathit{lco}(C)$, the key property which characterizes the pair expander/compressor, is $\tau \circ \tau^+ = \tau$ and it holds iff τ is join-uniform (see Def. 2.34). Join-uniformity captures precisely the intuitive insight of the pair expander/compressor. If τ is join-uniform, and $x \in C$, then there always exists a (unique) element $\bigvee Z$, such that $\tau(\bigvee Z) = \tau(x)$ when we have $Z = \{y \in C \mid \tau(x) = \tau(y)\}$. In this case, $\bigvee Z$ is the most abstract domain which leads to the same domain refinement as x does. As observed in [63], τ^+ may fail monotonicity. In [63] the authors proved that τ^+ is monotone on a lifted order induced by τ.

Definition 3.6. *[63] Let $\tau : C \xrightarrow{m} C$ and $\langle C, \leq \rangle$ be a complete lattice. The lifted order $\leq_\tau \subseteq C \times C$ is defined as follows:*

$$x \leq_\tau y \;\Leftrightarrow\; (\tau(x) \leq \tau(y)) \;\wedge\; (\tau(x) = \tau(y) \;\Rightarrow\; x \leq y)$$

The lifted $\leq_\tau$ is such that $\leq \Rightarrow \leq_\tau$. The following theorem strengthen [63, Th. 5.10][1] proving the equivalence between reversibility and adjointness in the lifted order for any lower closure.

Theorem 3.7. *Let $\tau \in \mathit{lco}(L)$ and $\tau^+ = \lambda x. \bigvee \{\, y \,|\, \tau(y) = \tau(x) \,\}$. The following facts are equivalent:*

1. *$\tau \circ \tau^+ = \tau$;*
2. *τ is join-uniform on $\leq$;*
3. *τ is additive on $\leq_\tau$ and the right adjoint of τ on $\leq_\tau$ is τ^+.*

[1] In [63, Th. 5.10] the authors proved only that 1. $\Leftrightarrow$ 2. $\Rightarrow$ 3.

Proof. Since τ is join uniform if $\tau(\bigvee \{\ y \,|\, \tau(y) = \tau(x)\ \}) = \tau(x)$, by definition, we have that the equivalence of the first two points is straightforward. Let's prove the equivalence between the second and the third point, namely τ join-uniform on $\leq$ if and only if τ is additive on $\leq_\tau$. The fact that τ is additive on the lifted order when it is join-uniform on the standard order comes by duality from [63, Th. 5.10]. Let us prove the other implication, and consider the definition of lifted least upper bound [63]

$$\bigvee_\tau Y = \begin{cases} \bigvee \{\ y \in Y \,|\, \tau(y) = \tau(x)\ \} & \text{if } \exists x \in Y \,.\, \bigvee \tau(Y) = \tau(x) \\ \bigvee \tau(Y) & \text{otherwise} \end{cases}$$

Note that $x \in \{\ y \,|\, \tau(y) = \tau(x)\ \}$, and that $\bigvee \{\ \tau(y) \,|\, \tau(y) = \tau(x)\ \} = \tau(x)$, therefore

$$\begin{aligned} \textstyle\bigvee_\tau \{\ y \,|\, \tau(y) = \tau(x)\ \} &= \bigvee \{\ y \in \{\ z \,|\, \tau(z) = \tau(x)\ \} \,|\, \tau(y) = \tau(x)\ \} \\ &= \bigvee \{\ y \,|\, \tau(y) = \tau(x)\ \} \end{aligned}$$

Hence we have

$$\begin{aligned} \tau(\bigvee \{\ y \,|\, \tau(y) = \tau(x)\ \}) &= \tau(\textstyle\bigvee_\tau \{\ y \,|\, \tau(y) = \tau(x)\ \}) \\ &= \textstyle\bigvee_\tau \{\ \tau(y) \,|\, \tau(y) = \tau(x)\ \} = \tau(x) \end{aligned}$$

This is join-uniformity of τ on the order $\leq$. Let us prove that $\bigvee_\tau \{\ y \,|\, \tau(y) \leq_\tau x\ \}$ is the right adjoint of τ in the order $\leq_\tau$ is τ^+ when τ is join-uniform. Note that

$$\begin{aligned} \{\ y \,|\, \tau(y) \leq_\tau x\ \} &= \{\ y \,|\, \tau(y) \leq \tau(x), \tau(y) = \tau(x) \ \Rightarrow\ \tau(y) \leq x\ \} \\ &= \{\ y \,|\, \tau(y) \leq \tau(x)\ \} \end{aligned}$$

by reductivity of τ. This implies that $\bigvee_\tau \{\ y \,|\, \tau(y) \leq_\tau x\ \} = \bigvee_\tau \{\ y \,|\, \tau(y) \leq \tau(x)\ \}$ holds. Consider the definition of least upper bound given above and note that $x \in \{\ y \,|\, \tau(y) \leq \tau(x)\ \}$, and that $\bigvee \{\ \tau(y) \,|\, \tau(y) \leq \tau(x)\ \} = \tau(x)$, therefore

$$\begin{aligned} \textstyle\bigvee_\tau \{\ y \,|\, \tau(y) \leq_\tau x\ \} &= \textstyle\bigvee_\tau \{\ y \,|\, \tau(y) \leq \tau(x)\ \} = \bigvee \{\ y \,|\, \tau(y) \leq \tau(x), \tau(y) = \tau(x)\ \} \\ &= \bigvee \{\ y \,|\, \tau(y) = \tau(x)\ \} \end{aligned}$$

The relation between join-uniformity and meet-uniformity is preserved by the relation of adjointness on the standard order.

Proposition 3.8. *Let $\tau \in lco(L)$ be a join-uniform operator on $\leq$. Then we have $\tau^+(x) = \bigvee\{y | \tau(x) = \tau(y)\}$ is meet-uniform on $\leq$.*

Proof. First of all we can note that, by Theorem 3.7, τ join-uniform on $\leq$ implies τ additive on the lifted order $\leq_\tau$ that implies τ^+ co-additive on this order. Note that

$$\tau(\textstyle\bigwedge \{\ y \,|\, \tau^+(y) = \tau^+(x)\ \}) = \tau(\bigwedge_\tau \{\ y \,|\, \tau^+(y) = \tau^+(x)\ \})$$

by the dual of [63, Lemma 5.6]. For the properties of adjoint closures this implies that

$$\begin{aligned}\tau\tau^+(\bigwedge\{\ y \mid \tau^+(y)=\tau^+(x)\ \}) &= \tau\tau^+(\bigwedge_\tau\{\ y \mid \tau^+(y)=\tau^+(x)\ \})\\ &= \tau(\bigwedge_\tau\{\ \tau^+(y) \mid \tau^+(y)=\tau^+(x)\ \})\\ &\quad \text{(by co-additivity of } \tau^+ \text{ on the lifted order)}\\ &= \tau\tau^+(x)\end{aligned}$$

Therefore, $\tau(\bigwedge\{\ y \mid \tau^+(y)=\tau^+(x)\ \}) = \tau(x)$ which implies that

$$\tau^+\tau(\bigwedge\{\ y \mid \tau^+(y)=\tau^+(x)\ \}) = \tau^+\tau(x)$$

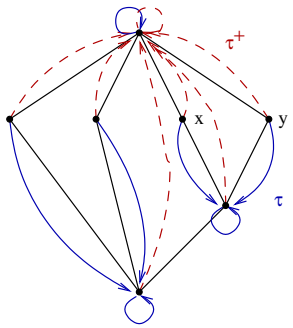
and, by the equations of adjointness between closures, this implies that

$$\tau^+(\bigwedge\{\ y \mid \tau^+(y)=\tau^+(x)\ \}) = \tau^+(x)$$

namely τ^+ is meet-uniform on the order $\leq$.

Unfortunately, the inverse implication of Prop. 3.8 does not hold in general, as we can see in the following example.

Example 3.9. In the figure below, we provide an example where the map τ^+ is meet-uniform, while τ is not join-uniform.



Indeed, note that $\tau^+ = \lambda X.\ \top$, which is clearly meet-uniform, while τ is not join-uniform since, for instance, $\tau(x) = \tau(y) \neq \top$, but $\tau(x \vee y) = \tau(\top) = \top$.

3.2.4 Complete expansion vs compression

In this section, we want to characterize, in the completeness framework, when a completeness compression exists. Namely, which conditions on f make the completeness refinement, for f, right reversible. Unfortunately, we have not found an elegant characterization of the functions that make the completeness refinement right reversible. Anyway, we derived a method for checking, given a function and a domain, whether the domain admits a base for the completeness refinement. We call *base* of ρ the result of the compression, when it exists, i.e., the most abstract domain, contained in ρ, that have the same refinement as ρ. In particular, our method is relative to the starting closure. This means that it allows to find the base of a given domain, when it exists, but it cannot prove that the completeness

refinement is right reversible. We can only say, whenever the given domain does not admit the base, that the corresponding refinement is not right reversible, i.e., not join uniform. Indeed, we can define a notion of *relative join-uniformity*. We say that a function $f : C \to C$ is join-uniform relatively to an element $x \in C$ if $f(\bigvee \{\ y \in C \,|\, f(y) = f(x)\ \}) = f(x)$. Clearly, if the refinement is join-uniform relatively to an element, it is not necessarily join-uniform, on the other hand if we find an element for which the refinement is not relative join-uniform, then it cannot be join-uniform.

As we have seen in the previous section, if $\mathcal{R}$ is join-uniform, then there exists a corresponding compressor $\mathcal{K}$. This means that for each closure operator $\rho \in uco(C)$ we can find its *base* $\mathcal{K}(\rho) \in uco(C)$ with the same refined domain $\mathcal{R}(\rho)$. Here, we provide a systematic method for simplifying finite abstract domains in order to isolate the most abstract domain, when it exists, whose refinement towards completeness for a given semantic function, $f : C \to C$, returns a given domain, here called *base* [50]. In the following, we would like to give some characterizations of join-uniform, namely right-reversible, completeness transformers, in terms of the function f for which we want completeness. We start from the consideration that *irreducible* elements play a key role in designing domain compressors.

Definition 3.10. *Consider the function $f : C \to C$, the set of f-reducible elements is $\dot{f}(C) \stackrel{def}{=} \{\ x \in C \,|\, \exists y \in C \smallsetminus \{x\}\ .\ f(y) = x\ \}$. The set of f-irreducible elements is defined as $firr(C) \stackrel{def}{=} C \smallsetminus \dot{f}(C)$.*

The idea is that x is f-reducible if x can be generated from elements which are different from x. Consider the sequence obtained by iterating a function f: $f^1(x) \stackrel{\text{def}}{=} f(x)$ and $f^{n+1}(x) \stackrel{\text{def}}{=} f(f^n(x))$. For $x \in C$ define $f^*(x) = Z \subseteq C$ if $\forall z \in Z\ .\ \exists n \in \mathbb{N}\ .\ f^n(x) = z$. The following proposition relates the lack of join-uniformity to cycles via f in C. Note that, in sake of simplicity, in the following of this section, we will denote by $\mathcal{R}_f$ the forward completeness refinement $\mathcal{R}_f^{\mathcal{F}}$.

Proposition 3.11. *Let C a complete lattice and let $Mirr(C) \cap \dot{f}(C) = X \neq \varnothing$. If $f : C \xrightarrow{m} C$ and there exists $Y \subseteq X$ and $x_1, x_2 \in X$ such that $x_1 \neq x_2$ and $f^*(x_1) = Y = f^*(x_2)$ then $\mathcal{R}_f$ is not join-uniform.*

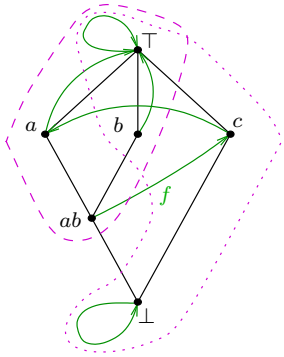
Proof. Consider the two trivial closures:

$$Z_1 \stackrel{\text{def}}{=} \{\top, x_1\} \in uco(C)$$
$$Z_2 \stackrel{\text{def}}{=} \{\top, x_2\} \in uco(C)$$

Then $Z_1 \sqcup Z_2 = \{\top\} \in uco(C)$. It is clear that $\mathcal{R}_f(\{\top\}) = \{\top\}$, while $\mathcal{R}_f(Z_1) = \mathcal{M}(Y) = \mathcal{R}_f(Z_2) \neq \{\top\}$. Namely $\mathcal{R}_f$ is not join-uniform.

Next example shows that, in general, the inverse of Proposition 3.11 doesn't hold.

Example 3.12. Consider the lattice depicted below, and f as drawn in the picture:



Consider the closure operator $\{\top, a, b, ab\}$ (represented with a dashed line) and the operator $\{\top, b, c, \bot\}$ (represented with a dotted line). Both these closures are such that their refinements $\mathcal{R}_f$ give back the whole lattice, while their intersection $\{\top, b\}$ is still complete but different from the whole lattice. In this case $\mathcal{R}_f$ is not join-uniform and the hypotheses of Proposition 3.11 are not satisfied.

At this point, we would like to formalize a method consisting in an algorithm that allows to characterize the base of any join-uniform completeness refinement. The algorithm that we are going to describe [50], given a closure ρ, finds the most abstract domain that generates $\mathcal{R}_f(\rho)$, when it exists. This is exactly the base of ρ if the completeness refinement is join-uniform. The key idea is that each element of the forward completeness refinement $\mathcal{R}_f(\rho)$ can be generated, starting from ρ, by meet or by f. In the following, we give an intuitive idea of the algorithm, whose aim is that of finding the most abstract domain ρ', contained in ρ, able to generate $\mathcal{R}_f(\rho)$ by completeness refinement, i.e., such that $\mathcal{R}_f(\rho) = \mathcal{R}_f(\rho')$.

1. Let us consider all the domains, more abstract than ρ, that *cover* ρ, i.e., all the domains in the filter of ρ, obtained by erasing only one element from ρ[2];
2. For each one of these elements we check if it can generate $\mathcal{R}_f(\rho)$. If none is able to generate this domain, then ρ was minimal in the set of all the domains that can generate $\mathcal{R}_f(\rho)$ by forward completeness refinement, and we put it in a set denoted $\mathcal{P}(\rho)$. Otherwise we repeat the same step for all the domains that can generate $\mathcal{R}_f(\rho)$.
3. When there are no more domains for which to check the generation of $\mathcal{R}_f(\rho)$, we check the cardinality of $\mathcal{P}(\rho)$. If $|\mathcal{P}(\rho)| = 1$, then it contains the most abstract domain generating $\mathcal{R}_f(\rho)$, which is base of ρ when the refinement is join-uniform. Otherwise the refinement cannot be join-uniform.

Note that, we always specify that we find the base only when the refinement is join-uniform, this is due to the fact that we can have $|\mathcal{P}(\rho)| = 1$ even if the completeness refinement is not join-uniform. Indeed, as we have said before, with

[2] Clearly, we can obtain these domains only by erasing elements that cannot be generated from the others by meet, since the resulting domains have to be Moore families.

this method we can only check *relative join-uniformity*. Then we can say that if $|\mathcal{P}(\rho)| = 1$, then the refinement is join-uniform relatively to ρ, but not necessarily join-uniform.

At this point, we can introduce the basic tools used in the algorithm. Hence, we consider the domain $\rho \in uco(C)$, to be compressed, and we consider a *candidate base* $\delta \in uco(C)$, which is the domain that represents the domain, candidate to be the base of ρ. In order to model join-uniformity, we need first to model when a set of objects can be generated from a candidate base by domain refinement $\mathcal{R}_f$.

Definition 3.13. *Let C be a finite lattice, $X \subseteq C$, $x \in C$, and $\delta \in uco(C)$. Consider $f : C \to C$, then $G(x) \stackrel{def}{=} min\left\{\ Z \subseteq C \,\middle|\, x \notin Z,\ (\bigwedge Z = x \ \vee\ f(Z) = x)\ \right\}$[3].*
In any case, $\overset{\frown}{X}(\delta) \stackrel{def}{=} \left\{ \bigcup_{x\in X} Y_x \,\middle|\, \begin{array}{l}((G(x) \neq \varnothing \ \wedge\ x \notin \delta) \ \Rightarrow\ Y_x \in G(x)),\\ ((G(x) = \varnothing \ \vee\ x \in \delta) \ \Rightarrow\ Y_x = \{x\})\end{array} \right\}$;

In the definition above, $G(x)$ is the set of all minimal (viz., non-redundant) sets of elements that do not include x and generate x by either meet or f. $\overset{\frown}{X}(\delta)$ is the collection of all the sets that can generate X by one step of the completeness refinement assuming δ to be the candidate base. This means that we have to find only the elements generating $X \smallsetminus \delta$. In order to design a method for filtering out those objects that can be generated by domain refinement, and therefore that are not in the base, we design a tree-like structure, where the descendant nodes of the tree are sets of objects from which the ancestor can be derived by refinement. Given an abstract domain $\delta \in uco(C)$, we introduce a binary relation $\rightarrow_\delta \subseteq \wp(C) \times \wp(C)$ such that $X \rightarrow_\delta Y$ if $Y \in \overset{\frown}{X}(\delta)$ (i.e. $X \subseteq \mathcal{R}_f(Y)$). At this point, we define the tree that we are going to use in the following. The definition is by induction and the tree has finite depth. In order to guarantee this fact we need to avoid infinite chains of generation by f or by meet. For this reason we consider finite domains.

Definition 3.14. *Let $\delta \in uco(C)$ and $x \in C$.*

- *$\Gamma_x(\delta)$: x is the root;*

$$\begin{cases} x \rightarrowtail \ \textsc{And}_Y\ Y \in G(x) \ \wedge\ x \notin \delta \ \wedge\ \forall y \in C\,.\,\exists m \in \mathbb{N}\,.\, y \rightarrowtail^m x \Rightarrow x \neq y \\ x \not\rightarrowtail \qquad\qquad otherwise \end{cases}$$

  $\forall\ \textsc{And}_Y \in \Gamma_x(\delta)\, \forall y \in Y\ .\ \ \textsc{And}_Y \rightarrowtail \Gamma_y(\delta)$.
- *Leaves of $\Gamma_x(\delta)$:*

$$\begin{cases} \mathcal{L}_x(\delta) = \{\{x\}\} \qquad \text{if in } \Gamma_x(\delta)\ x \not\rightarrowtail \\ \mathcal{L}_x(\delta) = \bigcup \left\{\ \dot{\bigcup} \left\{\ \mathcal{L}_y(\delta) \,\middle|\, x \rightarrowtail\ \textsc{And}_Y \rightarrowtail y\ \right\} \,\middle|\, Y \in G(x)\ \right\} \end{cases}$$

  where for each $X, Y \subseteq \wp(C)$ we have $X \dot{\cup} Y = \left\{\ X' \cup Y' \,\middle|\, X' \in X,\ Y' \in Y\ \right\}$.

[3] We abuse notation by denoting with f also its additive lift to sets $f(X) \stackrel{\text{def}}{=} \bigcup_{x\in X} f(x)$.

We abuse notation by letting $\Gamma_x(\delta)$ to represent this tree. A path in the tree from the root x to X represents a sequence of different sets of objects that, by iteratively applying R_f, may generate x. Note that X is not a leaf in the tree (i.e., $\exists Y \,.\, X \rightarrowtail Y$) if $X \not\subseteq \delta$ can be generated by $\mathcal{R}_f$ from a different set and there is no cycle, in the path from the root to X.

Remark 3.15. Note that, if C is a finite lattice and $\rho \in uco(C)$, then for each $x \in C$ the tree $\Gamma_x(\rho)$ is finite by construction. Indeed there is a check on the existence of cycles in $\Gamma_x(\delta)$, which avoids repetitions in the tree. Moreover, for each node X which is not a leaf, i.e., such that $\overset{\frown}{X}(\delta) \neq \{X\}$, $X \not\subseteq \delta$, whenever $\Gamma_x(\delta)$ is without cycles, we have that $\overset{\frown}{X}(\delta)$ contains each possible minimal set, different from X, that can generate, by either meet or f, all the elements of X that are not in δ.

In the following, we consider finite lattices C and monotone functions $f : C \to C$. Next proposition proves the correspondence between the tree construction and the generation of objects in the refined domain.

Theorem 3.16. *Let C be a finite lattice, $\delta \in uco(C)$ and $x \in C$. Then:*

(1) $Y \in \mathcal{L}_x(\delta) \;\Rightarrow\; x \in \mathcal{R}_f(Y)$ *and*
(2) $x \in \mathcal{R}_f(\delta) \;\Rightarrow\; \exists Y \in \mathcal{L}_x(\delta) \,.\, Y \subseteq \delta$.

Proof. We prove the two points separately.
(1) We have to prove that if $Y \in \mathcal{L}_x(\delta)$ then $x \in \mathcal{R}_f(Y)$. In order to obtain this we have to define the notion of depth of the tree $\Gamma_x(\delta)$.

$$\begin{cases} \mathcal{D}_x(\delta) = 0 & \text{if in } \Gamma_x(\delta)\ x \not\rightarrowtail \\ \mathcal{D}_x(\delta) = \max \left\{ \; 1 + \mathcal{D}_y(\delta) \;\middle|\; \begin{array}{l} \exists Y \in G(x), y \in Y \,. \\ x \rightarrowtail \ \textsc{And}_Y \rightarrowtail y \end{array} \right\} \end{cases}$$

We prove the thesis by induction on the depth $\mathcal{D}_x(\delta)$ of the tree $\Gamma_x(\delta)$.

Base: Consider $\mathcal{D}_x(\delta) = 1$, namely for each $y \in Y \in G(x)$ we have $x \rightarrowtail \textsc{And}_Y \rightarrowtail y \not\rightarrowtail$ in $\Gamma_x(\delta)$. Therefore, by definition of the set of leaves we have:

$$\begin{aligned} \mathcal{L}_x(\delta) &= \bigcup \left\{ \; \dot{\bigcup} \{ \; \mathcal{L}_y(\delta) \,|\, x \rightarrowtail \ \textsc{And}_Y \rightarrowtail y \; \} \,|\, Y \in G(x) \right\} \\ &= \bigcup \left\{ \; \dot{\bigcup} \{ \; \{\{y\}\} \,|\, x \rightarrowtail \ \textsc{And}_Y \rightarrowtail y \; \} \,|\, Y \in G(x) \right\} \\ &= \bigcup \{ \; \{Y\} \,|\, Y \in G(x) \; \} = G(x) \end{aligned}$$

So we have $Y \in \mathcal{L}_x(\delta) \;\Rightarrow\; Y \in G(x) \;\Rightarrow\; x \in \mathcal{R}_f(Y)$ by definition of $G(x)$.

Inductive Step: Suppose that, if $\mathcal{D}_x(\delta) = n$ then the thesis holds. Let's consider the case $\mathcal{D}_x(\delta) = n+1$. If we have $Y \in \mathcal{L}_x(\delta)$ then, by definition, we have

$$Y \in \bigcup \left\{ \; \dot{\bigcup} \{ \; \mathcal{L}_y(\delta) \,|\, x \rightarrowtail \ \textsc{And}_Y \rightarrowtail y \; \} \,|\, Y \in G(x) \right\}$$

Hence, there exists $Z \in G(x)$ such that $Y \in \dot{\bigcup}\{\ \mathcal{L}_z(\delta) \,|\, x \rightarrowtail \textsc{And}_Z \rightarrowtail z\ \}$. By definition of $\dot{\cup}$ we have that

$$\forall z_i \in Z\ .\ \exists Z_i \in \mathcal{L}_{z_i}(\delta)\ .\ Y = \bigcup_i Z_i$$

Clearly we have $\forall z_i \in Z\ .\ \mathcal{D}_{z_i}(\delta) = \mathcal{D}_x(\delta) - 1 = n$, therefore, by inductive hypothesis we have that $Z_i \in \mathcal{L}_{z_i}(\delta) \Rightarrow z_i \in \mathcal{R}_f(Z_i)$. This means that $Z \subseteq \mathcal{R}_f(\bigcup_i Z_i) = \mathcal{R}_f(Y)$. Since, by construction, we have $x \in \mathcal{R}_f(Z)$, we can conclude, by monotonicity and idempotence of $\mathcal{R}_f$ that $x \in \mathcal{R}_f(Y)$.

Let's prove now the point (2). We prove the thesis by induction on the smallest number of steps n of $\mathcal{R}_f$ necessary in order to introduce x in the refined domain. By construction $\mathcal{R}_f$ id the least fix point of the function $R_f = \lambda X.\, \mathcal{M}(X \cup f(X))$. Consider the closure $\delta \in \mathit{uco}(C)$ and $x \in C \smallsetminus \delta$.

Base: If $n = 1$, then $x \in \mathcal{M}(\delta \cup f(\delta))$. We can have two cases: $x \in f(\delta)$ or $x = \bigwedge Z$ where $Z \cap \delta \neq \varnothing$ and $Z \cap f(\delta) \neq \varnothing$. Suppose $x \in f(\delta)$, then there exists a minimal $Z \subseteq \delta$ such that $x = f(Z)$ (or $x \in f(Z)$). Therefore $\forall z \in Z$ we have $x \rightarrowtail \textsc{And}_Z \rightarrowtail z$. Being $Z \in G(x)$, we have

$$\begin{aligned}\{Z\} &= \dot{\bigcup}\{\ \{\{z\}\} \,|\, x \rightarrowtail \textsc{And}_Z \rightarrowtail z\ \} \\ &\subseteq \bigcup \left\{\ \dot{\bigcup}\{\ \mathcal{L}_y(\delta) \,|\, x \rightarrowtail \textsc{And}_Y \rightarrowtail y\ \} \,|\, Y \in G(x)\ \right\} \\ &= \mathcal{L}_x(\delta)\ \Rightarrow\ Z \in \mathcal{L}_x(\delta)\end{aligned}$$

By the hypothesis we have $Z \subseteq \delta$, and therefore we have the thesis.
Consider now that $x = \bigwedge Z$ in the hypotheses written above. Therefore, we have $Z_1 \stackrel{\text{def}}{=} Z \cap \delta \neq \varnothing$ and $Z_2 \stackrel{\text{def}}{=} Z \cap f(\delta) \neq \varnothing$. By construction of the tree, we have $x \rightarrowtail \textsc{And}_Z \rightarrowtail z$ since $Z \in G(x)$. Clearly for all $z \in Z_1$ we have $z \not\rightarrowtail$, while for all $z \in Z_2$ there exists $Y_z \subseteq \delta$ such that $z = f(Y_z)$ (or $x \in f(Y_z)$), then $z \rightarrowtail \textsc{And}_{Y_z} \rightarrowtail y$ for each $y \in Y_z$ with $y \not\rightarrowtail$. Let's now considered some consequences of the constructions made. By definition of $\mathcal{L}_x(\delta)$ and of Z, we have that

$$\dot{\bigcup}\{\ \mathcal{L}_z(\delta) \,|\, x \rightarrowtail \textsc{And}_Z \rightarrowtail z\ \} \subseteq \mathcal{L}_x(\delta)$$

On the other hand, we have, by construction, that

$$\begin{aligned}\dot{\bigcup}\left\{\ \mathcal{L}_z(\delta) \,\middle|\, x \rightarrowtail \textsc{And}_Z \rightarrowtail z\ \right\} &= \dot{\bigcup}\left\{\ \{\{z\}\} \,\middle|\, z \in Z_1\ \right\} \dot{\cup} \dot{\bigcup}\left\{\ \mathcal{L}_z(\delta) \,\middle|\, z \in Z_2\ \right\} \\ &= \{Z_1\} \dot{\cup} \dot{\bigcup}\left\{\ \bigcup\left\{\ \dot{\bigcup}\left\{\ \mathcal{L}_y(\delta) \,\middle|\, z \rightarrowtail \textsc{And}_Y \rightarrowtail y\ \right\} \,\middle|\, Y \in G(z)\ \right\} \,\middle|\, z \in Z_2\ \right\} \\ &= \{Z_1\} \dot{\cup} \dot{\bigcup}\left\{\ \dot{\bigcup}\left\{\ \{\{y\}\} \,\middle|\, z \rightarrowtail \textsc{And}_{Y_z} \rightarrowtail y\ \right\} \,\middle|\, z \in Z_2\ \right\} \\ &= \{Z_1\} \dot{\cup} \dot{\bigcup}\left\{\ \{Y_z\} \,\middle|\, z \in Z_2\ \right\} = Z_1 \cup \bigcup_{z \in Z_2} Y_z \in \mathcal{L}_x(\delta)\end{aligned}$$

but clearly, by construction, $Z_1 \cup \bigcup_{z \in Z_2} Y_z \subseteq \delta$.

Inductive Step: Suppose that if x is introduced in n steps of R_f then the thesis holds. Consider the case $n+1$ and consider $x \in \mathcal{R}_f(\delta)$. This last hypothesis implies that $\exists Z \subseteq \mathcal{R}_f(\delta) \,.\, Z \in G(x)$. In particular $n+1$ is the minimum number of steps for introducing x, namely $x \notin R_f^n(\delta)$ but $Z \subseteq R_f^n(\delta)$. By inductive hypothesis we have that

$$\forall z \in Z \ (y \in \mathcal{R}_f(\delta)) \ \Rightarrow \ \exists Y_z \in \mathcal{L}_z(\delta) \,.\, Y_z \subseteq \delta$$

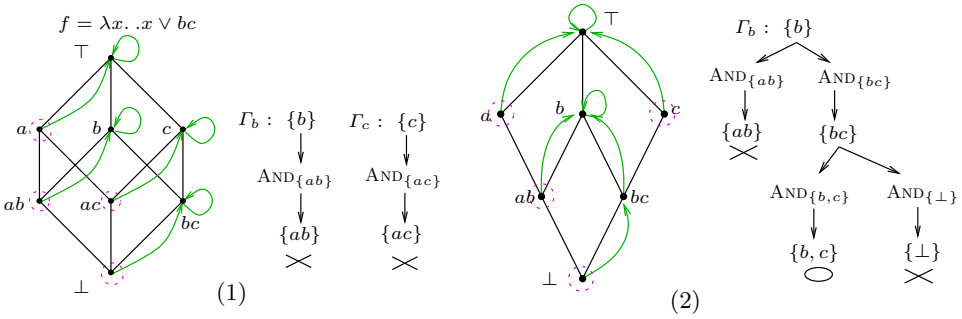
On the other hand we have $\forall z \in Z \,.\, x \rightarrowtail \ \textsc{And}_Z \rightarrowtail z$, so

$$\begin{aligned}\mathcal{L}_x(\delta) = \bigcup \left\{ \dot{\bigcup} \{ \ \mathcal{L}_y(\delta) \,|\, x \rightarrowtail \ \textsc{And}_Y \rightarrowtail y \ \} \,|\, Y \in G(x) \right\} \\ \Rightarrow \bigcup \{ \ \mathcal{L}_z(\delta) \,|\, x \rightarrowtail \ \textsc{And}_Z \rightarrowtail z \ \} \subseteq \mathcal{L}_x(\delta) \Rightarrow \bigcup_{z \in Z} Y_z \in \mathcal{L}_x(\delta)\end{aligned}$$

But, by construction, we have $\bigcup_{z \in Z} Y_z \subseteq \delta$ and therefore we have the thesis.

In the following example, we show two different domains together with the construction of the corresponding trees. In particular, in the following, we will put an X under a leaf contained in the candidate base, and we will put an O under leaves containing cycles.

Example 3.17. Consider the concrete domains depicted below and suppose that ρ is the identity. We want to check, by using the tree construction, if the candidate bases, generate, respectively, the whole domains. It's clear that, in order to generate the whole concrete domain from the base, we have to generate $\mathit{Mirr}(\rho)$, denoted with dashed circles. Let's consider an example of construction of $\Gamma_x(\delta)$ where $\delta = \mathcal{M}(\mathit{firr}(C))$ is the candidate base. Consider the function f is drawn on the domains.



Note that, in figure (1) the domain refinement $\mathcal{R}_f$ is join uniform and it is simple to verify that the base is exactly the closure $\{\top, a, ab, ac, \bot\}$. Also in figure (2) the refinement $\mathcal{R}_f$ associated with the represented function is join uniform, but the candidate base $\delta = \mathcal{M}(\mathit{firr}(C)) = \{\top, a, c, ab, \bot\}$ is clearly too concrete: $\mathcal{R}_f(\{\top, a, c, ab, \bot\}) = C$ and also $\mathcal{R}_f(\{\top, a, c, \bot\}) = C$.

Since abstract domains are Moore families, it's clear that, if the goal is to compress the concrete domain C, then δ can be a base for $\mathcal{R}_f$ if $\mathit{Mirr}(C) \subseteq \mathcal{R}_f(\delta)$. Moreover,

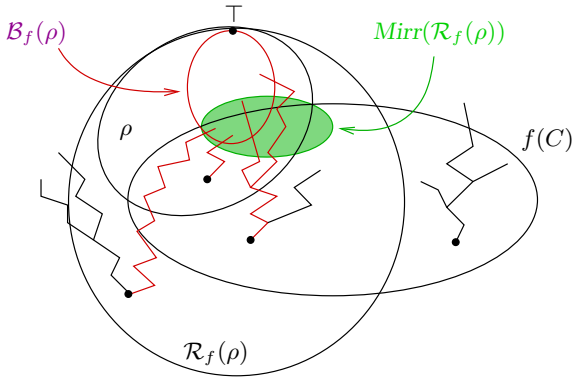


Fig. 3.3. The global picture

if $M_f^C \stackrel{\text{def}}{=} Mirr(C) \cap \dot{f}(C)$ and $I_f(C) \stackrel{\text{def}}{=} Mirr(C) \cap firr(C)$ then, since any $x \in M_f^C$ can still be generated by refinement from δ, being f-reducible, δ has to include at least $I_f(C)$. Indeed, the elements in $I_f(C)$ cannot be generated neither by f nor by meet. Therefore, if $\delta \in uco(C)$ is a candidate base then $I_f(C) \subseteq \delta$. In order to generalize this situation to the base of any abstraction ρ of C, we require $Mirr(\mathcal{R}_f(\rho)) \subseteq \mathcal{R}_f(\delta)$.

Lemma 3.18. *Let $\rho \in uco(C)$. Then*

(1) $x \in Mirr(\mathcal{R}_f(\rho)) \cap \rho \;\Rightarrow\; x \in Mirr(\rho)$ *and*
(2) $x \in Mirr(\mathcal{R}_f(\rho)) \smallsetminus \rho \;\Rightarrow\; x \in \dot{f}(C)$.

Proof. Let $x \in Mirr(\mathcal{R}_f(\rho)) \cap \rho$ and suppose that $x \notin Mirr(\rho)$. Then there exists $Y \subseteq \rho \smallsetminus \{x\}$ such that $\bigwedge Y = x$. But $\mathcal{R}_f(\rho) \supseteq \rho$ so $Y \subseteq \mathcal{R}_f(\rho)$, namely x is not meet-irreducible in $\mathcal{R}_f(\rho)$, which is absurd for the hypothesis made. Let $x \in Mirr(\mathcal{R}_f(\rho)) \smallsetminus \rho$ and suppose that $x \notin \dot{f}(C)$. Then if $x \in Mirr(C)$, being $x \notin \dot{f}(C)$, we have that x cannot be generated by refinement, but this implies that if $x \notin \rho$ then it cannot be in $\mathcal{R}_f(\rho)$ either, which is absurd. Therefore, being x generated in $\mathcal{R}_f(\rho)$, there exists $Y \subseteq \mathcal{R}_f(\rho)$ such that $\bigwedge Y = x$, but this means that x is not meet-irreducible in $\mathcal{R}_f(\rho)$, which is absurd. It is worth noting that the hypothesis made implies that $x \in \dot{f}(C) \smallsetminus \rho$.

Lemma 3.18 implies that any candidate base has to generate the following set of meet-irreducibles:

$$M_f^\rho \stackrel{\text{def}}{=} Mirr(\{x \in \dot{f}(C) \smallsetminus \rho \mid \exists L \in \mathcal{L}_x(\rho) \,.\, L \subseteq \rho\}) \cup Mirr(\rho).$$

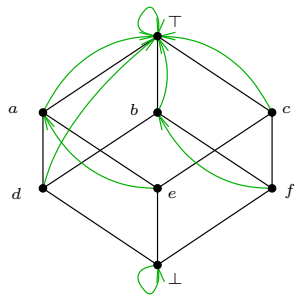
M_f^ρ is clearly redundant even though it does not require the construction of $\mathcal{R}_f(\rho)$, as specified by the following proposition.

Proposition 3.19. *If $\rho \in uco(C)$ and $\rho \sqsubseteq \delta$, then $M_f^\rho \subseteq \mathcal{R}_f(\delta) \Rightarrow \mathcal{R}_f(\delta) = \mathcal{R}_f(\rho)$.*

Proof. Note that Lemma 3.18 says that whenever $x \in \mathit{Mirr}(\mathcal{R}_f(\rho))$ we have that $(x \in \rho \;\Rightarrow\; x \in \mathit{Mirr}(\rho))$ or $x \notin \rho \;\Rightarrow\; x \in \dot{f}(C))$. If $x \notin \mathit{Mirr}(\rho) \cup (\dot{f}(C) \smallsetminus \rho)$ then $x \notin \mathit{Mirr}(\mathcal{R}_f(\rho))$.
If $x \in \mathit{Mirr}(\rho)$ then trivially $x \in M_f(\rho)$ and so we have $x \in \mathcal{R}_f(\delta)$, being $M_f(\rho) \subseteq \mathcal{R}_f(\delta)$. If $x \in \dot{f}(C) \smallsetminus \rho$, by the hypotheses we have $x \in \mathcal{R}_f(\rho)$, therefore by Theorem 3.16, we have $\exists Y \in \mathcal{L}_x(\rho) \,.\, Y \subseteq \rho$. This implies that $x \in \left\{\, x \in \dot{f}(C) \smallsetminus \rho \,\middle|\, \exists L \in \mathcal{L}_x(\rho) \,.\, L \subseteq \rho \,\right\}$. At this point if x is not meet-irreducible in this set, then there exists a set Y of meet-irreducible of the set such that $\bigwedge Y = x$, but if Y, which is in $M_f(\rho)$, is generated in $\mathcal{R}_f(\rho)$, then also x is generated in $\mathcal{R}_f(\rho)$. Namely $x \in \mathcal{R}_f(M_f(\rho))$, which implies $x \in \mathcal{R}_f(\delta)$, because $M_f(\rho) \subseteq \mathcal{R}_f(\delta)$. In this way we proved that $\mathit{Mirr}(\mathcal{R}_f(\rho)) \subseteq \mathcal{R}_f(\delta)$ that implies, by monotonicity of $\mathcal{R}_f$, that $\mathcal{R}_f(\delta) = \mathcal{R}_f(\rho)$.

The proposition above says that M_f^ρ contains all the elements of C that can be generated starting from elements in ρ and that are necessary in order to generate the whole domain $\mathcal{R}_f(\rho)$. Namely it contains the set of all the meet-irreducible elements of $\mathcal{R}_f(\rho)$ that will be generated by refinement. We use this set to derive an algorithm for computing the base of an abstract domain. Since we deal with finite domains, the algorithm gives also a method for deciding whether a refinement is not join-uniform. In Fig. 3.3 we depict the scenario. The idea is that, for each element in $\mathcal{R}_f(\rho)$ we should check if there is an element in the base generating it. Prop. 3.19 guarantees the same result even if we check the generation only for a particular set of elements, that is M_f^ρ.

Remark 3.20. Note that the left adjoint of an expander, when it exists, i.e., the left adjoint of a join-uniform lco, it is not necessarily an erosion, namely a meet-uniform core. Let's see an example:



In this example it is simple to verify that the completeness refinement is join-uniform and the base of the whole domain is $\{\top, c, e, f, \bot\}$. On the other hand, the completeness core is not meet-uniform. Indeed, the completeness core of both the closures $\{\top, e\}$ and $\{\top, f\}$ is $\{\top\}$, while the core of their reduced product $\{\top, e, f, \bot\}$ is $\{\top, \bot\}$.

Computing the base

Let $\mathcal{D}(\rho)$ be the collection of all the possible Moore families which are strictly contained in the closure $\rho \in \mathit{uco}(C)$, and can generate $\mathcal{R}_f(\rho)$ by refinement. They contain all the elements that cannot be generated by refinement, i.e., $I_f(C)$: $\mathcal{D}(\rho) \stackrel{\text{def}}{=} \left\{ \rho \smallsetminus \{x\} \,\middle|\, x \in \mathit{Mirr}(\rho) \smallsetminus I_f(C) \right\} \subseteq \mathit{uco}(\rho)$. We design an algorithm for computing the base of an abstract domain ρ, w.r.t. a refinement $\mathcal{R}_f$, by induction on the construction of a sequence of sets N_n, such that:

$$\forall n \in \mathbb{N},\ \forall \delta \in N_n :\ \rho \sqsubseteq \delta,\ \mathcal{R}_f(\delta) \sqsubseteq \rho, \text{and } (\sigma \in N_{n+1} \smallsetminus N_n \ \Rightarrow\ |\sigma| < |\delta|).$$

The domains $\delta \in N_n$ are the candidate bases for $\mathcal{R}_f(\rho)$ after n steps of the algorithm. The set $\mathcal{P}$ will include the minimal candidates, i.e., those domains that, by erasing any of their elements, are unable to regenerate the whole domain.

Step 1: $\mathcal{P}(\rho) := \varnothing$. We define the following sets: $\delta_1 := \rho$ and $N_1 := \{\delta_1\}$;

Step $n+1$: Let $\delta_n \in N_n$. If $\mathcal{D}(\delta_n) = \varnothing$ then $\mathcal{P}(\rho) := \mathcal{P}(\rho) \cup \{\delta_n\}$ (δ_n is minimal since all the elements that can be erased are re-introduced by the Moore closure). Otherwise $\mathcal{U}_n \stackrel{\text{def}}{=} \left\{ \delta \in \mathcal{D}(\delta_n) \,\middle|\, \exists y \in M_f^\rho \smallsetminus \delta \,.\, \forall L \in \mathcal{L}_y(\delta) \,.\, L \not\subseteq \delta \right\}$. This is the set of all the candidate bases that are not able to generate all the elements in M_f^ρ. We define the set of candidate bases more abstract than δ_n, at step $n+1$, as $N_{n+1}^{\delta_n} := \mathcal{D}(\delta_n) \smallsetminus \mathcal{U}_n$. If $N_{n+1}^{\delta_n} = \varnothing$ then δ_n is minimal and $\mathcal{P}(\rho) := \mathcal{P}(\rho) \cup \{\delta_n\}$. The set of all the candidate bases at step $n+1$ is therefore $N_{n+1} := \bigcup_{\delta_n \in N_n} N_{n+1}^{\delta_n}$.

Since, at each step n, we reduce the size of the domains in N_n, and since C is finite, it is immediate to prove the following proposition.

Proposition 3.21. *The algorithm terminates and*

$$\mathcal{P}(\rho) = \left\{ \delta \in \mathit{uco}(\rho) \,\middle|\, \begin{array}{l} \mathit{Mirr}(\delta) \smallsetminus I_f(C) = \varnothing \ \vee \\ (\forall x \in \mathit{Mirr}(\delta) \smallsetminus I_f(C) \,.\, \exists y \in M_f^\rho \smallsetminus (\delta \smallsetminus \{x\}) \,. \\ \qquad\qquad \forall L \in \mathcal{L}_y(\delta \smallsetminus \{x\}) \,.\, L \not\subseteq \delta \smallsetminus \{x\}) \end{array} \right\}$$

The correctness of the algorithm is obtained by proving that $\mathcal{P}(\rho)$ contains all and only the domains whose sub-domains are unable to generate ρ by refinement.

Lemma 3.22. *Let $\rho \in \mathit{uco}(C)$ and $\delta \in \mathit{uco}(\rho)$ with $\delta \neq \rho$, $\mathcal{R}_f(\delta) = \mathcal{R}_f(\rho)$, and $x \in \mathit{Mirr}(\delta)$ then $\mathcal{R}_f(\delta \smallsetminus \{x\}) \neq \mathcal{R}_f(\rho)$ if and only if $\delta \in \mathcal{P}(\rho)$.*

Proof. ($\Leftarrow$) If $\delta \in \mathcal{P}(\rho)$, then by construction we have that $M_f(\rho) \subseteq \mathcal{R}_f(\delta)$ and then, by Proposition 3.19, this implies that $\mathcal{R}_f(\delta) = \mathcal{R}_f(\rho)$. Moreover also the minimality, i.e., $\forall x \in \mathit{Mirr}(\delta) \,.\, \mathcal{R}_f(\delta \smallsetminus \{x\}) \neq \mathcal{R}_f(\rho)$, derives from the construction of $\mathcal{P}$ in the algorithm.

($\Rightarrow$) Consider $\delta \in \mathit{uco}(\rho)$ such that $\mathcal{R}_f(\delta) = \mathcal{R}_f(\rho)$ and such that for each element $x \in \mathit{Mirr}(\delta)$ we have $\mathcal{R}_f(\delta \smallsetminus \{x\}) \neq \mathcal{R}_f(\rho)$. Suppose that $\delta \notin \mathcal{P}(\rho)$, namely

that $\exists \bar{x} \in \mathit{Mirr}(\delta) \smallsetminus I_f(C) \,.\, \forall y \in M_f(\rho) \,.\, \exists L \in \mathcal{L}_y(\bar{\delta}(\rho) \smallsetminus \{\bar{x}\}) \,.\, L \subseteq \bar{\delta} \smallsetminus \{\bar{x}\}$ and this implies that $M_f(\rho) \subseteq \mathcal{R}_f(\delta \smallsetminus \{\bar{x}\})$. But for what we said above this implies, by Proposition 3.19, that $\rho \subseteq \mathcal{R}_f((\bar{\delta} \smallsetminus \{\bar{x}\}))$, which is absurd for the hypothesis made.

Theorem 3.23. *Let $\mathcal{B}_f(\rho) = \bigcap_{\delta \in \mathcal{P}(\rho)} \delta$. The completeness refinement $\mathcal{R}_f$ is join-uniform relatively to ρ, and its base is $\mathcal{B}_f(\rho)$ if and only if $\{\mathcal{B}_f(\rho)\} = \mathcal{P}(\rho)$.*

Proof. Consider $\mathcal{B}_f(\rho) = \bigcap_{\delta \in \mathcal{P}(\rho)} \delta$. By Lemma 3.22 its clear that in $\mathcal{P}(\rho)$ we have all and only the minimal closures δ such that $\mathcal{R}_f(\delta)$ generates the whole domain $\mathcal{R}_f(\rho)$. Suppose that $\mathcal{B}_f(\rho) \in \mathcal{P}(\rho)$, this implies that the following equalities hold:

$$\begin{aligned} \bigcap \{\ \delta \in \mathit{uco}(\rho) \,|\, \mathcal{R}_f(\delta) = \rho\ \} &= \bigcap \left\{\ \delta \in \mathit{uco}(\rho) \,\middle|\, \begin{array}{l} \mathcal{R}_f(\delta) = \rho, \\ \delta \text{ minimal} \end{array} \right\} \\ &= \textstyle\bigcap_{\delta \in \mathcal{P}(\rho)} \delta = \mathcal{B}_f(\rho) \end{aligned}$$

Moreover, by Lemma 3.22, we have that if $\mathcal{B}_f(\rho) \in \mathcal{P}(\rho)$ then $\mathcal{R}_f(\mathcal{B}_f(\rho)) = \rho$. Suppose now that $\mathcal{B}_f(\rho) \notin \mathcal{P}(\rho)$ and suppose that $\mathcal{R}_f(\mathcal{B}_f(\rho)) = \rho$, namely that $\mathcal{R}_f$ is join-uniform. Being $\mathcal{B}_f(\rho) = \bigcap_{\delta \in \mathcal{P}(\rho)} \delta \notin \mathcal{P}(\rho)$ it means that for each $\delta \in \mathcal{P}(\rho)$ there exists $C_\delta \neq \varnothing$ such that $\mathcal{B}_f(\rho) = \delta \cup C_\delta$, namely $\forall \delta \,.\, \mathcal{B}_f(\rho) \subsetneq \delta$ with $\mathcal{R}_f(\mathcal{B}_f(\rho)) = \rho$ which is absurd by construction of the δ's.

The algorithm described above provides a systematic method for deriving the most abstract domain with the same refinement, of any finite domain as regards a $\mathcal{F}$-completeness refinement. The complexity of the algorithm strongly depends upon the structure of C and on how f behaves. In the worst case it may be necessary to check the whole concrete domain C, while in the best case it is sufficient to check $\mathit{Mirr}(C)$ whose size, in Boolean lattices, is logarithmic on the size of C.

An application in predicate abstraction

In this section, we consider two different examples of bases for predicate abstractions of transition systems $\langle \Sigma, \tau \rangle$, w.r.t $\mathcal{F}$-complete refinements $\mathcal{R}_{post[\tau]}$. The idea of predicate abstraction is to choose a set of predicates φ representing sets $X \subseteq \Sigma$ of concrete states, those which satisfy φ: $X = \{\ s \,|\, s \models \varphi\ \}$ [11, 35, 70]. The key point in predicate abstraction is the choice of the so called *abstract state lattice*, which is naturally induced by composing, by conjunction, the chosen predicates. In this context, the abstract state lattice is a closure on $\wp(\Sigma)$. In the first example, we show two different abstract state lattices with comparable but different bases, while in the second example we show different abstract state lattices sharing the same base. These examples show how bases can be useful to derive a least set of most-abstract predicates which represent the base of any given abstract state lattice. These bases can be useful both to compare abstract state lattices and to design optimal predicates for a given system. In this latter case, the predicates in the base cannot be removed without changing the way abstract state lattices

can be refined. Moreover, imagine the analysis of a system which is specified by a family of transition relations $\tau_1, \ldots, \tau_n$ and an abstract domain X. It is clear that if we are not interested in the precision of the analysis for τ_i, then for any $j \neq i$: $\mathcal{R}_{\widetilde{pre}[\tau_j]}(\mathcal{B}_{\widetilde{pre}[\tau_i]}(X))$ is the candidate non-redundant domain, which is backward complete for $post[\tau_j]$. In predicate abstraction, this method provides a systematic way to simplify the definition of predicates by removing all those predicates that are necessary to achieve completeness for τ_i. The idea, shown in Fig. 3.4, is

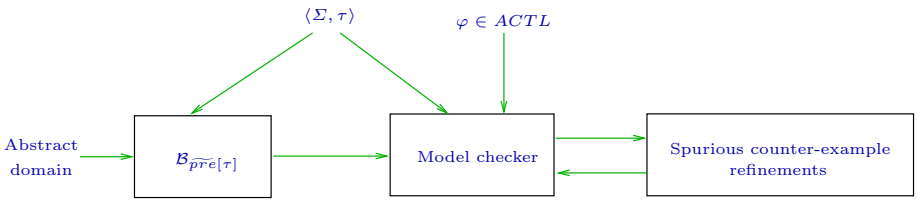


Fig. 3.4. An application to predicate abstraction.

that of applying the construction of the base, in order to simplify the formula's models, in model checking. In particular, recall that a model contains spurious counter-example for a formula, if it is not complete, as regards the post relation [60]. Indeed, given a formula φ, and the model that we want to check, we could find its base in order to erase all the information that could be added by completeness refinement, and then, by using, for instance, the spurious counter-example refinement [18], we could add just the information needed for modeling the given formula φ. In this way, we could obtain the most abstract model for the formula φ.

Example 3.24. Let's consider the transition system in Figure 3.5 (a), with transition relation τ, and $f = post[\tau]$. We consider the two different abstractions represented in Figure 3.5 (b) and (c). We have that the points double circled are the elements of the abstractions, while the points single circled are the elements generated by the refinement. In the following, we will omit the nodes $\textsc{And}_Y$, when possible. First we consider some sets of elements of C which are used by the algorithm: $M_f^C = \{\{1,2\},\{1,3\}\}$, $I_f(C) = \{\{2,3\}\}$, $\dot{f}(C) = \{\top, \{1,2\}, \{1,3\}\}$. Consider the abstraction $\rho = \{\top, \{1,2\}, \{2\}\}$, represented in figure (b). We note that $Mirr(\rho) = \{\{1,2\},\{2\}\}$ while $M_f^\rho = \{\{1,2\},\{2\}\}$, because the tree with root $\{1,3\}$ has no leaves in ρ.

Step 1: $\delta_1 = \rho$ and $N_1 = \{\delta_1\}$;

Step 2: $\mathcal{D}(\delta_1) = \{\{\top, \{1,2\}\}, \{\top, \{2\}\}\}$. Moreover, $\{1,2\} \in \Gamma_{\{1,2\}}(\{\top, \{1,2\}\})$ while $\{1,2\} \rightarrowtail \{2\}$ in $\Gamma_{\{1,2\}}(\{\top, \{2\}\})$, therefore both the possible successors $\{\top, \{1,2\}\}$ and $\{\top, \{2\}\}$ may generate $\{1,2\}$. Let's consider $\{2\}$. It's clear that $\{2\} \in \Gamma_{\{2\}}(\{\top, \{2\}\})$ while $\{2\} \rightarrowtail \{\{1,2\},\{2,3\}\}$ in $\Gamma_{\{2\}}(\{\top, \{1,2\}\})$, namely $\{2\}$ cannot be generated from $\{\top, \{1,2\}\}$. Hence, we obtain $\mathcal{U}_2 = \{\{\top, \{1,2\}\}\}$ and $N_2 = \{\{\top, \{2\}\}\}$;

STEP 3: Let $\delta_2 \in N_2$, then $\mathcal{D}(\delta_2) = \{\top\}$, but it is clear that nothing can be generated from $\top$, therefore $\mathcal{U}_3 = \{\top\}$ and $N_3 = \varnothing$. In this way we have that $\mathcal{P} = \{\{\top, \{2\}\}\}$ and the algorithm terminates.

It is clear that, by Theorem 3.23, the closure $\{\top, \{2\}\}$ is the base of $\mathcal{R}_f(\rho)$. Consider now the abstraction $\eta = \{\top, \{1,3\}, \{2\}, \{3\}, \bot\}$, represented in Fig. 3.5 (c) by double circled points.

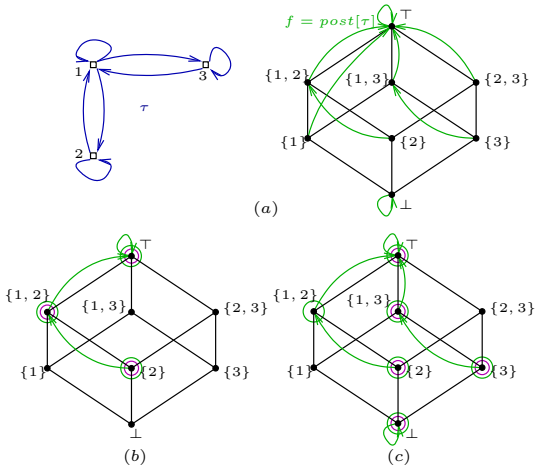


Fig. 3.5. Example of compression

We note that $Mirr(\eta) = \{\{1,3\}, \{2\}, \{3\}\}$ while it is simple to find that $M_f^\eta = \{\{1,2\}, \{1,3\}, \{2\}, \{3\}\}$, because $\{1,2\} \rightarrowtail \{2\}$ in $\Gamma_{\{1,2\}}(\eta)$.

STEP 1: $\delta_1 = \eta$ and $N_1 = \{\delta_1\}$;

STEP 2: $\mathcal{D}(\delta_1) = \{\{\top, \{2\}, \{3\}, \bot\}, \{\top, \{1,3\}, \{2\}, \bot\}, \{\top, \{1,3\}, \{3\}, \bot\}\}$. It is clear that $\{1,2\} \rightarrowtail \{2\} \rightarrowtail \{\{1,2\}, \{2,3\}\}$ in $\Gamma_{\{1,2\}}(\{\top, \{1,3\}, \{3\}, \bot\})$, namely $\{\top, \{1,3\}, \{3\}, \bot\}$ is in $\mathcal{U}_2$. Note that each other closure containing $\{2\}$ may generate $\{1,2\}$. Consider $\{3\}$, it's clear that $\{1,3\} \rightarrowtail \{3\} \rightarrowtail \{\{1,3\}, \{2,3\}\}$ in $\Gamma_{\{3\}}(\{\top, \{1,3\}, \{2\}, \bot\})$, namely $\{\top, \{1,3\}, \{2\}, \bot\} \in \mathcal{U}_2$ doesn't generate $\{3\}$. Finally $\{1,3\} \rightarrowtail \{3\}$ in $\Gamma_{\{1,3\}}(\{\top, \{2\}, \{3\}, \bot\})$. Hence, we obtain the sets $\mathcal{U}_2 = \{\{\top, \{1,3\}, \{2\}, \bot\}, \{\top, \{1,3\}, \{3\}, \bot\}\}$ and $N_2 = \{\{\top, \{2\}, \{3\}, \bot\}\}$;

STEP 3: Let $\delta_2 \in N_2$, then $\mathcal{D}(\delta_2) = \{\{\top, \{2\}, \bot\}, \{\top, \{3\}, \bot\}\}$, but it is clear that, from what we saw in the previous step, $\{2\}$ and $\{3\}$ can be generated only from closures that contain respectively these two elements. Namely we obtain $\mathcal{U}_3 = \{\{\top, \{2\}, \bot\}, \{\top, \{3\}, \bot\}\}$ and $N_3 = \varnothing$. In this way we have that $\mathcal{P} = \{\{\top, \{2\}, \{3\}, \bot\}\}$ and the algorithm terminates.

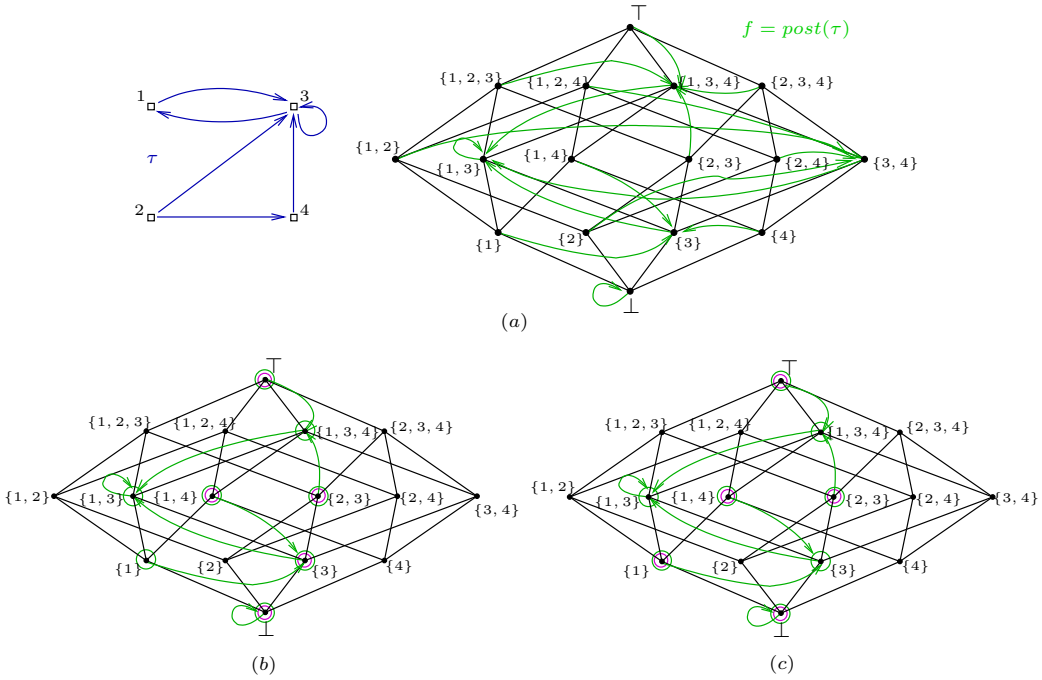


Fig. 3.6. A transition system, two abstractions and their complete refinements

It is clear that, by Theorem 3.23 the closure $\{\top, \{2\}, \{3\}, \bot\}$ is the base of $\mathcal{R}_f(\eta)$. By observing $\mathcal{B}_f(\rho) \sqsupset \mathcal{B}_f(\eta)$ we can compare the two closures ρ and η. This implies that $\mathcal{R}_f(\rho) \sqsupset \mathcal{R}_f(\eta)$.

Example 3.25. On the left side of Figure 3.6(a) we have the transition system with states $\Sigma = \{1, 2, 3, 4\}$ and transition relation τ. On the right side we have the concrete domain $\wp(\{1, 2, 3, 4\})$ including the function $f = post[\tau]$. In Figure 3.6(b) and (c) we consider, respectively, two different abstractions of the concrete domain and their corresponding refinements. The elements double circled are the points in the abstractions, the elements single circled are the points added by $\mathcal{R}_{post[\tau]}$. It is worth noting that both $\eta = \{\top, \{1, 4\}, \{2, 3\}, \{1\}, \bot\}$ and $\rho = \{\top, \{1, 4\}, \{2, 3\}, \{3\}, \bot\}$ have the same refinement: $\mathcal{R}_{post[\tau]}(\rho) = \mathcal{R}_{post[\tau]}(\eta) = \{\top, \{1, 3, 4\}, \{1, 4\}, \{2, 3\}, \{1, 3\}, \{1\}, \{3\}, \bot\}$. In this case $M_f^C = \{\{1, 3, 4\}\}$, the irreducibles are $I_f(C) = \{\{1, 2, 3\}, \{1, 2, 4\}, \{2, 3, 4\}\}$, and the reducibles are $\dot{f}(C) = \{\{1, 3, 4\}, \{1, 3\}, \{3, 4\}, \{3\}\}$. Note that $Mirr(\rho) = \{\{1, 4\}, \{2, 3\}, \{3\}\}$ while it is simple to verify that $M_f^\rho = \{\{1, 4\}, \{2, 3\}, \{3\}, \{1, 3\}, \{1, 3, 4\}\}$. Moreover $Mirr(\eta) = \{\{1, 4\}, \{2, 3\}, \{1\}\}$ while it is simple to verify that $M_f^\eta = \{\{1, 4\}, \{2, 3\}, \{1\}, \{1, 3\}, \{1, 3, 4\}\}$. These fact hold because the trees with root respectively $\{1, 3\}$ and $\{1, 3, 4\}$ have a leaf both in ρ and η. The algorithm terminates for both domains in 3 steps, computing the base $\mathcal{B}_f(\rho) = \mathcal{B}_f(\eta) = \{\top, \{1, 4\}, \{2, 3\}, \bot\}$. The least set of generating predicates is $\mathcal{B}_f(\rho)$. Analogously, if $\rho' = \{\top, \{1, 4\}, \{3\}, \bot\}$ and $\eta' = \{\top, \{1, 4\}, \{1\}, \bot\}$, then $\mathcal{B}_f(\rho') = \mathcal{B}_f(\eta') =$

$\{\top, \{1,4\}\}$. Because $\mathcal{B}_f(\rho) \sqsubset \mathcal{B}_f(\rho')$, this means that ρ and ρ' will lead to different complete domains once refined.

3.3 Making domain transformers right reversible

In this section, we introduce a systematic method for making any domain transformer reversible as a pair expander/compressor. As usual we consider domain refinements, being domain simplification dual.

Right-reversibility corresponds to join-uniformity. In this case, the set of all join-uniform refinements is a dual-Moore-family of $lco(C)$ [63]. In the following we characterize the corresponding transformation which maps any refinement $\mathcal{R}$ to the most abstract refinement which is more concrete than $\mathcal{R}$ and join-uniform:

$$(\mathcal{R})^{\vartriangle} = \bigsqcup \{\ \tau \mid \tau \sqsubseteq \mathcal{R}, \tau \text{ is join-uniform}\ \}$$

Theorem 3.26. *Let $\eta \in lco(L)$, C complete lattice, then η is join-uniform if and only if $\forall x \in \eta\,.\,\forall Z \subseteq C\,.\,((\eta(Z) = \text{const} \ \wedge\ Z \not\geq x) \Rightarrow\ \bigvee Z \not\geq x)$, where we denote as $\eta(Z) = \text{const}$ the fact $\exists w \in \eta\,.\,\forall z \in Z\,.\,\eta(z) = w$ and as $Z \not\geq x$ the fact $\forall z \in Z\,.\,z \not\geq x$.*

Proof. ($\Rightarrow$) We prove that if $\exists x \in \eta\,.\,\exists Z \subseteq C\,.\,(\eta(Z) = const \wedge Z \not\geq x \wedge \bigvee Z \geq x)$ then η is not join-uniform. We know that Z is such that $\exists w \in \eta\,.\,\forall z \in Z\,.\,\eta(z) = w$, this means that $\forall z \in Z\,.\,z \geq w$. Moreover the hypothesis $\bigvee Z \geq x$ implies, by monotonicity, that $\eta(\bigvee Z) \geq x$ and if $\eta(\bigvee Z) = w$ then we would have $w \geq x$ and this is absurd because otherwise we would have $\forall z \in Z\,.\,z \geq x$, which is avoided by the hypotheses on x and Z. Therefore $\eta(\bigvee Z) \neq w$, namely the closure η is not join-uniform.

($\Leftarrow$) We prove that if η is not join-uniform then $\exists x \in \eta\,.\,\exists Z \subseteq C\,.\,(\eta(Z) = const \ \wedge\ Z \not\geq x\ \wedge\ \bigvee Z \geq x)$. Consider $w \in \eta$ and $Z = \{\ z \in C \mid \eta(z) = w\ \}$, then $\forall z \in Z\,.\,z \geq w$ and this implies that $\bigvee Z \geq w$. By monotonicity this implies $\eta(\bigvee Z) \geq w$. We supposed that η was not join-uniform, this means that $\eta(\bigvee Z) > w$, i.e. there isn't the equality. Let $x = \eta(\bigvee Z)$, therefore $\bigvee Z \geq \eta(\bigvee Z) = x$. Moreover we have $\forall z \in Z\,.\,z \not\geq x$, otherwise if it exists $z \in Z$ such that $z \geq x$ we would have also $\eta(z) = w < x = \eta(x)$ by definition of x, which is absurd for the monotonicity of η. All these facts imply that Z is such that $\eta(Z) = const$, by construction, and $Z \not\geq x$ and $\bigvee Z \geq x$ for what we have just proved.

This theorem implies that we can isolate a set of refined domains in η which represent the closest join-uniform refinement contained in η. This set is precisely the transformer making a refinement reversible as a pair expander/compressor, i.e., the transformer which erases all the domains which make the refinement not join-uniform.

$$(\eta)^{\vartriangle} \stackrel{\text{def}}{=} \left\{\, x \in \eta \,\middle|\, \forall Z \subseteq C \,.\, ((\eta(Z) = const, Z \not\geq x) \Rightarrow \bigvee Z \not\geq x) \,\right\}$$

Lemma 3.27. *If $\eta \in lco(C)$ then $(\eta)^{\vartriangle} \in lco(C)$.*

Proof. Consider a set Y of elements y such that $y \in (\eta)^{\vartriangle}(C)$, we have to prove that $\bigvee Y \in (\eta)^{\vartriangle}(C)$. The hypotheses imply that for each $y \in Y$ we have that $\forall Z \subseteq C.(\eta(Z) = w \wedge Z \not\geq y) \Rightarrow \bigvee Z \not\geq y$. Consider $Z \subseteq C$ such that $\eta(Z) = const$ then we prove that $Z \not\geq \bigvee Y$ implies $\bigvee Z \not\geq \bigvee Y$. Therefore suppose $Z \not\geq \bigvee Y$ and suppose $\bigvee Z \geq \bigvee Y$, this condition implies that $\forall y \in Y \,.\, \bigvee Z \geq y$. Consider now the condition $Z \not\geq \bigvee Y$. Therefore $\forall z \in Z \,.\, z \not\geq \bigvee Y$, i.e. $\forall z \in Z \,.\, \exists y \in Y \,.\, z \not\geq y$. If we prove that, with these hypotheses, $\exists y \in Y \,.\, \forall z \in Z \,.\, z \not\geq y$, i.e $Z \not\geq y$, than we would have an absurd because we have $\bigvee Z \geq y$ and that $Z \not\geq y$ when the hypothesis was that $y \in (\eta)^{\vartriangle}(C)$. Suppose that $\forall y \in Y \,.\, \exists z \,.\, z \geq y$. Then we know that $\forall z \in Z \,.\, z \geq w$ and by monotonicity this implies that $w = \eta(z) \geq \eta(y) = y$. Therefore $\forall y \in Y \,.\, w \geq y$ and this implies that $\forall z \in Z \,.\, z \geq w \geq \bigvee Y$, by definition of $\bigvee$, that is absurd by the hypothesis made. This means that $\exists y \in Y.\forall z \in Z.z \not\geq y$, that for the absurd described above implies that $\bigvee Z \not\geq \bigvee Y$. Indeed if $Z \not\geq \bigvee Y$ then $\bigvee Z \not\geq \bigvee Y$, so $\bigvee Y \in (\eta)^{\vartriangle}(C)$

Theorem 3.28. *$\eta \in lco(C)$ is join-uniform iff $(\eta)^{\vartriangle} = \eta$.*

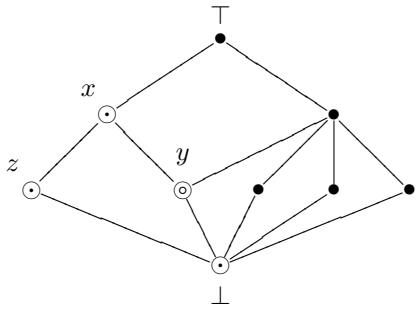
Proof. Trivially we have that if η is join-uniform then it is image of itself. Indeed by Theorem 3.26 all the elements of η satisfy the condition imposed by $(\eta)^{\vartriangle}$. Analogously for each $\eta \in lco(L)$ we have that $(\eta)^{\vartriangle}$ is join-uniform, again by Theorem 3.26. Moreover $(\eta)^{\vartriangle}$ is the most concrete join-uniform closure contained in η. Indeed if there exists another join-uniform closure operator η' contained in η such that $\eta' \sqsubseteq (\eta)^{\vartriangle}$, then there exists at least one element $x \in \eta'$ such that $x \notin (\eta)^{\vartriangle}$. By Theorem 3.26, this means that $x \in (\eta)^{\vartriangle}$ and consequently η' cannot be join-uniform.

Join-uniformity can be characterized on join-irreducible domains.

Theorem 3.29. $(\eta \smallsetminus (\eta)^{\vartriangle}) \cap Jirr(\eta) = \varnothing \;\Leftrightarrow\; \eta = (\eta)^{\vartriangle}$.

Proof. Consider $\eta \neq (\eta)^{\vartriangle}$, namely $\exists x \in \eta$ such that $Z \not\geq x$ and $\bigvee Z \geq x$. Consider $Y \subseteq Jirr(\eta)$ such that $x = \bigvee Y$. We have to prove that there exists $y \in Y$ such that $Z \not\geq y$. Suppose that $\forall y \in Y \,.\, Z \geq y$, namely $\forall z \in Z \,.\, \forall y \in Y \,.\, z \geq y$. Let $w = \eta(Z)$, we can note that $z \geq y$ implies, by monotonicity of η, that $w = \eta(z) \geq \eta(y) = y$, and this holds for each $y \in Y$. We supposed that $x = \bigvee Y$, so by definition of $\bigvee$ we have that $w \geq x$, but we know that for each z we have $z \geq w$, this would mean that $\forall z \in Z \,.\, z \geq x$, which is avoided by the hypotheses on Z and x. Therefore $\exists y \in Y \,.\, Z \not\geq y$. Finally, if we consider this y then we have that $\bigvee Z \geq x \geq y$, i.e. we have the thesis.
On the other hand if $\eta = (\eta)^{\vartriangle}$ then $\eta \smallsetminus (\eta)^{\vartriangle} = \varnothing$ and therefore the intersection is $\varnothing$.

Example 3.30. Consider the following lattice where the circled points, $\odot$ and $\circledcirc$, are the points in the closure δ. In particular the point $\circledcirc$ is the point in the lattice that makes join-uniformity to fail, i.e. it is $\delta_{\vartriangle}$.



It is sufficient to erase y to get a join-uniform closure.

As noticed in the beginning of this section all these results can be dualized for any $\delta \in uco(L)$, giving a method for making simplifications invertible by using a transformer $(\cdot)^{\triangledown} \in uco(uco(uco(C)))$. In this way we obtain the algebra of domain transformers, depicted in Fig. 3.7, where the dashed arrows represent the transformation $(\cdot)^{\triangledown}$ and its dual.

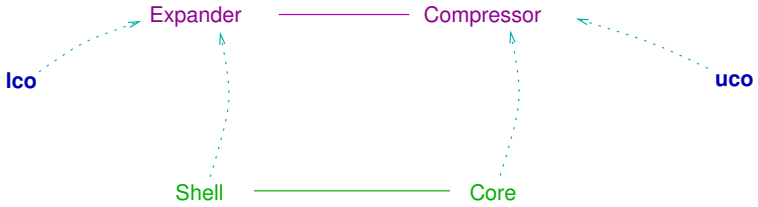


Fig. 3.7. The algebra of transformers.

3.4 The 3D geometry of completeness transformers

Note that, all the results that we have in order to reverse a completeness domain transformers are proved for the forward completeness ($\mathcal{F}$-completeness). Naturally we are interested in doing the same for $\mathcal{B}$-completeness. In [60] the authors proved that if the function f, w.r.t. we refine the domain, is additive then

$$\mathcal{R}^{\mathcal{B}}_{f} = \mathcal{R}^{\mathcal{F}}_{f+} \quad (\text{analogously } \mathcal{E}^{\mathcal{B}}_{f} = \mathcal{E}^{\mathcal{F}}_{f+})$$

where $\mathcal{R}_f^{\mathcal{B}}$ denotes the $\mathcal{B}$-completeness refinement for f and $\mathcal{E}_f^{\mathcal{B}}$ denotes the completeness core for f (see also page 38). This means that when we have to solve a problem of $\mathcal{B}$-completeness for an additive function then we can solve the $\mathcal{F}$-completeness for the right adjoint function, obtaining the searched result. It's clear that it would be much more important to translate a $\mathcal{B}$-completeness problem in an $\mathcal{F}$-completeness problem for a generic (continuous) function.

Whenever a function $f : C \to C$ is not additive, then we have that the map $f_+ \stackrel{\text{def}}{=} \lambda x.\ \max\left\{\ y \,\middle|\, f(y) \leq x\ \right\}$ is a relation on $C \times C$, or analogously a function on $C \to \wp(C)$. We want to prove that also in this case the function f^+ allows us to move from $\mathcal{B}$ to $\mathcal{F}$ completeness.

Let $\langle C, \leq \rangle$ be a complete lattice ordered by $\leq$. Let's consider the standard preorder on $\wp(C)$ $\preccurlyeq$: Let $X, Y \in \wp(C)$ then $X \preccurlyeq Y$ if $\forall x \in X. \exists y \in Y\ .\ x \leq y$. Our aim is to describe the function f_+ as the right adjoint of a transformation of f. In order to obtain this we have to define a partial order, since a preorder is not sufficient for defining an adjunction. For this reason note that the image of f_+ is not a generic subset of C but it is always an anti-chain of C. Therefore, if $\mathcal{A}c(C) \stackrel{\text{def}}{=} \left\{\ X \in \wp(C) \,\middle|\, X \text{ is an anti-chain}\ \right\}$, then we can prove the following proposition:

Proposition 3.31. *Let $\langle C, \leq \rangle$ a complete lattice,*

$$\langle \mathcal{A}c(C), \preccurlyeq \rangle \text{ is a partial ordered set.}$$

Proof. We have to prove that on $\mathcal{A}c(C)$ the order $\preccurlyeq$ is partial, namely that it is antisymmetric, being a preorder on $\wp(C)$. Let $X, Y \in \mathcal{A}c(C)$, then it is worth noting that $X = max(X)$ and that $Y = max(Y)$. Suppose now that $X \preccurlyeq Y$ and $Y \preccurlyeq X$ we want to prove that this implies that $X = Y$. These hypotheses say that $\forall x \in X. \exists y \in Y\ .\ x \leq y$ and $\forall y \in Y. \exists x \in X\ .\ y \leq x$.
Consider $x_1 \in X$, then there exists $y_1 \in Y$ such that $x_1 \leq y_1$. This implies that there exists $x_2 \in X$ such that $y_1 \leq x_2$, but this means that $x_1 \leq x_2$ which is absurd because X is an anti-chain, so it must be $x_1 = x_2 = y_1 \in Y$, namely $x_1 \in Y$. We proved in this way that $X \subseteq Y$.
Viceversa consider $y_1 \in Y$, then there exists $x_1 \in X$ such that $y_1 \leq x_1$. This implies that there exists $y_2 \in Y$ such that $x_1 \leq y_2$, but this means that $y_1 \leq y_2$ which is absurd because Y is an anti-chain, so it must be $y_1 = y_2 = x_1 \in X$, namely $y_1 \in X$. We proved in this way that $Y \subseteq X$, and therefore that $X = Y$.

Let $f : C \to C$, let's consider the following function:

$$\widetilde{f} \stackrel{\text{def}}{=} \lambda X.\ \bigvee \left\{\ f(x) \,\middle|\, x \in X\ \right\} : \mathcal{A}c(C) \to C$$

We can prove that this is the left adjoint of the function f_+.

Proposition 3.32. *Consider the complete lattices $\langle C, \leq \rangle$ and $\langle \mathcal{A}c(C), \preccurlyeq \rangle$ and the continuous function $f : C \to C$. Then*

$$C \underset{\widetilde{f}}{\overset{f_+}{\leftrightarrows}} \mathcal{A}c(C)$$

Proof. First of all we have to prove that both the function are monotone. Let's consider the function f_+, and take $x, y \in C$ such that $x \leq y$. This implies that $\{\ z \in C \,|\, f(z) \leq x\ \} \subseteq \{\ z \in C \,|\, f(z) \leq y\ \}$. Therefore, for each element $x' \in max\{\ z \in C \,|\, f(z) \leq x\ \}$ we have $x' \in \{\ z \in C \,|\, f(z) \leq y\ \}$, which means that there exists $y' \in Y\ .\ x' \leq y$, namely $f_+(x) \preccurlyeq f_+(y)$.
On the other hand, consider $\widetilde{f}$ and take $X, Y \in \mathcal{A}c(C)$ such that $X \preccurlyeq Y$. This means that $\forall x \in X \exists y \in Y\ .\ x \leq y$. We have to prove that $\bigvee\{\ f(x) \,|\, x \in X\ \} \leq \bigvee\{\ f(y) \,|\, y \in Y\ \}$. Note that if $\forall x \in X . \exists y \in Y\ .\ x \leq y$ then, by monotonicity of f, we have that $\forall x \in X . \exists y \in Y\ .\ f(x) \leq f(y)$. Therefore for each $x \in X$ we have $f(x) \leq \bigvee\{\ f(y) \,|\, y \in Y\ \}$, namely $\bigvee\{\ f(x) \,|\, x \in X\ \} \leq \bigvee\{\ f(y) \,|\, y \in Y\ \}$ which is $\widetilde{f}(X) \leq \widetilde{f}(Y)$.
Now we have to prove that they form a Galois connection. Consider $X \in \mathcal{A}c(C)$ and $x \in C$:

$$\begin{aligned} f_+\widetilde{f}(X) &= max\left\{\ y \in C \,\middle|\, f(y) \leq \widetilde{f}(X)\ \right\} \\ &= max\{\ y \in C \,|\, f(y) \leq \bigvee \bigcup_{x \in X} f(x)\ \} \supseteq X \end{aligned}$$

The last relation implies that $X \preccurlyeq f_+\widetilde{f}(X)$ because it is worth noting that $\subseteq \Rightarrow \preccurlyeq$.

$$\begin{aligned} &\widetilde{f}f_+(x) = \widetilde{f}(max\{\ y \in C \,|\, f(y) \leq x\ \}) = \bigvee\left\{\ f(z) \,\middle|\, z \in max\{\ y \in C \,|\, f(y) \leq x\ \}\ \right\} \\ &(*) \qquad \leq \bigvee\{\ f(z) \,|\, f(z) \leq x\ \} \leq x \end{aligned}$$

where $(*)$ holds since $\left\{\ f(z) \,\middle|\, z \in max\{\ y \in C \,|\, f(y) \leq x\ \}\ \right\} \subseteq \{\ f(z) \,|\, f(z) \leq x\ \}$.

Finally, we can prove that the function f_+ is exactly the one necessary in order to translate a $\mathcal{B}$-completeness problem in an $\mathcal{F}$-completeness problem.

Theorem 3.33. *Let C be a complete lattice and $f : C \xrightarrow{c} C$. Let us define the adjoint function $f_+ \stackrel{def}{=} \lambda x.\ max\{\ y \in C \,|\, f(y) \leq x\ \}$. Then*

$$\mathcal{R}^{\mathcal{B}}_f = \mathcal{R}^{\mathcal{F}}_{f_+} \quad (\textit{analogously } \mathcal{E}^{\mathcal{B}}_f = \mathcal{E}^{\mathcal{F}}_{f_+})$$

Proof.

$$\begin{aligned} \mathcal{R}^{\mathcal{B}}_f(X) &= \mathcal{M}(\textstyle\bigcup_{y \in X} max(f^{-1}(\downarrow y))) = \mathcal{M}(\textstyle\bigcup_{y \in X} max\{\ z \in C \,|\, f(z) \leq y\ \}) \\ &= \mathcal{M}(f_+(X)) = \mathcal{R}^{\mathcal{F}}_{f_+}(X) \end{aligned}$$

3.5 Discussion: The 3D scenario

In this chapter, we describe an algebra for abstract domain transformers, where the operations on abstract domains can be designed and classified. In particular, the first distinction is between *simplifications*, reducing domain's information, and *refinements*, improving the domain's information. Moreover, we notice how, in general, we can identify simplifications with upper closure operators, and refinements

with lower closure operators, on the lattice of abstract interpretations.
In this general setting, we show what it means to *reverse* a domain transformer. In particular, we show that this inversion can be modeled by using adjunctions. This implies that we can consider two possible ways for reversing abstract domain transformers: either computing the left adjoint, or computing the right one. At this point, we characterize reversible transformers with a particular attention to completeness refinements, and we study how to make a domain transformer right reversible. As it is shown also in Fig. 3.7, we don't have a method for making transformers left reversible, and this deserves further research.
The whole work is in particular instantiated to abstract domain completeness transformers. In this case, as shown in Fig. 3.8, we can add a new dimension, since we have two different completeness notions, and therefore transformers. The important thing is that, as shown in [60], we can move from one to the other completeness problem simply by adjunction.

Note that this algebra provides all the notions and the background, necessary in order to design, classify and relate new abstract domain transformers. Indeed, in the following of this thesis some aspects of what have been introduced here, will be used for understanding and relating the abstract domain transformers considered for weakening the notion of non-interference in language-based security and for classifying the secrecy level of programs, and of computational systems in general.

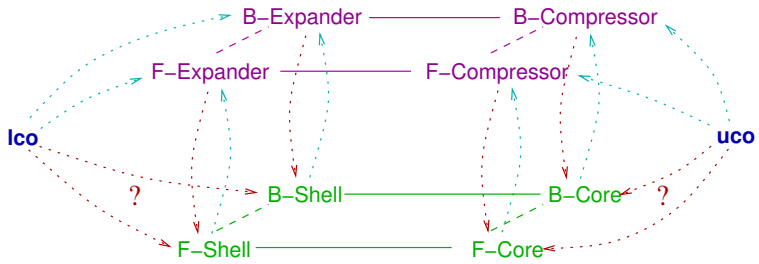


Fig. 3.8. The 3D algebra of transformers.

4

Computational Systems and Semantics

Wisdom does not inspect, but behold.
Henry David Thoreau

In this chapter, we introduce the computational systems that are considered in the following of this thesis and the possible semantics that can be used for modelling them.
In Sect 4.1, we describe the possible models that can be used for describing the meaning of computational systems. In particular, in Sect 4.1.1, we describe the model of computation that, from our point of view, gives the most concrete description of computation: We consider computational trees, in the following called tree semantics, which emboby both the branching and the linear aspects of computation. In this section, we also describe how it is possible to derive the trace semantics as abstraction of the tree semantics. In Sect 4.1.2, we recall the Cousot hierarchy of semantics, where different semantics are related by abstract interpretations, starting from the maximal trace semantics [27].
At this point, in Sect 4.2, we define the syntax and the operational semantics of the considered computational systems. In particular, we consider *imperative languages*, and starting from the simple deterministic fragment Imp [116] introduced in Sect. 4.2.1, we arrive to describe its non-deterministic and multi-threaded extensions. These computational systems are described together with an operational semantics given in terms of inference rules, which determine a transition system. Other important computational systems are *process algebras*, which model systems communicating through synchronization. In Sect 4.2.2, we introduce a particular process algebra defined for modelling security properties: Spa [47]. Also in this case, together with the syntactic definition of the system, we provide the operational semantics given in terms of a labelled transition system derived from a set of inference rules. Finally, we define *timed automata*, which are a well-known model

for real-time systems [5]. Again we describe the operational semantics of this kind of systems as a labeled transition system.

4.1 Semantics

In the following, we consider the tree models of computation as the most concrete possible semantics, that can be derived from a transition system associated with a computational system. Starting from this models we can obtain the trace semantics as the abstraction that forgets about the branching structure of the computation. At this point, we can follow Cousot's construction [27, 33], defining semantics, at different levels of abstraction, as the abstract interpretation of the maximal trace semantics of a transition system associated with each well-formed program.

4.1.1 Transition systems

The standard models used for computational systems are *transition systems*. A transition system is a pair $\langle \Sigma, \longrightarrow \rangle$, where Σ is the set of configurations that the system can have, called *states*, while $\longrightarrow \subseteq \Sigma \times \Sigma$ is a binary relation between a state and its possible successors, called *transition relation*. This relation is such that $s \longrightarrow s'$ means that s' is a configuration reachable from s in the system modeled by $\langle \Sigma, \longrightarrow \rangle$. In the following, Σ^+ and $\Sigma^\omega \stackrel{\text{def}}{=} \mathbb{N} \to \Sigma$ denote, respectively, the set of all the finite nonempty sequences, and the set of all the infinite sequences, of symbols in Σ. Given a sequence $\sigma \in \Sigma^\infty \stackrel{\text{def}}{=} \Sigma^+ \cup \Sigma^\omega$, its length is denoted by $|\sigma| \in \mathbb{N} \cup \{\omega\}$ and its i-th element is denoted by σ_i. Moreover, $\sigma_\vdash$ will denote σ_0, and when $|\sigma| = n \leq \omega$, then $\sigma_\dashv$ will denote σ_{n-1}. A non-empty finite (infinite) *trace* σ is a finite (infinite) sequence of program states, where two consecutive elements are in the transition relation $\longrightarrow$, i.e., for all $i < |\sigma|$: $\sigma_i \longrightarrow \sigma_{i+1}$.
A *labelled transition system* is a triple $\langle \Sigma, \longrightarrow, \lambda \rangle$, which is a transition system $\langle \Sigma, \longrightarrow \rangle$ equipped with a *labelling function* $\lambda : \Sigma \times \Sigma \to Lab$, where Lab is a set of *labels* (also called *actions*) for transitions. Given a transition $s \longrightarrow s'$, we have that $\lambda(s, s')$ is the label associated with the transition. A *run* in the system is a sequence $\rho = \langle s_0 \longrightarrow^{l_0} s_1 \longrightarrow^{l_1} \ldots \rangle$, such that for each i, $s_i \in \Sigma$, $l_i = \lambda(s_i, s_{i+1}) \in Lab$, and $s_i \longrightarrow s_{i+1}$. The projections of runs onto labels are called *traces* of the labelled transition system. The set of traces of a transition system is denoted by $Tr(\langle \Sigma, \longrightarrow, \lambda \rangle) \stackrel{\text{def}}{=} \left\{ \langle l_0, l_1 \ldots \rangle \,\middle|\, \forall i.\, s_i \longrightarrow^{l_i} s_{i+1} \right\}$.
Given a transition system, we can extend it to a labelled one by defining a labelling function λ. In particular, we can note that the action associated with any transition is the transformation of the state of the machine, namely we can think of labelling a transition with the reached state. Therefore $\langle \Sigma, \longrightarrow \rangle$ can be extended to the system $\langle \Sigma, \longrightarrow, \lambda \rangle$, where $Lab \stackrel{\text{def}}{=} \Sigma$ and $\lambda : \Sigma \times \Sigma \to \Sigma$ is defined as $\lambda(s, s') \stackrel{\text{def}}{=} s'$. In this case runs and traces coincide.

Tree semantics.

Consider labeled transition systems $\langle \Sigma, \longrightarrow, \lambda \rangle$, where Σ is a set of states, $\longrightarrow$ is a transition relation on Σ, and $\lambda : \Sigma \times \Sigma \to Act$ is a labeling function.
In the following, we will model computational systems by considering the tree generated by the corresponding transition system. We assume that for any $r \in \Sigma$: $\left|\left\{\ s \in \Sigma \,\middle|\, r \longrightarrow s\ \right\}\right| < \omega$, allowing finite branching. In particular, we consider the set of trees generated by a transition system $\langle \Sigma, \longrightarrow \rangle$, denoted by $\mathbb{T}_\Sigma$. We recall that $t \in \mathbb{T}_\Sigma$ is the tuple $t = \langle r, t_1, \ldots, t_n \rangle$, where $r \in \Sigma$ is the *root* of the tree, i.e., $r = root(t)$, and for each $i \leq n \in \mathbb{N}$ we have $t_i \in \mathbb{T}_\Sigma$. Moreover, we define the notion of *height* of a tree t, denoted $h(t)$, inductively: Let ε be the empty tree, then $h(\varepsilon) = 0$, $h(\langle r, \varepsilon \rangle) = 1$ and $h(\langle r, t_1, \ldots, t_n \rangle) = 1 + \max\left\{\ h(t_i) \,\middle|\, i \leq n\ \right\}$. We can also characterize the *leaves* of a tree inductively: $leaves(\varepsilon) = \varnothing$, $leaves(\langle r, \varepsilon \rangle) = r$ and $leaves(\langle r, t_1, \ldots, t_n \rangle = \bigcup_{i \leq n} leaves(t_i)$. In the following, we denote by $t_{\rfloor_n}$ the subtree of t, with height n, obtained taking t cut at height n. A *run* of the system is a path on the tree starting from the root, while a *trace* is the sequence of actions corresponding to a *run*. When t' is subtree of t we will write $t' \preccurlyeq t$. If, in particular, there exists $n \leq h(t)$ such that $t' \equiv t_{\rfloor_n}$, then we write $t' \preccurlyeq_n t$. In the following, $\mathbb{T}^n_\Sigma = \left\{\ t \in \mathbb{T}_\Sigma \,\middle|\, h(t) = n\ \right\}$, $\mathbb{T}^\omega_\Sigma = \left\{\ t \in \mathbb{T}_\Sigma \,\middle|\, \forall n \in \mathbb{N} \,.\, \exists t_{\rfloor_n}\ \right\}$ and $\mathbb{T}^+_\Sigma = \bigcup_{n \in \mathbb{N}} \mathbb{T}^n_\Sigma$. We define finite and infinite semantics of a computational system P by considering finite and infinite computational trees: $\{\!|P|\!\}^{\dot{n}} = t \in \mathbb{T}^n_\Sigma$ if and only if $\forall \langle r, t_1, \ldots, t_k \rangle \preccurlyeq t, \forall i \leq k \,.\, r \longrightarrow t_i$, where $r \longrightarrow t$ denotes that there exists a transition in $\longrightarrow$ from r to the root of t. Let T be the set of final/blocking states, then we define $\{\!|P|\!\}^n = t \in \{\!|P|\!\}^{\dot{n}}$ iff $leaves(t) \subseteq T$ and $\{\!|P|\!\}^+ = \bigvee_{n \in \mathbb{N}} \{\!|P|\!\}^n$, where $\bigvee_i t_i$ returns the minimal tree t that contains all the trees t_i as subtrees. Finally, $\{\!|P|\!\}^\omega = t \in \mathbb{T}^\omega_\Sigma$ if and only if $\forall n \in \mathbb{N}, \forall \langle r, t_1, \ldots, t_k \rangle \preccurlyeq_n t, \forall i \leq k \,.\, r \longrightarrow t_i$. In the following, we will denote by $\mathbb{T}_\Sigma$ the set of all the trees on the action alphabet Σ.

4.1.2 Cousot's semantics hierarchy

In this section, we recall Cousot's hierarchy of semantics [27, 33]. Semantics, in the hierarchy, are derived as abstract interpretations of a more concrete operational semantics that associates a discrete transition system with each well-formed program. Consider a transition system, where the transition relation $\longrightarrow$ is denoted by τ. The *maximal trace semantics*[1] of the transition system [33] is $\tau^\infty \stackrel{\text{def}}{=} \tau^+ \cup \tau^\omega$, where if $T \subseteq \Sigma$ is a set of final/blocking states $\tau^{\dot{n}} = \{\sigma \in \Sigma^+ || \sigma | = n, \forall i \in [1, n) \,.\, \langle \sigma_{i-1}, \sigma_i \rangle \in \tau\}$, $\tau^\omega = \{\sigma \in \Sigma^\omega | \ \forall i \in \mathbb{N} \,.\, \langle \sigma_i, \sigma_{i+1} \rangle \in \tau\}$, $\tau^+ = \cup_{n>0} \{\sigma \in \tau^{\dot{n}} | \ \sigma_{\dashv} \in T\}$, and $\tau^n = \tau^{\dot{n}} \cap \tau^+$. In the following, we will use the *concatenation* operation between traces: The concatenation $\sigma = \eta^\frown \xi$ of the traces $\eta, \xi \in \Sigma^\infty$ is defined only if $\eta_{|\eta|-1} = \xi_0$. In this case, σ has length $|\sigma| = |\eta| + |\xi| - 1$

[1] Note that, if we consider a labelled transition system $\langle \Sigma, \longrightarrow, \lambda \rangle$, then we can simply define the maximal trace semantics as the set $Tr(\langle \Sigma, \longrightarrow, \lambda \rangle)$, i.e., the set of all the sequences of actions that the system can make.

and it is such that $\sigma_l = \eta_l$ for each $0 \leq l < |\eta|$, while $\sigma_{|\eta|-1+n} = \xi_n$ if $0 \leq n < |\xi|$. Moreover, if $\eta \in \Sigma^\omega$, then, for each $\xi \in \Sigma^\infty$, we have $\eta^\frown\xi = \eta$.

The semantics τ^∞ has been obtained in [33] as the least fixpoint of the monotone operator $F^\infty : \wp(\Sigma^\infty) \to \wp(\Sigma^\infty)$, defined on traces as $F^\infty(X) = \tau^1 \cup \tau^{\dot{2}} {}^\frown X$. This operator provides a bi-induction (induction and co-induction) on the complete lattice of the maximal trace semantics $\langle \wp(\Sigma^\infty), \sqsubseteq^\infty, \sqcap^\infty, \sqcup^\infty, \sqcap^\infty, \Sigma^+, \Sigma^\omega \rangle$, where $X \sqsubseteq^\infty Y$ if and only if $X \cap \Sigma^+ \subseteq Y \cap \Sigma^+$ and $Y \cap \Sigma^\omega \subseteq X \cap \Sigma^\omega$. This order, later called the *computational order*, allows to combine both least and greatest fixpoint process in a unique least fixpoint presentation: finite (terminating) traces are obtained by induction (*least fixpoint*) of F^∞ on $\langle \wp(\Sigma^+), \subseteq \rangle$, and infinite traces are obtained by co-induction (*greatest fixpoint*) on $\langle \wp(\Sigma^\omega), \subseteq \rangle$, which corresponds to the *least fixpoint* of F^∞ on $\langle \wp(\Sigma^\omega), \supseteq \rangle$. In this case: $\tau^\infty = lfp^{\sqsubseteq^\infty}_{\Sigma^\omega} F^\infty$ (see [27, 33] for details). Cousot proved also that the natural trace semantics can be calculated as the *greatest fixpoint*, of the same function, on the domain with the usual inclusion order, here called *approximation order*, namely $\tau^\infty = gfp^{\subseteq}_{\Sigma^\infty} F^\infty$. Note that, this maximal trace semantics can be seen as an abstraction of the tree semantics. Indeed, if we consider the function $\alpha_{\mathtt{T}} : \mathbb{T}_\Sigma \to \wp(\Sigma^\infty)$ defined as $\alpha_{\mathtt{T}}(t) \stackrel{\text{def}}{=} \left\{ \sigma \,\middle|\, \sigma \text{ trace in } t \right\}$, then the maximal trace semantics of a program P is $\langle\!|P|\!\rangle \stackrel{\text{def}}{=} \alpha_{\mathtt{T}}(\{\!|P|\!\})$.

All the semantics, in the hierarchy, are derived as abstract interpretation of the trace-based semantics. In particular, each semantics in natural style corresponds to a suitable abstraction of the basic natural trace-based semantics τ^∞. In the following we denote by *Nat* the identical abstraction of the maximal trace semantics.

Relational semantics.

The relational semantics $\mathcal{R}^\infty$ associates, with program traces, an input-output relation by using the bottom symbol, $\perp \notin \Sigma$, to denote non-termination. This corresponds to an abstraction of the maximal trace semantics, where intermediate computation states are ignored. The abstraction function $\alpha^{\mathcal{R}}$, that allows to get the relational semantics as abstraction of the maximal trace one, i.e., $\mathcal{R}^\infty = \alpha^{\mathcal{R}}(\tau^\infty)$, is given in Table 4.1. The corresponding closure is:

$$Rel(X) \stackrel{\text{def}}{=} \gamma^{\mathcal{R}}\alpha^{\mathcal{R}}(X) = \left\{ \sigma \in \Sigma^+ \,\middle|\, \exists \delta \in X^+ \,.\, \sigma_\vdash = \delta_\vdash \ \wedge \ \sigma_\dashv = \delta_\dashv \right\} \cup \left\{ \sigma \in \Sigma^\omega \,\middle|\, \exists \delta \in X^\omega \,.\, \sigma_\vdash = \delta_\vdash \right\}$$

Denotational semantics.

The denotational semantics $\mathcal{D}^\infty$ abstracts away from the history of computations, by considering input-output functions. This semantics is isomorphic to relational semantics. The abstraction function $\alpha^{\mathcal{D}}$, that allows to get the denotational semantics as abstraction of the relational one, i.e., $\mathcal{D}^\infty = \alpha^{\mathcal{D}}(\mathcal{R}^\infty)$, is given in Table 4.1. The corresponding closure operator on the trace semantics is:

$$Den(X) \stackrel{\text{def}}{=} \gamma^{\mathcal{R}}\gamma^{\mathcal{D}}\alpha^{\mathcal{D}}\alpha^{\mathcal{R}}(X) = \left\{ \sigma \in \Sigma^+ \,\middle|\, \exists \delta \in X^+ \,.\, \sigma_{\vdash} = \delta_{\vdash} \ \wedge \ \sigma_{\dashv} = \delta_{\dashv} \right\} \cup \left\{ \sigma \in \Sigma^\omega \,\middle|\, \exists \delta \in X^\omega \,.\, \sigma_{\vdash} = \delta_{\vdash} \right\}$$

In the following, given a program P, we will denote its denotational semantics by $[\![P]\!]$.

Weakest precondition semantics.

Dijkstra's predicate transformer $g\mathcal{W}p$ is represented as co-additive functions, denoting weakest-precondition predicate transformers [42]. We consider the program S, and a *post-condition* (set of desired final states) P, that we want to hold after the execution of S. The semantics consists in finding the weakest *pre-condition*, namely the biggest set of possible initial states, which allows the program to finish in P. The abstraction function $\alpha^{g\mathcal{W}p}$, that allows to get the weakest precondition semantics as abstraction of the denotational one, i.e., $g\mathcal{W}p = \alpha^{g\mathcal{W}p}(\mathcal{D}^\infty)$, is given in Table 4.1. The corresponding closure operator on the trace semantics is:

$$\begin{aligned} gWp(X) \stackrel{\text{def}}{=} \ & \gamma^{\mathcal{R}}\gamma^{\mathcal{D}}\gamma^{g\mathcal{W}p}\alpha^{g\mathcal{W}p}\alpha^{\mathcal{D}}\alpha^{\mathcal{R}}(X) = \\ & \left\{ \sigma \in \Sigma^+ \,\middle|\, \exists \delta \in X^+ \,.\, \sigma_{\vdash} = \delta_{\vdash} \ \wedge \ \sigma_{\dashv} = \delta_{\dashv} \right\} \cup \\ & \left\{ \sigma \in \Sigma^\omega \,\middle|\, \exists \delta \in X^\omega \,.\, \sigma_{\vdash} = \delta_{\vdash} \right\} \end{aligned}$$

Hoare's axiomatic semantics.

Similarly to the $g\mathcal{W}p$ semantics, in the Hoare axiomatic semantics we consider triples of the kind $\{Q\}\ S\ \{P\}$, and, in this case, we give semantics to the program S by finding all the pairs $\langle P, Q\rangle$ such that $\{Q\}\ S\ \{P\}$ is a valid Hoare triple [74]. Hoare's axiomatic semantics $g\mathcal{H}$ is represented as elements in tensor product domain, i.e., GC's, specifying the adjoint relation between weakest-precondition and strongest-postcondition in Hoare's triples $\{P\}\ C\ \{Q\}$. The abstraction function $\alpha^{g\mathcal{H}}$, that allows to get the axiomatic semantics as abstraction of the weakest precondition one, i.e., $g\mathcal{H} = \alpha^{g\mathcal{H}}(g\mathcal{W}p)$, is given in Table 4.1. The corresponding closure operator on the trace semantics is the same as the denotational semantics.

Each semantics in natural style may have a corresponding *angelic*, *demonic*, and *infinite* observable, which is again an abstraction. For each semantics, all the observables are derived as the fixpoints, in the computational order, of semantic functions obtained by applying the fixpoint transfer theorems [27].

Angelic.

The angelic trace semantics τ^+ is designed as an abstraction of the maximal trace semantics, and it is obtained by approximating sets, of possibly finite or infinite traces, with the set of finite traces only, i.e., $\tau^+ = \alpha^+(\tau^\infty)$ (see Table 4.2).
We denote by $\mathcal{R}^+$, $\mathcal{D}^+$, $\mathcal{W}lp$, and $p\mathcal{H}$ respectively the big-step relational semantics [99], angelic denotational, weakest-liberal precondition [43], and Hoare's partial

Semantics	Domain relation	Abstraction and Concretization
$\mathcal{R}^\infty = \alpha^{\mathcal{R}}(\tau^\infty)$	$\langle \wp(\Sigma^\infty), \subseteq \rangle$ $\underset{\alpha^{\mathcal{R}}}{\overset{\gamma^{\mathcal{R}}}{\leftrightarrows}}$ $\langle \wp(\Sigma \times \Sigma_\perp), \subseteq \rangle$	$\alpha^{\mathcal{R}}(X) = \left\{ \langle \sigma_\vdash, \sigma_\dashv \rangle \,\middle|\, \sigma \in X \cap \Sigma^+ \right\} \cup \left\{ \langle \sigma_\vdash, \perp \rangle \,\middle|\, \sigma \in X \cap \Sigma^\omega \right\}$ $\gamma^{\mathcal{R}}(Y) = \left\{ \sigma \in \Sigma^+ \,\middle|\, \langle \sigma_\vdash, \sigma_\dashv \rangle \in Y \right\} \cup \left\{ \sigma \in \Sigma^\omega \,\middle|\, \langle \sigma_\vdash, \perp \rangle \in Y \right\}$
$\mathcal{D}^\infty = \alpha^{\mathcal{D}}(\mathcal{R}^\infty)$	$\langle \wp(\Sigma \times \Sigma_\perp), \subseteq \rangle$ $\underset{\alpha^{\mathcal{D}}}{\overset{\gamma^{\mathcal{D}}}{\leftrightarrows}}$ $\langle \Sigma \longrightarrow \wp(\Sigma_\perp), \sqsubseteq \rangle$	$\alpha^{\mathcal{D}}(X) = \lambda s.\{s' \in \Sigma_\perp \mid \langle s, s' \rangle \in X\}$ $\gamma^{\mathcal{D}}(f) = \left\{ \langle x, y \rangle \,\middle|\, y \in f(x) \right\}$
$g\mathcal{W}p = \alpha^{g\mathcal{W}p}(\mathcal{D}^\infty)$	$\langle \Sigma \longrightarrow \wp(\Sigma_\perp), \sqsubseteq \rangle$ $\underset{\alpha^{g\mathcal{W}p}}{\overset{\gamma^{g\mathcal{W}p}}{\leftrightarrows}}$ $\langle \wp(\Sigma_\perp) \xrightarrow{\text{coa}} \wp(\Sigma), \sqsupseteq \rangle$	$\alpha^{g\mathcal{W}p}(f) = \lambda P.\ \left\{ s \in \Sigma \,\middle|\, f(s) \subseteq P \right\}$ $\gamma^{g\mathcal{W}p}(\Phi) = \lambda s\ .\ \left\{ s' \,\middle|\, s \notin \Phi(\Sigma_\perp \smallsetminus \{s'\}) \right\}$
$g\mathcal{H} = \alpha^{g\mathcal{H}}(g\mathcal{W}p)$	$\langle \wp(\Sigma_\perp) \xrightarrow{\text{coa}} \wp(\Sigma), \sqsupseteq \rangle$ $\underset{\alpha^{g\mathcal{H}}}{\overset{\gamma^{g\mathcal{H}}}{\leftrightarrows}}$ $\langle \wp(\Sigma) \otimes \wp(\Sigma_\perp), \supseteq \rangle$	$\alpha^{g\mathcal{H}}(\Phi) = \left\{ \langle X, Y \rangle \,\middle|\, X \subseteq \Phi(Y) \right\}$ $\gamma^{g\mathcal{H}}(H) = \lambda Y\ .\ \bigcup \left\{ X \,\middle|\, \langle X, Y \rangle \in H \right\}$

Table 4.1. Basic natural-style semantics as abstract interpretations

correctness semantics [74]. All these semantics have been proved, in [25], to be the angelic abstractions of the corresponding semantics in natural style. The basic angelic trace semantics is constructively derived as the least fixpoint, in the computational order, of a semantic function: $\tau^+ = lfp^{\subseteq}_{\varnothing} F^+$ where $F^+(X) = \tau^1 \cup \tau^{\dot{2}} \frown X$.

Demonic.

The demonic trace semantics, denoted as τ^∂, is derived from the maximal trace semantics by approximating non-termination by *chaos*, namely by the set of all the possible finite computations starting from the state that leads to non-termination. This corresponds to allowing the worst possible behavior of the program [27,40,43]. This semantics is obtained as abstraction of the natural semantics by using the function α^∂, i.e., $\tau^\partial = \alpha^\partial(\tau^\infty)$ (see Table 4.2). In this way, the demonic observable is defined on the domain $D^\partial = \alpha^\partial(\wp(\Sigma^\infty))$, which is such that $X \in D^\partial$ if and only if

$$\sigma \in X \cap \Sigma^\omega \ \Rightarrow\ \mathit{chaos}(\sigma_\vdash) \subseteq X \cap \Sigma^+$$

where $\mathit{chaos}(\sigma_\vdash) \stackrel{\text{def}}{=} \left\{ \delta \in \Sigma^+ \,\middle|\, \delta_\vdash = \sigma_\vdash \right\}$.
We denote by $\mathcal{R}^\partial$, $\mathcal{D}^\partial$, $\mathcal{W}p^\partial$, and $g\mathcal{H}^\partial$ the demonic relational, demonic denotational [10], demonic weakest-precondition and demonic Hoare's semantics. These

Semantics	Domain relation	Abstraction and Concretization
$\tau^+ = \alpha^+(\tau^\infty)$	$\langle \wp(\Sigma^\infty), \subseteq\rangle \underset{\alpha^+}{\overset{\gamma^+}{\leftrightarrows}} \langle \wp(\Sigma^+), \subseteq\rangle$	$\alpha^+(X) = X \cap \Sigma^+$ $\gamma^+(Y) = Y \cup \Sigma^\omega$
$\tau^\partial = \alpha^\partial(\tau^\infty)$	$\langle \wp(\Sigma^\infty), \subseteq\rangle \underset{\alpha^\partial}{\overset{\gamma^\partial}{\leftrightarrows}} \langle D^\partial, \subseteq\rangle$	$\alpha^\partial(X) \stackrel{\text{def}}{=} X \cup \bigcup \left\{ \mathit{chaos}(\sigma_\vdash) \,\middle|\, \sigma \in X \cap \Sigma^\omega \right\}$ $\gamma^\partial(Y) = Y$
$\tau^\omega = \alpha^\omega(\tau^\infty)$	$\langle \wp(\Sigma^\infty), \subseteq\rangle \underset{\alpha^\omega}{\overset{\gamma^\omega}{\leftrightarrows}} \langle \wp(\Sigma^\omega), \subseteq\rangle$	$\alpha^\omega(X) = X \cap \Sigma^\omega$ $\gamma^\omega(Y) = Y \cup \Sigma^+$

Table 4.2. Observable semantics as abstract interpretations

semantics have been proved, in [25], to be the demonic abstractions of the corresponding semantics in natural style. The basic demonic trace semantics is constructively derived as the least fixpoint, in the computational order, of a semantic function: $\tau^\partial = lfp^{\sqsubseteq^\partial}_{\Sigma^\infty} F^\partial$ where $F^\partial(X) = \tau^1 \cup \tau^{\dot{2}} \frown X$.

Infinite.

The infinite trace semantics, denoted τ^ω, is derived by observing non-terminating traces only, i.e., $\tau^\omega = \alpha^\omega(\tau^\infty)$ (see Table 4.2). The corresponding infinite semantics are denoted by $\mathcal{R}^\omega$, $\mathcal{D}^\omega$, $\mathcal{W}p^\omega$, and $g\mathcal{H}^\omega$. The basic infinite trace semantics is constructively derived as the greatest fixpoint, in the computational order, of a semantic function: $\tau^\omega = gfp^{\subseteq}_{\Sigma^\omega} F^\omega$ where $F^\omega(X) = \tau^{\dot{2}} \frown X$.

Weakest precondition.

The weakest precondition semantics for total correctness $\mathcal{W}p$, is modeled as a further abstraction of the natural trace semantics. This semantics considers only those computations that surely terminate, in other words, the weakest precondition is the largest set of initial states terminating in the given post-condition. This observable is obtained as abstraction of the $g\mathcal{W}p$ semantics: $\mathcal{W}p = \alpha^{\mathcal{W}p}(g\mathcal{W}p)$ where

$$\alpha^{\mathcal{W}p}(\Phi) = \Phi \mid_{\wp(\Sigma)}$$
$$\gamma^{\mathcal{W}p}(\Psi) = \lambda P \,.\, (\text{if } \bot \notin P \text{ then } \Psi(P) \text{ else } \varnothing)$$

and $\langle (\wp(\Sigma_\bot) \xrightarrow{\text{coa}} \wp(\Sigma)), \supseteq\rangle \underset{\mathfrak{A}^{\mathcal{W}p}}{\overset{\gamma^{\mathcal{W}p}}{\leftrightarrows}} \langle (\wp(\Sigma) \xrightarrow{\text{coa}} \wp(\Sigma)), \supseteq\rangle$.
The semantics $t\mathcal{H}$ is the Hoare's axiomatic abstraction of $\mathcal{W}p$, i.e., $t\mathcal{H} = \alpha^{g\mathcal{H}}(\mathcal{W}p)$.

The whole hierarchy[2], relating semantics styles and observables, is shown in Fig. 4.1, where lines and arrows denote, respectively, isomorphisms and strict abstractions between semantics.

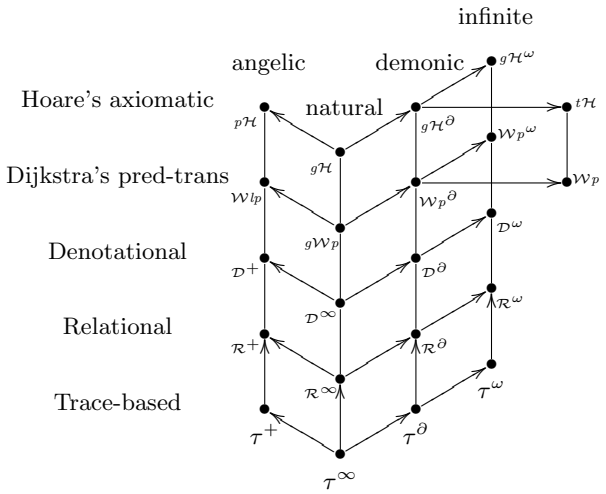


Fig. 4.1. Cousot's hierarchy.

4.2 Computational systems

In the following, we are going to introduce some very simple fragments of imperative languages: the deterministic fragment Imp [116], the non-deterministic fragment, with a non-deterministic choice operator, that we will call Nd-Imp, and the multi-threaded (or concurrent) fragment, with a parallel composition operator, that we will call Mt-Imp. Afterwords, we introduce a standard model for parallel systems communicating through synchronization: Process algebras [92]. In particular, we describe a particular process algebra introduced for modelling security properties: Spa [47]. Finally, we introduce timed automata, a standard model for real-time systems [5]. The semantics of all these computational systems is given by using a transition system, induced by a set of inference rules.

4.2.1 A simple imperative language

In this section, we introduce the syntax of a programming language, Imp [116], a small imperative language. Imp is called an *imperative* language because program

[2] In [58] the symmetric and relational structure of this hierarchy is studied and in [51] this hierarchy is extended in order to model also transfinite computations, providing a model for program slicing.

execution involves carrying out a series of explicit commands to change state. Formally, IMP's behaviour is described by rules which specify how its expressions are evaluated and its commands executed.

IMP*: The deterministic fragment.*

First of all, we list the syntactic sets associated with IMP:

- Values $\mathbb{V}$;
- Truth values $\mathbb{B} = \{\mathit{true}, \mathit{false}\}$;
- Variables *Var*;
- Arithmetic expression *Aexp*;
- Boolean expression *Bexp*;
- Commands *Com*

We assume that the syntactic structure of numbers is given. For other syntactic sets we have to say how their elements are built-up. We will use a variant of BNF as a way of writing down the rules of construction of the elements of these syntactic sets. We will use the following convention:

- m, n range over values $\mathbb{V}$;
- x, y range over variables *Var*;
- a ranges over arithmetic expression *Aexp*;
- b ranges over boolean expression *Bexp*;
- c ranges over commands *Com*;

So, we describe the formation rules for arithmetic expression *Aexp* by:

$$a ::= n \mid x \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \cdot a_1$$

For *Bexp* we have:

$$a ::= \mathit{true} \mid \mathit{false} \mid a_0 = a_1 \mid a_0 \leq a_1 \mid \neg b \mid b_0 \wedge b_1 \mid b_0 \vee b_1$$

Finally, for commands we have the following syntax:

$$c ::= \textbf{nil} \mid x := a \mid c_0; c_1 \mid \textbf{while } b \textbf{ do } c \textbf{ endw} \mid \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1$$

Note that

$$\begin{aligned} \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1 \equiv \textbf{while } b \textbf{ do } c_0; b := \mathit{false} \ \ \textbf{endw}; \\ \textbf{while } \neg b \textbf{ do } c_1; b := \mathit{true} \ \ \textbf{endw} \end{aligned}$$

Therefore, in the following we will, sometimes, consider the language IMP, omitting the control statement **if**.
As usual, $\mathbb{V}$ can be structured as a flat domain whose bottom element, $\perp$, denotes the value of not initialized variables. In the following we will denote by $\mathit{Var}(P)$ the set of variables of the program $P \in$ IMP. Let's consider the well-known (small-step) operational semantics of IMP in Table 4.3 [116]. The operational semantics

$$\langle \mathbf{nil}, s\rangle \longrightarrow s \qquad \frac{\langle e, s\rangle \longrightarrow n \in \mathbb{V}_x}{\langle x := e, s\rangle \longrightarrow \langle \mathbf{nil}, s[n/x]\rangle}$$

$$\frac{\langle c_0, s\rangle \longrightarrow \langle c_0', s'\rangle}{\langle c_0; c_1, s\rangle \longrightarrow \langle c_0'; c_1, s'\rangle} \qquad \frac{\langle c_1, s_0\rangle \longrightarrow \langle c_1', s_0'\rangle}{\langle \mathbf{nil}; c_1, s\rangle \longrightarrow \langle c_1', s_0'\rangle}$$

$$\frac{\langle b, s\rangle \longrightarrow \mathit{true},\ \langle c_0, s\rangle \longrightarrow \langle c_0', s'\rangle}{\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, s\rangle \longrightarrow \langle c_0', s'\rangle} \qquad \frac{\langle b, s\rangle \longrightarrow \mathit{false},\ \langle c_1, s\rangle \longrightarrow \langle c_1', s'\rangle}{\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, s\rangle \longrightarrow \langle c_1', s'\rangle}$$

$$\frac{\langle b, s\rangle \longrightarrow \mathit{true},\ \langle c, s\rangle \longrightarrow \langle c', s'\rangle}{\langle \mathbf{while}\ b\ \mathbf{do}\ c\ \mathbf{endw}, s\rangle \longrightarrow \langle c'; \mathbf{while}\ b\ \mathbf{do}\ c\ \mathbf{endw}, s'\rangle}$$

$$\frac{\langle b, s\rangle \longrightarrow \mathit{false}}{\langle \mathbf{while}\ b\ \mathbf{do}\ c\ \mathbf{endw}, s\rangle \longrightarrow \langle \mathbf{nil}, s\rangle}$$

Table 4.3. Operational semantics of IMP

naturally induces a transition relation on a set of states Σ, denoted $\longrightarrow$, specifying the relation between a state and its possible successors. In this transition system, states are representations of the memory, i.e., associations between variables and values. For this reason, in the following we will denote states as tuples of values, the values associated with the variables by the given state. Therefore, if $|Var(P)| = n$, then Σ is a set of n-tuples of values, i.e., $\Sigma = \mathbb{V}^n$. In sake of simplicity, we denote by $\mathbb{V}^x$ the set of values over which x can range, i.e., the domain of x.

The non-deterministic fragment.

A simple way to introduce some basic issues in order to obtain non-deterministic languages is to extend the simple imperative language IMP by an operation of non-deterministic choice. We define in this way the language ND-IMP, whose commands are defined in the following way:

$$c ::= \mathbf{nil} \mid x := a \mid c_0; c_1 \mid \mathbf{while}\ b\ \mathbf{do}\ c\ \mathbf{endw} \mid \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1 \mid c_0 \,\square\, c_1$$

Clearly. we have to extend the operational semantics with the rules for the non-deterministic choice:

$$\frac{\langle c_0, s\rangle \longrightarrow \langle c_0', s'\rangle}{\langle c_0 \,\square\, c_1, s\rangle \longrightarrow \langle c_0', s'\rangle} \qquad \frac{\langle c_1, s\rangle \longrightarrow \langle c_1', s'\rangle}{\langle c_0 \,\square\, c_1, s\rangle \longrightarrow \langle c_1', s'\rangle}$$

The multi-threaded fragment.

A simple way to introduce some basic issues in order to obtain parallel languages is to extend the simple imperative language IMP by an operation of parallel composition of commands. We define in this way the multi-threaded imperative language MT-IMP, whose commands are defined in the following way:

$$c ::= \mathbf{nil} \mid x := a \mid c_0; c_1 \mid \mathbf{while}\ b\ \mathbf{do}\ c\ \mathbf{endw} \mid \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1 \mid c_0 \parallel c_1$$

Now, we have to extend the operational semantics with the rules for the parallel composition:

$$\frac{\langle c_0, s\rangle \longrightarrow \langle c_0', s'\rangle}{\langle c_0 \parallel c_1, s\rangle \longrightarrow \langle c_0' \parallel c_1, s'\rangle} \qquad \frac{\langle c_1, s\rangle \longrightarrow \langle c_1', s'\rangle}{\langle c_0 \parallel c_1, s\rangle \longrightarrow \langle c_0 \parallel c_1', s'\rangle}$$

4.2.2 A process algebra: SPA

The *Security Process Algebra* (SPA for short) [47] is a slight extension of Milner's CCS [92], where the set of visible actions is partitioned into high level actions and low level ones, in order to specify multilevel systems. SPA syntax is based on the same elements as CCS, which are:

- A set $I = \{a, b, \ldots\}$ of *input* actions, a set $O = \{\overline{a}, \overline{b}, \ldots\}$ of *output* actions, and a set $\mathcal{L} = I \cup O$ of *visible* actions, ranged over by α;
- A function $\overline{\cdot} : \mathcal{L} \to \mathcal{L}$ such that if $a \in I$ then $\overline{a} \in O$, and if $\overline{a} \in O$ then $\overline{\overline{a}} \in I$. If $L \subseteq \mathcal{L}$ then $\overline{L} \stackrel{\text{def}}{=} \{\ \overline{a} \mid a \in L\ \}$;
- Two sets $Act_{\mathtt{H}}$ and $Act_{\mathtt{L}}$, of high and low level actions, such that $Act_{\mathtt{H}} = \overline{Act_{\mathtt{H}}}$ and $Act_{\mathtt{L}} = \overline{Act_{\mathtt{L}}}$, $Act_{\mathtt{H}} \cap Act_{\mathtt{L}} = \varnothing$ and $Act_{\mathtt{H}} \cup Act_{\mathtt{L}} = \mathcal{L}$;
- A set $Act = \mathcal{L} \cup \{\tau\}$ of actions (τ is the special unobservable, internal action), ranged over by μ;
- A set K of constants, ranged over by Z.

The syntax of SPA *agents* (or *processes*) is defined as follows [48]:

$$A ::= \mathbf{0} \mid \mu.A \mid A + A \mid A \parallel A \mid A\backslash\backslash L \mid A[f] \mid Z$$

where $L \subseteq Act \setminus \{\tau\}$ and $f : Act \to Act$ is such that $f(\overline{\alpha}) = \overline{f(\alpha)}$, $f(\tau) = \tau$. Moreover, for every constant Z there must be the corresponding definition $Z \stackrel{\text{def}}{=} A$, and A must be *guarded* on constants[3]. $\mathbf{0}$ is the empty process, which cannot do any action; $\mu.A$ can do the action μ and then behaves like A; $A_1 + A_2$ can choose to behave like A_1 or like A_2; $A_1 \parallel A_2$ is the parallel composition of A_1 and A_2, where the executions of the two systems are interleaved, possibly synchronized on complementary input/output actions, producing an internal action τ; $A\backslash\backslash L$ can execute all the actions A is able to do, provided they do not belong to L; if A can execute the action μ, then $A[f]$ executes the action $f(\mu)$. Starting from the defined syntax, we consider two specific restriction operators:

$$\begin{aligned} A\backslash L &\stackrel{\text{def}}{=} A\backslash\backslash L \cup \overline{L} \\ A\backslash_I L &\stackrel{\text{def}}{=} A\backslash\backslash L \cap I \end{aligned}$$

and the *hiding* operator of CSP [75]:

[3] The recursive substitution of all the non prefixed, i.e., not appearing in a context $\mu.A'$, constants in A with their definitions terminates after a finite number of steps.

Prefix	$\mu.A \xrightarrow{\mu} A$
Sum	$\dfrac{A_1 \xrightarrow{\mu} A_1'}{A_1 + A_2 \xrightarrow{\mu} A_1'} \qquad \dfrac{A_2 \xrightarrow{\mu} A_2'}{A_1 + A_2 \xrightarrow{\mu} A_2'}$
Parallel	$\dfrac{A_1 \xrightarrow{\mu} A_1'}{A_1 \parallel A_2 \xrightarrow{\mu} A_1' \parallel A_2} \qquad \dfrac{A_2 \xrightarrow{\mu} A_2'}{A_1 \parallel A_2 \xrightarrow{\mu} A_1 \parallel A_2'} \qquad \dfrac{A_1 \xrightarrow{\alpha} A_1' \quad A_2 \xrightarrow{\overline{\alpha}} A_2'}{A_1 \parallel A_2 \xrightarrow{\tau} A_1' \parallel A_2'}$
Restriction	$\dfrac{A \xrightarrow{\mu} A'}{A \backslash\backslash L \xrightarrow{\mu} A' \backslash\backslash L}$ if $\mu \notin L$
Relabelling	$\dfrac{A \xrightarrow{\mu} A'}{A[f] \xrightarrow{f(\mu)} A'[f]}$
Constant	$\dfrac{A \xrightarrow{\mu} A'}{Z \xrightarrow{\mu} A'}$ if $Z \stackrel{\text{def}}{=} A$

Table 4.4. Operational semantics of SPA

$$A/L \stackrel{\text{def}}{=} A[f_L] \text{ where } f_L(x) = \begin{cases} x \text{ if } x \notin L \\ \tau \text{ if } x \in L \end{cases}$$

Let $\mathcal{E}$ the set of SPA agents, given $A \in \mathcal{E}$, $\mathcal{L}(A)$ denotes the set of actions occurring syntactically in A. We can define the set of high level agents, i.e., $\mathcal{E}_{\mathtt{H}} \stackrel{\text{def}}{=} \{\ A \in \mathcal{E} \,|\, \mathcal{L}(A) \subseteq Act_{\mathtt{H}} \cup \{\tau\}\ \}$, and the set of low level agents, which is $\mathcal{E}_{\mathtt{L}} \stackrel{\text{def}}{=} \{\ A \in \mathcal{E} \,|\, \mathcal{L}(A) \subseteq Act_{\mathtt{L}} \cup \{\tau\}\ \}$.

The operational semantics is a labelled transition system $\langle \mathcal{E}, Act, \longrightarrow \rangle$, where the states are the terms of the algebra, and the transition relation is defined, as for CCS, by structural induction as the least relation generated by the axioms and inference rules reported in Table 4.4.

4.2.3 Timed Automata

In this section, we recall the notion of timed automata [5]. In the following, $\mathbb{R}$ is the set of real numbers and $\mathbb{R}^+$ the set of non-negative real numbers. A *clock* takes values from $\mathbb{R}^+$. Given a set $\mathcal{X}$ of clocks, a *clock valuation* over $\mathcal{X}$ is a function assigning a non-negative real number to every clock. The set of valuations over $\mathcal{X}$ is denoted $\mathcal{V}_{\mathcal{X}}$ and it is a set of total functions from $\mathcal{X}$ to $\mathbb{R}^+$. Given $\nu \in \mathcal{V}_{\mathcal{X}}$ and $\delta \in \mathbb{R}^+$, then $\nu + \delta$ is the map that with each clock x associates $\nu(x) + \delta$. Given a set $\mathcal{X}$ of clocks, a *reset* γ is a subset of $\mathcal{X}$. The set of all the resets is denoted by $\Gamma_{\mathcal{X}}$. Given a valuation $\nu \in \mathcal{V}_{\mathcal{X}}$ and a reset $\gamma \in \Gamma$, with $\nu \setminus \gamma$ we denote the

valuation:

$$\nu \setminus \gamma(x) = \begin{cases} 0 & \text{if } x \in \gamma \\ \nu(x) & \text{if } x \notin \gamma \end{cases}$$

Given a set $\mathcal{X}$ of clocks, the set $\Psi_{\mathcal{X}}$ of *clock constraints* over $\mathcal{X}$ are defined as follows:

$$\psi ::= \mathit{true} \mid \mathit{false} \mid \psi \vee \psi \mid \psi \wedge \psi \mid \neg\psi \mid x \; op \; t \mid x - y \; op \; t$$

where $x, y \in \mathcal{X}$, $t \in \mathbb{R}^+$, and $op \in \{<, >, \leq, \geq, =\}$. Clock constraints are evaluated over clock valuation. The satisfaction of a clock constraint $\psi \in \Psi_{\mathcal{X}}$ by a valuation $\nu \in \mathcal{V}_{\mathcal{X}}$ is denoted $\nu \models \psi$ and it is defined in the standard way.

Definition 4.1 (Timed automaton). *A timed automaton A is defined by a tuple $\langle Q, Q_0, \Sigma, \mathcal{X}, I, \mathcal{E}\rangle$, where: Q is a finite set of* states*, $Q_0 \subseteq Q$ is the set of initial states, Σ is a finite alphabet of* actions*, $\mathcal{X}$ is a finite set of clocks, I is a mapping that labels each location $q \in Q$ with any clock constraint in $\Psi_{\mathcal{X}}$, $\mathcal{E}$ is a finite set of* edges*. Each edge $e \in \mathcal{E}$ is a tuple in $Q \times \Psi_{\mathcal{X}} \times \Gamma_{\mathcal{X}} \times \Sigma \times Q$ such that if $e = \langle q, \psi, \gamma, \sigma, q'\rangle$ then q is the* source*, q' is the* target*, ψ is the constraint, σ is the action label and γ is the reset.*

The semantics of a timed automaton A is an infinite (labelled) transition system $\mathcal{S}(A) = \langle \Sigma, \longrightarrow, \lambda\rangle$, where Σ is the set of states and $\longrightarrow$ is the transition relation. The states Σ are the pairs $\langle q, \nu\rangle$, where $q \in Q$ is a state of A, and $\nu \in \mathcal{V}_{\mathcal{X}}$ of A. An initial state in $\mathcal{S}(A)$ is a state $\langle q, \nu\rangle$ such that $\nu(x) = 0$ for each $x \in \mathcal{X}$. At any state $\langle q, \nu\rangle$, A can perform an action labeling an outgoing edge e or to stay idle in the state, anyway we obtain the following labelling λ:

$$1.\ \frac{\delta \in \mathbb{R}^+, 0 \leq \delta' \leq \delta \;\Rightarrow\; \nu + \delta' \models I(q)}{\langle q, \nu\rangle \longrightarrow^{\delta} \langle q, \nu + \delta\rangle} \qquad 2.\ \frac{\langle q, \psi, \gamma, \sigma, q'\rangle \in \mathcal{E},\ \nu \models \psi}{\langle q, \nu\rangle \longrightarrow^{\sigma} \langle q', \nu \setminus \gamma\rangle}$$

Thus, a run in $\mathcal{S}(A)$ is $\rho = s_0 \longrightarrow^{l_0} s_1 \longrightarrow^{l_1} \ldots$ where for each i we have $s_i \in \Sigma$ and $l_i \in \Sigma \cup \mathbb{R}^+$. The *time sequence* $\overline{t_j}$ of the time elapsed from the state s_0 to the state s_j in the trace τ is:

$$\begin{cases} t_0 = 0 \\ t_{i+1} = t_i + \begin{cases} 0 \text{ if } l_i \in \Sigma \\ l_i \text{ otherwise} \end{cases} \end{cases}$$

The *event sequence* of the events occurring during the run σ, including the elapsed time, is $\langle l_0, t_0\rangle\langle l_1, t_1\rangle \ldots$. The *action sequence* (or *trace*) of the run σ is the projection of the event sequence of r on the pairs in the set $\{\ \langle l, t\rangle \mid l \in \Sigma\ \}$.

Definition 4.2 (Timed language). *Let $A = \langle Q, Q_0, \Sigma, \mathcal{X}, I, \mathcal{E}\rangle$ be a timed automaton, the* timed language *accepted by A is the set of the action traces of all the sequences possible in $\mathcal{S}(A)$, i.e., $Tr(\mathcal{S}(A))$. This language is denoted $\langle\!|A|\!\rangle$.*

Therefore we can say that two timed automata A_1 and A_2 are equivalent, i.e., $A_1 \approx A_2$, iff $\langle\!|A_1|\!\rangle = \langle\!|A_2|\!\rangle$.

5

Non-Interference in Language-based Security

When we imagine, we can only see,
when we know we can compare.
Jean-Jacques Rousseau

In the last decades, an important task of language based security is to protect confidentiality of data manipulated by computational systems. Namely, it is important to guarantee that no information, about confidential/private data, can be caught by an external viewer. In many fields, where protection of confidentiality is a critical problem, the standard way used to protect private data is access control: special privileges are required in order to read confidential data. Unfortunately, these methods allow to restrict accesses to data but cannot control propagation of information. Namely once the information is released from its container, it can be improperly transmitted without any further control. This means that the security mechanisms, such as signature, verification, and antivirus scanning, do not provide assurance that confidentiality is maintained during the whole execution of the checked program. This implies that, to ensure that confidentiality policies are satisfied, it becomes necessary to analyze how information flows within the executed program. In particular, if a user wishes to keep some data confidential, he might state a policy stipulating that no data visible to other users is affected by modifying confidential data. This policy allows programs to manipulate and modify private data, as long as visible outputs of those programs do not reveal information about these data. A policy of this sort is called *non-interference* policy [68], since it states that confidential data may *not interfere* with public data. Non-interference is also referred as *secrecy* [111], since confidential data are considered *private*, while all other data are public [39]. The difficulty of preventing a program P from leaking private information depends greatly on what kind of observations of P are possible [109]. If we can make *external observations* of P's running time, memory usage, and so on, then preventing leaks becomes very difficult. For exam-

ple, P could modulate its running time in order to encode the private information. Furthermore, these modulations might depend on low level implementation details, such as caching behaviours. But this means that it is insufficient to prove confinement with respect to an abstract semantics, every implementation detail, that affects running time, must be addressed in the proof of confinement. If, instead, we can only make *internal observations* of P's behaviour, the confinement problem become more tractable [109]. Internal observations include the values of program variables, and everything is observable internally, e.g. time in real-time systems.

In this chapter, we provide an excursus on the different notions of non-interference, in different computer science fields, and we describe the main approaches studied (see [104] for a survey). In the following, we first provide a brief background of the notion of non-interference, from the Lampson's formalization of the *confinement problem* [80] to the Cohen's *strong dependency* [19; 20], to the Goguen and Meseguer's definition of *non-interference* [68]. We conclude this part, introducing the semantic approach to non-interference of Joshi and Leino [78] and the PER model, applied to non-interference by Sabelfeld and Sands [106]. At this point, we provide a background about the existing techniques used for *enforcing* non-interference. Starting from the initial access control methods, such as the Bell and LaPadula model [13], we arrive to introduce the Denning and Denning information flow static analysis [38]. We conclude this part describing the Smith and Volpano security type system [114] and the axiomatic approaches to non-interference [7, 6]. Afterwards, we describe how this notion has been extended in order to cope with richer and more complex computational systems (e.g., non-deterministic and multi-threaded languages, process algebras and timed automata). We also introduce the notion of covert channel and we describe some existing solutions studied for avoiding this kind of information flows (e.g, timing and probabilistic channels, termination channels, and so on). Finally, we describe some existing weakenings of the notion of non interference, from the quantitative approaches that measures the information released [17, 84], to the definition of robust declassification [118], from the probabilistic approach characterizing how much statistical tests are necessary to disclose secrets [41], to the complexity-based approach which determines how complex is to disclose secrets [82].

5.1 Background: Defining non-interference

We have underlined above, how the problem of keeping confidential data private can be modeled as a non-interference problem, by stating that secure programs can manipulate and modify private data, as long as visible outputs of those programs do not reveal confidential information. In this section, we describe how non-interference can be defined in different fields of computer science, depending

on what the low level user is supposed to be able to observe. Before entering in the specific of the non-interference notion, we want to define what is a *security property*. Consider a set SC of *security classes* [38] (also called security domains in [86]), corresponding to disjoint classes of information. Suppose that each object e of a system is bound to a security class in SC, denoted $dom(e)$, which specifies the security class associated with the information represented by e. In general, a security domain can be, e.g., a group of users, a collection of files or a memory section. A *security property* is composed of a *non-interference relation* $\not\rightsquigarrow \subseteq \mathrm{SC} \times \mathrm{SC}$, which formalizes a *security policy* by stating which domains may not interfere with others, with a *definition of security* [86]. In the following, we simplify and consider only two domains, private/high `H` and public/low `L`, and the security policy which demands that `H` must not interfere with `L`, i.e., $\mathtt{H} \not\rightsquigarrow \mathtt{L}$.

In order to describe the background of the notion of *security* as absence of *flows* from private to public we have to go back to the seminal paper [80] where the notion of *confinement problem* is introduced (also known as *secrecy*). Consider a *customer* program and a *service* (host) program, the customer would like to ensure that the service cannot access (read or modify) any of his data, except those information to which he explicitly grants access (said *public*). In other words, the confinement problem consists in *preventing the results of the computation from leaking even partial information about confidential inputs*. Clearly, if the public data depends, in any way, on the private ones, then confinement becomes a problem. This strict relation between the confinement problem and the dependencies among data allows to describe the confinement problem as a problem of *non-interference* [68] by using the notion of *strong dependency* introduced in [19]. In the latter, the transmission of information is defined by saying that *information is transmitted over a channel when variety is conveyed from the source to the destination*. Clearly, if we substitute source with private and destination with public, then we obtain the definition of insecure information flow. More formally speaking, Cohen in [19] says that information can be transmitted from a to b during the execution of a system S, if by suitably varying the initial value of a (exploring the variety in a), the resulting value in b after S's execution will also vary (showing the variety is conveyed to b). The absence of strong dependency has been interpreted as non-interference in [68], where non-interference is defined as:

> *"One group of users [...] is noninterfering with another group of users if what the first group does [...] has no effect on what the second group of users can see".*

Starting from this informal definition, a non-interference policy which states that a group of users G does not interfere with another group of users G' is defined by saying that what any user $u \in G$ can observe when the machine is in the state representing the effect of an input string w on the states, starting from the initial state of the whole system, denoted by $[\![w]\!]_u$, is the same of what it can observe by

erasing all the actions of users in G'.
Therefore, we have that security, defined as presence of only secure information flows, is non-interference, which is absence of strong dependencies. These definitions are general and can be applied to different kind of computational systems, as we will see later on.
The notion of non-interference is used to stipulate policies of non-interference whenever a user wishes to keep some data confidential. This policy allows programs to manipulate and modify private data so long as visible outputs of those programs do not reveal confidential information. Therefore these policies stipulate that no data visible to other users is affected by confidential data [68].

5.1.1 Cohen's strong and selective dependency

Starting from the observation that in sequential programs information can be transmitted among variables, Cohen noted that the approaches previously studied were almost intuitionistic. His aim was that of providing a formal approach to information transmission so that information paths can be determined precisely given the formal semantics of a program. Moreover, the formal approach allows to answer more *selective* questions about information transmission. For example, we may not care if the output variable b reflects whether the input variable a is odd or even. However we might like to show that b depends on a in *no other way*. This leads clearly both to a semantic formalization of the problem and to a way for weakening the problem itself. In information theory, information can be transmitted from a source a to a destination b if a *variety* can be conveyed from a to b, namely as the result of program execution [19, 20]. This is exactly the idea used for defining *strong dependency*. The selective aspect of dependency, called *selective dependency*, comes from the observation that assertions, constraints on inputs of computation, can eliminate certain information paths, for example making a test always true. Cohen considers a simple imperative language with the usual semantics.

The idea for defining strong dependency is that of considering that if the input a may initially take on a number of different values, resulting in a number of different values in b after the execution of the program P, then we can say that b strongly depends on a. To show that information transmission is possible, we need only to find two different input values for a that yield different values for b after the execution of P. Therefore, adapting Cohen's definition to the security framework, we consider $\mathtt{L}$ as the set of public variables and $\mathtt{H}$ as the set of private ones. Then we say that the variables $\mathtt{L}$ are *strongly dependant* on $\mathtt{H}$ in the program P, $\mathtt{H} \rhd^P \mathtt{L}$, namely the program is not secure, if

$$\exists s_1, s_2 \,.\, s_1^{\mathtt{L}} = s_2^{\mathtt{L}} \ \wedge \ [\![P]\!](s_1)^{\mathtt{L}} \neq [\![P]\!](s_2)^{\mathtt{L}}$$

where s_1 and s_2 are states of P, namely tuples of values for the variables in P, and $s^{\mathtt{L}}$ is the tuple of values in s for the variables in $\mathtt{L}$.

Cohen realized that this definition, in some situations, was too strong, moreover he noted that adding input assertions reduces the information that can be transmitted. In general, any addition or strengthening of an input assertion may reduce (and never increase) information transmission [20]. This consideration leads him to the definition of the *selective dependency*: Often we are not interested in the fact that information is indeed transmitted from one object to another as long as specific properties, portions of the information, are protected. Consider the program

$$P \stackrel{\text{def}}{=} b := x + (a \ mod \ 4)$$

We can note that b does depend on a, i.e., $a \rhd^P b$, but only upon the last two bits of a. We can prove that the rest of a is protected from b by using strong dependency with a constraint, for example $\phi : \ a \ mod \ 4 = 3$. We can note that, even though P conveys variety from a to b, ϕ eliminates all the variety that is conveyed. Therefore, we define selective dependency for security (with a simplified notation). Consider as above $\mathtt{H}$ and $\mathtt{L}$ in a program P. We say that $\mathtt{L}$ is *selectively independent* from $\mathtt{H}$ in P, as regards the assertion ϕ, i.e., $\mathtt{H} \not\rhd^P_\phi \mathtt{L}$, if:

$$\forall s_1, s_2 \,.\, (\phi(s_1^{\mathtt{H}}) \ \wedge \ \phi(s_2^{\mathtt{H}}) \ \wedge \ s_1^{\mathtt{L}} = s_2^{\mathtt{L}}) \ \Rightarrow \ [\![P]\!](s_1)^{\mathtt{L}} = [\![P]\!](s_2)^{\mathtt{L}}$$

The role of ϕ is clearly that of characterizing which information we admit to flow from $\mathtt{H}$ to $\mathtt{L}$, indeed in the previous example any possible constraint on the value $a \ mod \ 4$ makes b selectively independent from a.

5.1.2 Goguen-Meseguer non-interference

In [68] the authors treat directly the problem of information transmission for enforcing program's security. Their approach to non-interference is intended to deal with both the abstract conceptual level of the security problem, where general concepts and methods are described, and the concrete modelling level, where actual systems are modeled in order to prove that they are secure in any sense. The definition introduced by Goguen and Meseguer is based on the notion of *security policy*, which defines the security requirements for a secure system. Therefore, in this context, security verification consists of showing that a given policy is satisfied by a given model. In general, information flow techniques attempt to analyze how users (or processes, or variables) can potentially *interfere* with other users. On the other hand, the security policy wants to say when users (or processes, or variables) must not interfere with other users. The purpose of a so-called *security model* is to provide a basis for determining whether or not a system is secure, and if not, for detecting its flaws.
In [68], the basic definition used to make non-interference precise considers systems as machines having a set of users U, a set of commands (changing-states) C, a set of read commands R, a set of outputs O and a set of internal states S, with initial state s_0. Moreover, there is a *next* function: $\mathbf{do} : S \times U \times C \longrightarrow S$, where

$\mathbf{do}(s, u, c)$ gives the next state after the user u executes the command c in the state s; and an *output* function: $\mathbf{out} : S \times U \times R \longrightarrow O$ where $\mathbf{out}(s, u, r)$ gives the result of a user u executing a read command r in the state s. Therefore, output commands have no effect on states, and state commands produce no output. The *history* of a system is the sequence $w = \langle (u_1, c_1) \ldots (u_n, c_n) \rangle$ where all the pairs are of commands $c_i \in C \cup R$ with their users $u_i \in U$, since the initial startup of the system is in the state s_0. When reasoning about states we can omit all output commands from the history, since output commands do not affect states. Thus, the state reached after the execution, in the system, of a sequence of state commands, starting from the initial state s_0, is given by the function $\mathbf{do}^* : S \times (U \times C)^*$ defined inductively by $\mathbf{do}^*(s, \text{empty}) = s$, $\mathbf{do}^*(s, \langle w(u, c) \rangle) = \mathbf{do}(\mathbf{do}^*(s, w), u, c)$. Let $[\![w]\!]$ denote the state reached from s_0 after the execution of the sequence w, i.e., $[\![w]\!] = \mathbf{do}^*(s_0, w)$.
A non-interference assertion expresses that a certain group G of users executing a certain set $\mathtt{H}$ of state transition commands does not interfere with, i.e., cannot be detected by, another group of users G' executing a set $\mathtt{L}$ of output commands; this is denoted $G, \mathtt{H} :| G', \mathtt{L}$. This assertion holds if and only if for each sequence $w \in (U \times C)^*$, each $v \in G'$, and each $l \in \mathtt{L}$ we have:

$$\mathbf{out}([\![w]\!], v, l) = \mathbf{out}([\![P_{G,\mathtt{H}}(w)]\!], v, l)$$

where $P_{G,\mathtt{H}}(w)$ is the sequence obtained from w by eliminating all occurrences of pairs (u, c) with $u \in G$ and $c \in \mathtt{H}$. Intuitively, this means that whatever any $v \in G'$ can tell by executing output commands in $\mathtt{L}$, everything looks as if the users in G had never executed any commands in $\mathtt{H}$. This is mostly the core work in [68] and it is a slightly different notion from the one introduced by Cohen. Indeed here we require that whenever private actions are executed the output observable behaviour has to be as if no private actions have been executed at all. In sequential programs the private actions are those where private variables are modified, and therefore in general it is a very strong requirement to extract computations where the execution of private actions are avoided. This impose the definition of a weaker notion of non-interference that we will call *standard* non-interference, and which is defined as the negation of Cohen's strong dependency, where private actions may interfere with the output behaviour, unless they do not convey a variety.

5.1.3 Semantic-based security models

As we have seen in the formalization of the Cohen's strong dependency, the problem of non-interference can be characterized by considering semantics of systems. A semantic approach has several features. First, it gives a more precise characterization of security than other approaches. Second, it applies to any programming constructs whose semantics are definable, for example, the introduction of non-determinism poses no additional problems. Third, it can be used for reasoning about indirect leaking of information through variation of the program behaviour

(e.g., whether or not the program terminates). Finally, it can be extended to the case where the high and the low security variables are defined abstractly, as function of the actual program variables [78]. We introduce here two main semantic approaches.

A semantic approach to secure information flow.

As we said above, a program is secure if any observation of the initial and final values of the low variables, denoted $l : \mathtt{L}$, do not provide any information about the initial value of the private variables, denoted $h : \mathtt{H}$ [78]. Assume that the adversary has knowledge of the program text and of the initial and final values of l. The idea of Joshi and Leino's semantic-based approach to language-based security is that of characterizing secure information flow as program equivalence, denoted by $\doteq$. They introduce a program $\mathtt{H}\,\mathtt{H} \stackrel{\text{def}}{=}$ "assign to h an arbitrary value". Consider a program P, for which we want to prove non-interference. The program $\mathtt{H}\,\mathtt{H}\,;P$ corresponds to run P after having set h to an arbitrary value; while the program $P;\mathtt{H}\,\mathtt{H}$ discards the final value of h resulting from the execution of P. Then a program P is said to be *secure* if

$$\mathtt{H}\,\mathtt{H}\ ;\ P;\ \mathtt{H}\,\mathtt{H}\ \doteq\ P\ ;\ \mathtt{H}\,\mathtt{H} \tag{5.1}$$

where $\doteq$ is the relational input/output semantic equality between programs, namely for each possible input the two programs have to show the same public output behavior. In order to understand this characterization, note that the occurrence of $\mathtt{H}\,\mathtt{H}$ after P on both the sides of the equality indicates that only the final values of l are of interest, whereas the occurrence of $\mathtt{H}\,\mathtt{H}$ before P on the left side of the equality indicates that the program starts with an arbitrary assignment to h. Clearly, the two programs are input/output equivalent provided that the final value of l, produced by P, does not depend on the initial value of h, which is indeed standard non-interference.

PER's model.

The semantic approach described above has also been equivalently formalized by using *partial equivalence relations (PER)* [106]. In this paper, the authors show how PER can be used to model dependencies in programs. Indeed, as we noted above, the problem of non-interference can be seen as absence of dependencies among data, where the meaning of dependency is given by Cohen [19]. The idea behind this characterization consists in interpreting security types as partial equivalence relations. In particular the variables $\mathtt{H}$ on D are interpreted by using the equivalence relation All_D, and $\mathtt{L}$ by using the relation Id_D, where for all $x, x' \in D$:

$$x\ All_D\ x' \qquad\qquad x\ Id_D\ x' \ \Leftrightarrow\ x = x'$$

The intuition behind the relations All_D and Id_D is that they represent the perspective of the user who does not have access to the high information. This user

can see the difference between distinct low data, but any high datum is indistinguishable from any other. This perspective can simply be generalized to multilevel security problems.

In order to use this model in the security framework, consider partial equivalence relation, namely equivalence relation which can fail the reflexive property. At this point, we can define a relation between functions. Let $Per(D)$ be the set of partial equivalence relations on D. Given $\mathtt{P} \in Per(D)$ and $\mathtt{Q} \in Per(E)$ we can define $(\mathtt{P} \multimap \mathtt{Q}) \in Per(D \longrightarrow E)$:

$$f\,(\mathtt{P} \multimap \mathtt{Q})\,g \;\Leftrightarrow\; \forall x, x' \in D\,.\,x\,\mathtt{P}\;x' \;\Rightarrow\; f(x)\,\mathtt{Q}\;g(x')$$

which is in general partial since it can fail reflexivity. Consider $\mathtt{P} \in Per(D)$, if $x \in D$ is such that $x\,\mathtt{P}\;x$ then we write $x : \mathtt{P}$. Therefore, if f is such that $\forall x, x' \in D\,.\,x\,\mathtt{P}\;x' \;\Rightarrow\; f(x)\,\mathtt{Q}\;f(x')$ then we write $f : \mathtt{P} \multimap \mathtt{Q}$. Finally for binary relations $\mathtt{P}$ and $\mathtt{Q}$, we define the relation $\mathtt{P} \times \mathtt{Q}$ by:

$$\langle x, y\rangle\,\mathtt{P} \times \mathtt{Q}\;\langle x', y'\rangle \;\Leftrightarrow x\,\mathtt{P}\;x' \;\wedge\; y\,\mathtt{Q}\;y'$$

At this point, we can formalize security in this model: let us distinguish, in the state s of P the values for low and private variables, i.e., $s = \langle s^{\mathtt{H}}, s^{\mathtt{L}}\rangle$ and let P be a program and $[\![P]\!]$ its semantics, then P is *secure* iff

$$[\![P]\!] : All \times Id \multimap All \times Id \;\equiv\; \forall s, t.\langle s^{\mathtt{H}}, s^{\mathtt{L}}\rangle\, All \times Id \langle t^{\mathtt{H}}, t^{\mathtt{L}}\rangle \;\Rightarrow\; [\![P]\!](s)\, All \times Id [\![P]\!](t)$$

where clearly $[\![P]\!](s)$ returns a state which is again a tuple of low and private values.

5.2 Background: Enforcing non-interference

Belief that a system is secure, with respect to confidentiality, should arise from a rigorous analysis showing that the system, as a whole, *enforces* the confidentiality policies of its users. In particular, we are interested in enforcing information flow policies. With the term *enforcement* we mean the checking process that ensures that a program does not reveal private information [80]. There are several approaches for checking non-interference. The standard method used for checking non-interference is to show that an attacker cannot observe any difference between two executions that differ only in the confidential input [69]. Clearly information flow analysis methods can be used for this purpose, but other approaches can be studied and developed. Statically, we can enforce non interference by using a type system. The idea is that of augmenting the type of variables and expressions with annotations that specify policies on the use of typed data, in order to enforce security policies at compile time. Other approaches define a semantic-based security model, providing powerful reasoning techniques. Checking non-interference is indeed an abstraction of the rigorous notion of non-interference that we want to enforce. In particular multi-level security can be expressed at three levels of abstraction [69]:

1. As a precise security policy, defined by a simple security requirement on languages, like the one given above;
2. As a set of general conditions on the transition function of the system that inductively guarantees its multi-level security, as in Bell-LaPadula model [13];
3. As a finite set of lemmas obtained by syntactic analysis of system specifications, such that if all the lemmas are true then any system satisfying these specifications is guaranteed multi-level secure with complete mathematical certainty.

The first formulation is the closest to intuition, since it expresses directly the constraints that should be enforced on the information flow, i.e., it expresses the policy that has to be enforced. The second formulation, which is obtained as *unwinding* [69] of the first one, reduces the proof of satisfaction of the policy to simpler conditions that, by inductive argument, guarantee that the policy holds. Finally, the third formulation is such that, if the process of derivation of lemmas from the policy has been proved mathematically sound, then it reduces the problem of obtaining full mathematical certainty about the security of a system to a form that can be checked by a theorem-prover.

5.2.1 Standard security mechanism

As we noted in the introduction, the standard mechanism used for checking non-interference is *access control*. Access control, which consists in a collection of access control lists and capabilities, is an important part of language-based security. For instance, it can be used when a file may be assigned access control permissions that prevent users, other than its owner, from reading the file. One of the most famous models, based on access control, is the Bell and LaPadula model [13] (see below).
The problem with access control is that it cannot control how data are propagated after they have been read from the file. For this reason, access control is not sufficient for guaranteeing certain kind of security policies, and information-flow control has to be used. Other common mechanisms are, for example, firewalls, encryption and antivirus software which can be used for protecting confidential information. The problem with these mechanisms is that they do not provide end-to-end security. For example, with encryption, we have not assurance that, after decryption, the confidentiality of data is respected. Another problem, related to access control mechanisms, is that it has been proved undecidable whether an access right to an object will "leak" to a process in a system whose access control mechanism is modeled by an access matrix [73].

Bell and LaPadula model.

Bell and LaPadula use finite-state machines to formalize their model [13]. They define the various components of the finite state machine defining what it means (formally) for a given state to be secure. In particular, they consider only the

transitions that can be allowed so that a secure state can never lead to an insecure one. This model is based on the access matrix model which is composed by an *access matrix* that decides in which mode each *subject* (user, program,...) can access to an *object* (files, variables,...). In addition to subjects and objects of the access matrix, the Bell and LaPadula model includes the security levels of the system: each subject has a *clearance* and each object has a *classification*. Each object has also a current security level which has not to exceed the subject's clearance. At this point, a set of rules, governing the transitions among states, is used in order to preserve the given security properties. Each rule is formally defined and it is provided together with a set of restrictions on the possible applications of the rule itself.

5.2.2 Denning and Denning Information flow static analysis

One of the first work which aim is to provide a mathematical framework suitable for formulating the requirement of secure information flow is [38]. The central component of this model is a lattice structure derived from the security classes and justified by the semantics of information flow. Security here means that *no unauthorized flow of information is possible*, which is another formulation for non-interference.

Consider an information flow model $\mathcal{F}$, defined as $\mathcal{F} = \langle N, P, \mathcal{S}, \oplus, \rightarrow \rangle$, where $N = \{a, b, \ldots\}$ is a set of objects, $P = \{p, q, \ldots\}$ is a set of processes, which are active agents responsible of information flow. $\mathcal{S} = \{A, B, \ldots\}$ is a complete lattice of *security classes* corresponding to disjoint classes of information, with least upper bound $\oplus$ and greatest lower bound denoted by $\otimes$. The idea behind these classes is that of modeling the security classification of objects. Each object a is bounded to a security class A which specifies the security class associated with the information stored in a. There are two kinds of binding: *static*, where the security classes associated with objects are constants, and *dynamic*, where the security classes may vary during the execution. The operator $\oplus$ is the class-combining operator, an associative and commutative binary operation that specifies, given two operand classes, the class in which the result of any binary function on values from the operand classes belongs. Finally $\rightarrow$ is a flow relation among classes. Given two classes A and B, we write $A \rightarrow B$ if information in class A is permitted to flow into class B. *Information is said to flow from class A to class B whenever information associated with A affects the value of information associated with B.* At this point, a flow model $\mathcal{F}$ is secure if and only if the execution of a sequence of operations cannot give rise to a flow that violates the relation $\rightarrow$.

Enforcing security.

The primary problem in guaranteeing security lies in detecting (and monitoring) all flows causing a variation of data [38]. Here, we find the first distinction between

implicit and *explicit* flows. Consider the statement **if** $a = 0$ **then** $b := 0$ **else nil**; if initially $b \neq 0$ then we can know something about a after the execution of the statement. For this reason the authors distinguish between *implicit* and *explicit* flows. Explicit flows are those due to the execution of any statement that directly transfer information among variables. Implicit flows to b occur when the result of executing or not a statement, that causes an explicit flow to b, is conditioned on the value of a guard. At this point, in order to specify the security requirements of programs causing implicit flows, it is convenient to consider an abstract representation of programs that preserves the flows but not necessarily the whole original structure. The abstract program S is defined recursively:

1. S is an elementary statement, i.e., an assignment;
2. There exist S_1 and S_2 such that $S = S_1; S_2$;
3. There exist $S_1, \ldots, S_m$ and an m-valued variable c such that $S = c : S_1, \ldots, S_m$.

where the third point defines conditional structures in which the value of a variable selects among alternative programs.
At this point, security is enforced by modeling implicit and explicit flows in the lattice of security classes and checking if these flows are allowed by the security policy chosen. Let us see how this is defined for the abstract program S described above. An elementary statement S is secure if any explicit flow caused by S is secure, namely if the value of b is derived in S from the values of $a_1, \ldots, a_m$ then $A_1 \oplus \ldots \oplus A_m \rightarrow B$ is allowed. $S = S_1; S_2$ is secure if both S_1 and S_2 are secure. Finally $S = c : S_1, \ldots, S_m$ is secure if each S_k is secure and all implicit flows from c are secure, namely let $b_1, \ldots, b_m$ be the objects into which S specifies explicit flows, then all the implicit flows are from c to each b_k and they are secure if $C \rightarrow B_1 \otimes \ldots \otimes B_m$ is allowed.
The authors use this model for generating a certification mechanism for secure information flow [39]. In particular, in the hypothesis of static binding, they easily incorporate the certification process into the analysis phase of a compiler and the mechanism is presented in the form of certification semantics - actions for the compiler to perform, together with usual semantic actions such as type checking and code generation, when a string of a given semantic type is recognized. This analysis has been widely studied and has been characterized as an extension of an axiomatic logic for program correctness in [7] (see Sect. 5.2.4).

5.2.3 Security type systems

A *security type system* is a collection of inference rules and axioms for deriving typing judgments, in particular it describes which security type is assigned to a program (or expression), based on the types of subprograms (or subexpressions). In [114] the Denning's approach is formulated as a type system, in such a way that all the well-typed programs are proved satisfy the non-interference property. A typing judgment has the form:

$$\gamma \vdash n : \tau \qquad \gamma \vdash x : \tau\ var \qquad \frac{\gamma \vdash e : \tau\ var}{\gamma \vdash e : \tau} \qquad \frac{\gamma \vdash x : \tau\ var \quad \gamma \vdash e : \tau}{\gamma \vdash x := e : \tau\ cmd}$$

$$\frac{\gamma \vdash c_1 : \tau\ cmd \quad \gamma \vdash c_2 : \tau\ cmd}{\gamma \vdash c_1; c_2 : \tau\ cmd} \qquad \frac{\gamma \vdash e : \tau \quad \gamma \vdash c : \tau\ cmd \quad \gamma \vdash c' : \tau\ cmd}{\gamma \vdash \textbf{if}\ e\ \textbf{then}\ c\ \textbf{else}\ c' : \tau\ cmd}$$

$$\frac{\gamma \vdash e : \tau \quad \gamma \vdash c : \tau\ cmd}{\gamma \vdash \textbf{while}\ e\ \textbf{do}\ c}$$

Table 5.1. Security type system

$$\frac{\tau \leq \tau'}{\vdash \tau \subseteq \tau'} \qquad \vdash \rho \subseteq \rho \qquad \frac{\vdash \rho \subseteq \rho', \quad \vdash \rho' \subseteq \rho''}{\vdash \rho \subseteq \rho''}$$

$$\frac{\vdash \tau \subseteq \tau'}{\vdash \tau\ cmd \supseteq \tau'\ cmd} \qquad \frac{\gamma \vdash p : \rho, \quad \vdash \rho \subseteq \rho'}{\gamma \vdash p : \rho'}$$

Table 5.2. Subtyping rules

$$\gamma \vdash p : \tau$$

which asserts that the program p has type τ with respect to the identifier typing γ. An identifier typing is a map from identifiers to types; it gives the type of any free identifier of p. So, for example, we have the inference rule $\gamma \vdash x : \tau$ if $\gamma(x) = \tau$. Let's start from the Denning's model [38]. The types of a systems are stratified into two levels. At one level are *data types*, denoted by τ, which are the security classes of $\mathcal{S}$, partially ordered by $\leq$. At the other level are *phrase types*, denoted by ρ. These include data types, assigned to expression, variable types of the form $\tau\ var$ and command types of the form $\tau\ cmd$. As expected, a variable type $\tau\ var$ stores information whose security class is τ or lower. Moreover, a command c has type $\tau\ cmd$ only if it is guaranteed that every assignment within c is made to a variable whose security class is $\tau\ var$ or higher. This is the confinement property ensuring secure implicit flows. In order to formalize this relation we have to extend the partial order $\leq$ on security classes to a subtype relation $\subseteq$ among types. A simplified version of the rules introduced in [114] are given in Table 5.1. In Table 5.2 we can find the subtyping rules. This system has been proved to be sound and therefore each program that can be typed in this system has only secure information flow. On the other hand, the system is not complete, which means that there are programs with only secure information flows and that cannot be typed in this system. For instance, the program $p \stackrel{\text{def}}{=} \textbf{if}\ h = 1\ \textbf{then}\ l := 0\ \textbf{else}\ l := 0$ with $l : \mathtt{L}$, $h : \mathtt{H}$ and $\mathtt{L} \leq \mathtt{H}$, has clearly only secure information flows but it cannot be typed in the system in Table 5.1.

5.2.4 The axiomatic approach

Another important approach for checking the existence of insecure information flows is the axiomatic one introduced, for the first time, in [7]. This approach uses a program *flow proof* constructed applying flow axioms and inference rules. An important aspect of this technique is that it can certify flows in both parallel and sequential programs. Moreover, once the flow proof for a program has been constructed, the proof can be used to validate a variety of flow policies. The idea of this work consists in using assertions of the kind $\{P\}\, S\, \{Q\}$, which means that if P is true before the execution of S, then Q is true after the execution of S, provided that S terminates. This is the standard notation used in correctness proofs, the difference is that P and Q here refers to classes rather than to values. In order to develop a *flow proof* of $\{P\}\ S\ \{Q\}$, the authors describe a deductive logic that allows to characterize the information flow semantics of statements. The inference rules used are of the form

$$\frac{A_1, \ldots, A_n}{B}$$

which means that if logical statements A_i are true, then so is B.

More recently, in [6], another Hoare-style logic has been defined in order to analyze information flow for confidentiality. In this case, confidentiality is treated as *independency* of variables [19], and program traces, potentially infinitely many, are abstracted, in the standard framework of abstract interpretation [28], by a finite set of variable independencies. The potentiality of this approach is that these variable independencies can be statically checked against the logic. Moreover, this method allows, once a program is deemed insecure, to explain why the program is insecure by statically generating counterexamples. The basic idea of this paper is to annotate the program in order to statically check independencies. This is achieved by using the Hoare-like logic described in Table 5.3, where $[x\#w]$ denotes that the current value of x is independent of the initial value of w, and where judgements are of the form $G \vdash \{T_1^\#\}\ C\ \{T_2^\#\}$. This judgement is interpreted by saying that if the independencies described in $T_1^\#$ hold *before* execution of C, then the independencies described in $T_2^\#$ will hold *after* the execution of C, provided that C terminates. In [6], the authors provide also a correctness result, which can be seen as the non-interference result for information flow. Indeed, with l and h interpreted as low and high respectively, suppose that $[l\#h]$ appears in the final set of independencies $T^\#$, after the execution of a program C. Then, any two traces in the execution of C, that have initial values that differ only on h, must agree on the current value of l. Moreover, if, on the other hand, the program is deemed insecure, i.e, $[l\#h]$ does not appear in the final set of independencies, then it means that l is dependant on h, and, in addition, the derived assertions allow to find a counterexample, i.e., two initial values of h that produce two different final values of l.

[Assign] $G \vdash \{T_0^\#\} x := e \{T^\#\}$

If $\forall [y\#w] \in T^\#$.
$(x \neq y \ \Rightarrow \ [y\#w] \in T_0^\#)$,
$(x = y \ \Rightarrow \ w \notin G \ \wedge \ \forall z$ free variable in e. . $[z\#w \in T_0^\#])$

[Seq] $$\frac{G \vdash \{T_0^\#\} C_1 \{T_1^\#\}, \ G \vdash \{T_1^\#\} C_2 \{T_2^\#\}}{G \vdash \{T_0^\#\} C_1; C_2 \{T_2^\#\}}$$

[If] $$\frac{G_0 \vdash \{T_0^\#\} C_1 \{T^\#\}, \ G_0 \vdash \{T_0^\#\} C_2 \{T^\#\}}{G \vdash \{T_0^\#\} \textbf{if } e \textbf{ then } C_1 \textbf{ else } \ C_2 \{T^\#\}}$$

If $G \subseteq G_0$,
$w \notin G_0 \ \Rightarrow \ \forall x$ free variable in e . $[x\#w] \in T_0^\#$

[While] $$\frac{G_0 \vdash \{T^\#\} C \{T^\#\}}{G \vdash \{T^\#\} \textbf{while } e \textbf{ do } C \{T^\#\}}$$

If $G \subseteq G_0$,
$w \notin G_0 \ \Rightarrow \ \forall x$ free variable in e . $[x\#w] \in T^\#$

[Sub] $$\frac{G_1 \vdash \{T_1^\#\} C \{T_2^\#\}}{G_0 \vdash \{T_0^\#\} C \{T_3^\#\}}$$

If $T_0^\# \subseteq T_1^\#, \ T_2^\# \subseteq T_3^\#, \ G_0 \subseteq G_1$

Table 5.3. An axiomatic logic for independencies

5.3 Non-interference for different computational systems

A major line of research in information flow purses the goal of defining non interference for the different computational models, and for accommodating the increased expressiveness of modern programming languages.

5.3.1 Deterministic systems: Imperative languages

As we underlined before, non-interference for *programs* essentially means that any possible variation of confidential (high/private) input does not cause a variation of public (low) output. This in particular means that each variable has a static attribute called *security level*. In [114] the confinement property for deterministic languages is defined as follows.

Definition 5.1. *A program P has the* non-interference *property if for all memories μ and ν such that $\mu(l) = \nu(l)$ for all low variables l, and such that P terminates*

successfully starting both from μ and ν, yielding, respectively, to μ' and to ν', then we have $\mu'(l) = \nu'(l)$ for all low variables l.

Basically, it says that altering the initial contents of private variables does not interfere with the final value of any low variable. For instance, if variable PIN is private and y is public then the following program does not preserve confinement, exactly as the program $y :=$ PIN:

$$\begin{array}{l}\textbf{while } \neg(\mathit{mask} = 0)\\ \quad \textbf{if } \neg(\text{PIN } \& \ \mathit{mask} = 0) \text{ (bitwise } \mathit{and})\\ \quad\quad y := y \mid \mathit{mask}; \quad \text{(bitwise } \mathit{or})\\ \quad \mathit{mask} := \mathit{mask}/2;\end{array}$$

If *mask* is a power of two, then it indirectly copies PIN to y, one bit at time [114].

Starting from the Cohen's seminal study of strong dependency [19], the notion of non-interference can be rigorously formalized using the programming-language semantics. Suppose that $s \in \Sigma$ is the denotation for states of programs, and that states, representing the tuples of values assigned to variables (i.e., representing memories), can be partitioned in order to distinguish the values of private variables from the values of public ones: $s = \langle s^{\mathtt{H}}, s^{\mathtt{L}} \rangle$. In general a program, starting from a state s can terminate in a state s' or can diverge. The denotational semantics of programs is the function that associates with each possible initial state the set of all the corresponding terminal state together with $\perp$, if the given initial state can lead to non-termination. Moreover, we can define an equivalence relation among states: $s_1 =_{\mathtt{L}} s_2$ iff $s_1^{\mathtt{L}} = s_2^{\mathtt{L}}$. Therefore, for a given semantic model $[\![P]\!]$ of the program P, non-interference can be formalized as follows: P is secure iff

$$\forall s_1, s_2 \in \Sigma \,.\, s_1 =_{\mathtt{L}} s_2 \ \Rightarrow \ [\![P]\!](s_1) =_{\mathtt{L}} [\![P]\!](s_2) \tag{5.2}$$

which is exactly the absence of strong dependency of public data from private ones [19]. For example the program

$$c \stackrel{\text{def}}{=} \textbf{if } h = 3 \textbf{ then } l := 5 \textbf{ else nil}$$

is clearly insecure since the high initial values 3 and 4 provides different results for the variable l: $\langle 4,1\rangle =_{\mathtt{L}} \langle 3,1\rangle$ but $[\![c]\!](4,1) = \langle 4,1\rangle$ while $[\![c]\!](3,1) = \langle 3,5\rangle$, where $\langle 4,1\rangle \neq_{\mathtt{L}} \langle 3,5\rangle$.
In general we can rewrite non-interference by saying that if two state share the same low values, then the behaviours of the program executed on these states are indistinguishable by the attacker. This means that the notion can be made parametric on what the attacker can really see. This is a key observation in order to abstract the notion of non-interference.

5.3.2 Non-deterministic and thread-concurrent systems

The natural extension of the notion of non-interference to non-deterministic systems is the notion of *possibilistic* non-interference [86]. As we have said before,

in order to prevent direct information flows, certain aspects of the system behaviour must not be directly observable by users who do not have the appropriate clearance. However, in general, an observer might still be able to deduce confidential information from other observations. In the worst case, the observer has complete knowledge of the system and can construct all the possible system behaviours which generate a given observation, trying to deduce confidential information from this set. The basic idea of possibilistic security is to demand that this set is so large that the observer cannot deduce confidential information since it cannot be sure which behaviour has actually occurred [86]. In [108] the confinement (non-interference) property for non-deterministic languages is defined as:

Definition 5.2. *A non-deterministic program P satisfies the* possibilistic non-interference *property if for all memories μ and ν such that $\mu(l) = \nu(l)$ for all low variables l, and P can terminate successfully starting from μ yielding to the final state μ', then there exists a state ν' such that P can terminate successfully starting from ν yielding ν' and $\mu(l) = \nu'(l)$ for all low variables l.*

It says that altering the initial contents of high variables does not interfere with the *set of possible final values* of any low variable [108]. The property rules out concurrent programs with information channels that exploit thread synchronization. In particular, we have a purely non-deterministic system if the scheduler of the system, that activates the threads, is characterized by the simple rule: *At each step, any thread can be selected to run for one step.* For instance, consider the following system:

Thread α:	Thread β:	Thread γ:
$y := x;$	$y := 0;$	$y := 1$

Suppose that x is a private binary variable, while y is public. Then the program satisfies the possibilistic non-interference property.

Possibilistic security properties.

Due the complex structure of non-deterministic systems, the notion of possibilistic non-interference given above, is not the only *confidentiality property* that can be defined on this kind of systems. The first attempts to provide a general theory in which uniformly define possibilistic security properties was through the use of *selective interleaving functions* [91]. In this paper, it is observed that possibilistic security properties fall outside of the Alpern-Schneider safety/liveness domain [4], since these properties are not properties of traces, i.e., trace sets, but properties of trace sets, i.e., sets of trace sets. In particular, possibilistic security properties are defined as *closure properties with respect to some functions that takes two traces and interleaves them to form a third trace* [91]. This theory is then used for studying how these security properties behave when systems are composed, i.e., if a system satisfying property X is composed with a system satisfying property

Y, using composition constructor Z, what properties will the composite system satisfy? In the following we will recall the principal security properties treated in this general theory.

Non-inference: Informally, non-inference requires that for any trace of the system, removing all the high level events, we obtain a trace that is still valid. More formally, if *purge*(τ) is the function that takes a trace τ and sets all high level inputs and outputs in τ to the empty value λ, then a system satisfies non-inference if the set of its traces is closed under the function *purge*, i.e., the image of *purge* is always contained in the set of valid traces of computation.

Generalized Non-inference: Informally, generalized non-inference requires that for any trace τ, it must be possible to find another trace σ such that the low level events of τ are equal to σ and σ has not high level inputs. More formally, if *input-purge*(τ) is the function that takes a trace τ and sets all high level inputs in τ to the empty value λ, then a system satisfies non-inference if the set of its traces is closed under the function *input-purge*.

Separability: Informally, separability holds if no interaction is allowed between high level and low level events. It is like having two separate systems, one running the high level processes, and one running the low level ones. More formally, if *interleave*(τ_1, τ_2) is the function that takes two traces τ_1 and τ_2 and returns the trace τ such that the high input and output of τ are taken in τ_1 and low input and output of τ are taken in τ_2, then a system satisfies separability if the set of its traces is closed under the function *interleave*.

Generalized Non-interference: Generalized non-interference holds if modifying a trace τ, inserting or deleting high level input, results in a sequence σ that can be transformed in a valid trace by inserting or deleting high level outputs. More formally, if *input-interleave*(τ_1, τ_2) is the function that takes two traces τ_1 and τ_2 and returns the trace τ such that the high input of τ are taken in τ_1 and low input and output of τ are taken in τ_2, then a system satisfies generalized non-interference if the set of its traces is closed under the function *input-interleave*.

This framework has been made more intuitive and general in [117], in order to model more security properties, such as *perfect security property* (PSP), which allows high level outputs to be influenced by low level events [117]. More recently, all these security properties have been modeled in a *modular structure* in [85], where they are obtained as combination of *basic security predicates*.

5.3.3 Communicating systems: Process algebras

The possibilistic notions of non-interference introduced in the previous section allows to consider non deterministic system, but are not adequate for treating non-interference in systems with the *synchrony* assumption: a system is composed of several components which have to proceed together at every time instant [47].

Synchrony is a basic feature, together with non-determinism, of *process algebras*, and probably, the most famous representative of this class is CCS [92]. In particular, in [47], the problem of studying secure information flows is considered in a particular process algebra, SPA (see Sect. 4.2.2), which is a slight extension of CCS. At this point, we recall the principal notions of non-interference defined on SPA in [47]. In particular there are two classes of definitions, depending on the equivalence of processes chosen: trace-based or bisimulation-based. In order to better understand the notions of non-interference that we are going to introduce, let's reformulate the idea of non-interference as follows: *Let G and G' be two user groups, given any input sequence γ, let γ' be its subsequence obtained by deleting all the actions of users in G; G is non-interfering with G' iff for every input sequence γ, the users of G' obtain the same output after the execution of γ and of γ'.*

Trace-based security properties.

Let us consider the trace-based equivalence of processes $\approx_T$, i.e., $A_1 \approx_T A_2$ iff the set of traces associated with A_1 is equal to the set of traces associated with A_2. Then the first extension of the notion of non-interference to SPA is the *Non-deterministic Non-Interference* (NNI), defined as follows:

$$A \in NNI \quad \Leftrightarrow \quad (A \backslash_I Act_{\mathtt{H}})/Act_{\mathtt{H}} \approx_T A/Act_{\mathtt{H}}$$

This notion requires that, when we avoid high level inputs, we obtain a trace whose projection on low level actions (i.e., the hiding of high level actions) is equal to the low level projection of a generic trace of actions of the system. A more restrictive form of NNI requires that, for every trace γ, the sequence γ', obtained deleting all the high level actions (input and output), is still a trace. This property is called *Strong NNI* (SNNI) and is defined as follows:

$$A \in SNNI \quad \Leftrightarrow \quad A/Act_{\mathtt{H}} \approx_T A \backslash Act_{\mathtt{H}}$$

The relation between these two notions is that, in SPA, $SNNI \subset NNI$. If, such as in CSP, we don't have distinction between inputs and outputs, then $NNI = SNNI$.

Another interesting notion of non-interference is *Non-Deducibility on Compositions* (NDC). A system is NDC if the set of its low level views cannot be modified by composing the system with any high level process. This property can be defined as follows:

$$A \in NDC \quad \Leftrightarrow \quad \forall \Pi \in \mathcal{E}_{\mathtt{H}} \; . \; A/Act_{\mathtt{H}} \approx_T (A \parallel \Pi) \backslash Act_{\mathtt{H}}$$

In [47] it is proved that $NDC = SNNI$.

Bisimulation-based security properties.

All the security notions introduced so far are based on the assumption that the semantics of a system is the set of its execution traces. In this section we show

that, in [47], these security properties have been rephrased on the finer notion of system behaviour called *weak bisimulation* (or observational equivalence) [92]. This extension was considered since trace semantics is rather weak, as it is unable to distinguish systems which give different observations to a user, even if they have the same traces. Here we recall the definition of weak bisimulation over SPA agents [47]. Let $A \stackrel{\mu}{\Longrightarrow} A'$ a short hand for $A \stackrel{\tau}{\longrightarrow}{}^* A_1 \stackrel{\mu}{\longrightarrow} A_2 \stackrel{\tau}{\longrightarrow}{}^* A'$, where $\stackrel{\tau}{\longrightarrow}{}^*$ means zero or more times τ. In the following $A \stackrel{\widehat{\mu}}{\Longrightarrow} E'$ stands for $A \stackrel{\mu}{\Longrightarrow} A'$ if $\mu \in \mathcal{L}$, for $A \stackrel{\tau}{\longrightarrow}{}^* A'$ if $\mu = \tau$. The following example shows that trace-based equivalence is weaker than bisimulation based equivalence. Indeed the two systems have the same set of traces but they are not bisimilar.

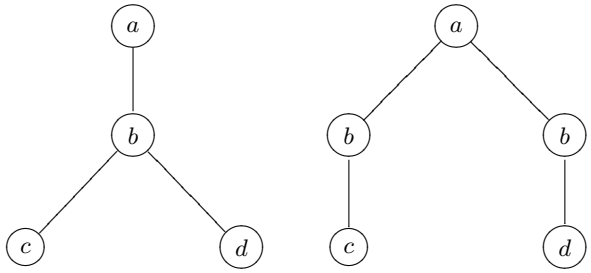


Fig. 5.1. Trace vs bisimulation equivalence

Definition 5.3. *A relation $R \subseteq \mathcal{E} \times \mathcal{E}$ is a* weak bisimulation *if it satisfies:*

- *Whenever $\langle A, B\rangle \in R$ and $A \stackrel{\mu}{\longrightarrow} A'$, then there exists $B' \in \mathcal{E}$ such that $B \stackrel{\widehat{\mu}}{\Longrightarrow} B'$, and $\langle A', B'\rangle \in R$;*
- *Whenever $\langle A, B\rangle \in R$ and $B \stackrel{\mu}{\longrightarrow} B'$, then there exists $A' \in \mathcal{E}$ such that $A \stackrel{\widehat{\mu}}{\Longrightarrow} A'$, and $\langle A', B'\rangle \in R$;*

Two SPA *agents $A, B \in \mathcal{E}$ are observationally equivalent, $A \approx_B B$, if there exists a weak bisimulation containing the pair $\langle A, B\rangle$.*

Note that $\approx_B$ is an equivalence relation, and that it is stronger than $\approx_T$. At this point in [47] the *Bisimulation NNI* (BNNI), *Bisimulation SNNI* (BSNNI) and the *Bisimulation NDC* (BNDC) are introduced simply by substituting $\approx_B$ for $\approx_T$ in their algebraic SPA-based characterizations. In particular we can give a characterization of BNDC equivalent to the simple substitution of the equivalence relation.

- $A \in BNNI$ iff $(A \backslash_I Act_{\mathtt{H}})/Act_{\mathtt{H}} \approx_B A/Act_{\mathtt{H}}$;
- $A \in BSNNI$ iff $A/Act_{\mathtt{H}} \approx_B A \backslash Act_{\mathtt{H}}$;
- $A \in BNDC$ iff $\forall \Pi \in \mathcal{E}_{\mathtt{H}} \,.\, A \backslash Act_{\mathtt{H}} \approx_B (A \parallel \Pi) \backslash Act_{\mathtt{H}}$.

All the relations among these notions are deeply studied in [47].

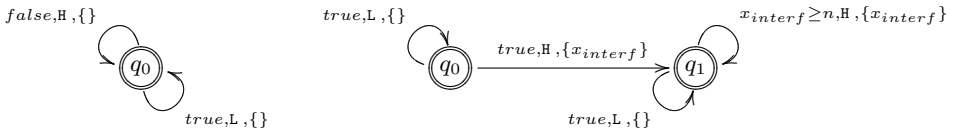


Fig. 5.2. The automata $Inhib_{\mathtt{H}}$ and $Interf^n_{\mathtt{H}}$.

5.3.4 Real-time systems: Timed automata

The most widespread models for real-time systems are timed automata (see Sect. 4.2.3). In [12], a new notion of non-interference for timed automata is introduced. The notion is based on high-level action delays magnitude and on equivalence of timed automata. Given a natural number n, the authors say that *high-level actions do not interfere with the system, considering minimum delay n, if the system behaviour in absence of high-level actions is equivalent to the system behaviour, observed on low-level actions, when high-level actions can occur with a delay between them greater than or equal to n.* Thus, the environment of the system does not offer high-level events separated by less that n times units, and if the property holds, there is no way for low-level users to detect any high-level action. The main improvement of this notion, if compared with untimed notions, is that time is observable and the property captures those systems in which the time delay between high-level actions cannot be used to construct illegal information flows.
Let A be a timed automaton over the alphabet of actions Σ and $\langle\!|A|\!\rangle$ the accepted language associated with A. We suppose that Σ is partitioned into two disjoint sets of actions $\mathtt{H}$ and $\mathtt{L}$ such that $\mathtt{H}$ is the set of the high-level actions, while $\mathtt{L}$ is the set of the low-level ones. First of all, consider an automaton A, we want to observe its behaviour in absence of high-level actions. In order to obtain this, we compose it in parallel with an automaton, called $Inhib_{\mathtt{H}}$, that does not allow the execution of high-level actions (see Fig. 5.2). In Fig. 5.2 we use the conventions that double-circled states are final, and q_0 is initial, moreover, an edge having as label a set of actions represents a set of edges, one for each action in the set, with the same clock constraint and clock reset. In the product $A||Inhib_{\mathtt{H}}$ the component A cannot have transition labeled by $h \in \mathtt{H}$ since $Inhib_{\mathtt{H}}$ never performs high-level actions (its constraints on high-level actions are false). Thus only low-level actions are executed.
Consider $Interf^n_{\mathtt{H}}$ in Fig. 5.2. This automaton allows the execution of high-level actions only when they are separated by at least n time units. Indeed, both the states can execute low-level actions without any restriction. But, if a high-level action occurs, then the automaton goes in state q_1 and reset the clock x_{interf}, which is reset by all high-level actions, and all high-level actions can be executed if x_{interf} is greater or equal than n. Namely a high-level action can be executed only if at least n time units have elapsed from the previous one.

Then an automaton A is said to be n *non-interfering* if:

$$(A||Interf_{\mathtt{H}}^{n})/\mathtt{H} \approx A||Inhib_{\mathtt{H}}$$

where the operator $/\mathtt{H}$ hides high-level actions, namely whenever the label of an edge is $\sigma \in \mathtt{H}$ it is replaced by ε, and $\approx$ is defined by: $A_1 \approx A_2$ iff $\mathcal{L}(A_1) = \mathcal{L}(A_2)$.

The notion of non-interference for timed automata can be equivalently characterized on languages [12]. Let $\langle\!|A|\!\rangle$ be the timed language accepted by A on a alphabet Σ and consider the following manipulation of languages:

$$\begin{aligned}
\langle\!|A|\!\rangle|_{\mathtt{L}} &\stackrel{\text{def}}{=} \left\{ \overline{\langle\sigma,t\rangle} \in \langle\!|A|\!\rangle \;\middle|\; \forall\langle\sigma_i,t_i\rangle \in \overline{\langle\sigma,t\rangle}\,.\,\sigma_i \in \mathtt{L} \right\} \\
\langle\!|A|\!\rangle/\mathtt{H} &\stackrel{\text{def}}{=} \left\{ \omega \;\middle|\; \begin{array}{l} \exists\overline{\langle\sigma,t\rangle} \in \langle\!|A|\!\rangle \text{ such that } \omega \text{ is the projection of } \overline{\langle\sigma,t\rangle} \\ \text{on the pairs } \left\{ \langle\sigma,t\rangle \,\middle|\, \sigma \in \mathtt{L} \right\} \end{array} \right\} \\
\langle\!|A|\!\rangle_{\mathtt{H}}^{n} &\stackrel{\text{def}}{=} \left\{ \overline{\langle\sigma,t\rangle} \in \langle\!|A|\!\rangle \;\middle|\; \begin{array}{l} \forall\langle\sigma_i,t_i\rangle,\langle\sigma_j,t_j\rangle \in \overline{\langle\sigma,t\rangle}\,.\, i \neq j,\ \sigma_i,\sigma_j \in \mathtt{H} \\ \Rightarrow\ |t_i - t_j| \geq n \end{array} \right\}
\end{aligned}$$

Namely, $\langle\!|A|\!\rangle|_{\mathtt{L}}$ avoids high-level actions, it takes only the traces of the system that make only low-level actions. On the other hand, $\langle\!|A|\!\rangle/\mathtt{H}$ hides the high-level actions, i.e., it executes them and then it hides them. Finally, $\langle\!|A|\!\rangle_{\mathtt{H}}^{n}$ selects only those traces where the high-level actions are distant at least n.
Then, in [12], a system is said to be *n-non-interfering* iff

$$\langle\!|A|\!\rangle_{\mathtt{H}}^{n}/\mathtt{H} = \langle\!|A|\!\rangle|_{\mathtt{L}}$$

5.4 Covert Channels

By covert channels we mean those channels that are not intended for information transfer at all [80]. The importance of studying these kind of channels lies on the fact that they pose the greatest challenge in preventing improper transmission leaks. There are several kind of covert channels [104]:

Implicit channels : Channels of information flow due to the control structure of a program;

Termination channels : Channels of information flow due to the termination or non-termination status of a program;

Timing channels : Channels of information flow due to the time at which an action occurs rather than due to the data associated with the action. The action may be termination;

Probabilistic channels : Channels of information flow due to the change of the probability distribution of observable data. These channels are dangerous when the attacker can run repeatedly a computation and observe its stochastic behaviour;

Resources channels : Channels of information flow due to the possible exhaustion of a finite, shared resource, such as disk memory;

The kind of covert channel, that may be created, depends on what the attacker/user can view of the computational system. This means that a computational system can be said to protect confidential information only with respect to a model of what attackers/users are able to observe of its execution.

5.4.1 Termination channels

Consider Definition 5.1 of non-interference for deterministic languages. In this definition it is said that the program has to "terminate successfully", starting from the given states. It is clear that, changes in high variables may cause the program to diverge, leaving unchanged the fact that the program can still satisfy the definition. This may make the property unsuitable in situations where this sort of behaviour can be observed. If, such as for PER model, the denotational semantics is used for defining non-interference, then we note that in case of non-termination denotational semantics associates with each state, leading to non-termination, the symbol $\perp$. In this way, Eq. 5.2 can be used also for defining termination-sensitive non-interference. Therefore, the PERs model can be simply adapted by considering domains of values enriched with the symbol $\perp$, i.e., $D_\perp$, and extending relations $\mathtt{R} \in \mathit{Per}(D)$ to $\mathtt{R}_\perp \in \mathit{Per}(D_\perp)$ naturally by adding the relation $\perp \mathtt{R}_\perp \perp$. In this way we make the definition insensitive to non-termination [106]. Namely *termination channels* are avoided simply by enriching the semantics. Note that, also in [1], where for the first time dependencies were given in term of PERs, for a calculus based on a variation of λ calculus, was shown that PERs capture termination sensitive security.
When non-interference is checked on the syntax, by typing secure programs (see Sect. 5.2.3), then it become necessary to enrich the type system in order to avoid termination channels [112]. In this paper, the authors show that termination flows can be handled with just a simple modification of the original type system in [114], based on the notion of *minimum type*. They say that a type τ is minimum if $\tau \leq \tau'$ for every type τ', to handle the covert flow arising from non-termination they merely change the typing rule for **while** b **do** c **endw** to require that b has minimum type. In other words, this means that this type system disallows high loops and require high conditionals have no loops in the branches.

5.4.2 Timing channels

Note that, in practice, non-termination cannot be distinguished from a very time-consuming computation, thus the termination channel can be viewed as an instance of the *timing channel*. Timing-sensitive non-interference can be formalized by considering Eq. 5.2, where the low view relation $=_{\mathtt{L}}$ is substituted by $\approx_{\mathtt{L}}$, which relates two behaviour iff both diverges or both terminate in the same number of

execution steps in low-equal final states [104]. In [113] the authors avoid timing channels in the type system by restricting high conditionals to have no loops in the branches and wrapping each high conditional in a *protect* statement whose execution is atomic. In [2] *program transformation* is used in order to close timing leaks. In particular, the "type" of a program C is its *low slice* $C_{\mathtt{L}}$, which is syntactically identical to C but only contains assignments to low level variables. All the assignments to high level variables are replaced with appropriate dummy commands with no effect on variables, therefore the low slice has the same observational behaviour as the original program with respect to low level variables. Finally, the usual type system is considered and, either the original program C is rejected (in case of a potential explicit or implicit insecure information flow) or accepted and transformed into the program $C_{\mathtt{L}}$ free of timing leaks.

5.4.3 Probabilistic channels

Probability-sensitive non-interference can be formalized in the Eq. 5.2 by replacing $=_{\mathtt{L}}$ with an equivalence relation $\approx_{\mathtt{L}}$ that relates two behaviours iff the distribution of low output is the same. Indeed, as we can see in the following example, possibilistic non-interference is not sufficient to prevent probabilistic information flows [90]. Consider, for example, the following multi-threaded system:

Thread α:	Thread β:
$y := x$;	$y := \mathit{random}(100)$

where $\mathit{random}(100)$ returns a random number between 1 and 100, and $x \in [1, 100]$. Then the program satisfies the possibilistic non-interference since regardless the initial value of x, the final value of y is a random number between 1 and 100. But with a probabilistic semantics, this is not good enough, because the final values of y are not equally probable, indeed the more probable value for y is the initial value of x [109]. Moreover, in multi-threaded systems, also the scheduler of processes may be probabilistic. In [113] the authors define a notion of probabilistic non-interference that captures the probabilistic information flows that may result from a uniform scheduler in a multi-threaded language. In [106] the authors considers PERs on probabilistic powerdomains in order to catch probabilistic flows. While in [105] the authors connect probabilistic security with probabilistic bisimulation [81], improving the precision of the previous probability-sensitive notions.

5.5 Weakening non-interference

The limitation of the notion of non-interference described so far, is that it is an extremely restrictive policy. Indeed, non-interference policies require that *any* change upon confidential data has not to be revealed through the observation of public data. There are at least two problems with this approach. On one side,

many real systems are intended to leak some kinds of information. On the other side, even if a system satisfies non-interference, some kinds of tests could reject it as insecure. These observations address the problem of *weakening* the notion of non-interference both characterizing the information that *is allowed* to flow, and considering *weaker* attackers that cannot observe any property of public data. Clearly, as we will show in this thesis, these are dual aspects of the same problem, and in the following sections we will describe the most relevant works in this direction.

5.5.1 Characterizing released information

As we have addressed above, real systems often do leak confidential information, therefore it seems sensible to try to measure that leakage as best as possible. The first work on this direction is [19], where the notion of *selective dependency* (see Sect. 5.1.1) is introduced. Selective dependency consists in a weaker notion of dependency, and therefore of non-interference, that identifies what flows during the execution of programs. More recently, in literature we can find several works that attack this problem from different points of view. A first approach consists in a quantitative (information theoretic) definition of information flows [17, 84]. Another relevant approach models the attacker's power by using equivalence relations, and by transforming these equivalence relations it characterizes the released information [118]. Afterwards, several papers treated the *declassification* of confidential information [83, 96, 103].

An information theory approach.

In [17], Shannon's information theory is used to quantify the amount of information a program may leak and to analyze in which way this depends on the distribution of inputs. In particular, the authors are interested in analysing how much an attacker may learn (about confidential information) by observing the input/output behaviour of a program. The basic idea is that all information in the output of a deterministic program has to come from the input, and what it is not provided by the low input has to be provided by the high input. Therefore, this work wants to investigate how much of the information carried by the high inputs to a program can be learned by observation of the low outputs, assuming that the low inputs are known. Now, since the considered language is deterministic, any variation of the output is due to a variation of the input. Hence, once we account for knowledge of the program's low inputs, the only possible source of *surprise* in an output is the interference from the high inputs. So, given a program variable X (or a set of program variables), let X^ι and X^ω be, respectively, the corresponding random variables on entry and exit from the program. In [17] the authors take as measure of the amount of leakage into X due to the program: $\mathcal{L}(X) = \mathcal{H}(X^\omega | L^\iota)$, where L is the set of low variables, this L^ι is the random variable describing the distribu-

tion of the program's non-confidential inputs, and $\mathcal{H}$ is the *entropy*[1]. Moreover, in [17], it is shown that there exists a more general characterization of the amount of information released that is appropriate even for languages with an inherently probabilistic semantics. In this case, they say that a natural definition of the leakage into X is the amount of information shared between the final value of X and the initial value of H, given the initial value of L: $\mathcal{L}' = \mathcal{I}(H^\iota; X^\omega | L^\iota)$, where $\mathcal{I}$ is the *conditional mutual information*[2] between H^ι and X^ω given knowledge of L^ι. This is essentially the definition used by Gray [71], specialized in a simpler semantic setting. In [17] it is also proved that, for deterministic languages $\mathcal{L} = \mathcal{L}'$.

Shannon's information theory is not the only approach, existing in literature, for quantifying information flow. Indeed in [84] the capacity of covert channels, i.e., the *information flow quantity*, is measured in terms of the number of high level behaviours that can be accurately distinguished from the low level point of view. The idea is that if there are N such distinguishable behaviours, then the high level user can use the system to encode an arbitrary number in the range $0, \ldots, N-1$ to send it to the low level user, in other words $\log_2 N$ bits of information are passed.

Declassification.

In the previous paragraph, we described a method that allows to quantify the amount of information released. In literature, there exists another important, more qualitative, approach whose aim is to discover *which* is the information that flows in order to *declassify* it for guaranteeing non-interference. *Declassifying* information means downgrading the sensitivity of data in order to accommodate with (intentional) information leakage[3]. Robust declassification has been introduced in [118] as a systematic method to drive declassification by characterizing what information flows from confidential to public variables. In particular, the observational attacker's capability is modeled by using equivalence relations as in PER models, and declassification of private data is obtained by manipulating these relations in a semantic-driven way. The semantics considered is the operational semantics, defined on a transition system. The idea is to consider *views* of the computational traces determined by the *observational capability* of the attacker. Hence, given a trace τ of computations of the system S, and given the $\approx$-view of τ (where $\approx$ is an equivalence relation), a view of τ is $\tau/\approx$ defined as follows: $\forall i < |\tau| . (\tau/\approx)_i = [\tau_i]_\approx$. The intuition is that a passive attacker (that cannot modify computations), who is

[1] Recall that, given a random variable X, let x ranges over the set of values which X may take and let $p(x)$ the probability that X take x, then $\mathcal{H}(X) = \Sigma_x p(x) \log \frac{1}{p(x)}$. The conditional entropy measuring the uncertainty in the variable X given the knowledge of the variable Y is $\mathcal{H}(X|Y) = \mathcal{H}(X,Y) - \mathcal{H}(X)$.

[2] Recall that, given the random variables X, Y and Z, the conditional mutual information between X and Y given the knowledge of Z is defined as $\mathcal{I}(X;Y|Z) = \mathcal{H}(X|Z) + \mathcal{H}(Y|Z) - \mathcal{H}(X,Y|Z)$.

[3] Note that this is similar to the Cohen's notion of selective dependency [19].

able to distinguish states up to $\approx$, will see the trace τ as a sequence of equivalence classes. Then, an *observation* of the system S, with respect to starting state σ and view $\approx$, is defined as: $Obs_\sigma(S, \approx) \stackrel{\text{def}}{=} \{\ \tau/\approx \mid \tau \text{ trace of } S \text{ starting in } \sigma\ \}$. This is the set of all the possible sequences of equivalence classes under $\approx$, that might be observed by watching the system whenever it starts in state σ. At this point, the information that might be learned by observing S through the view $\approx$ is obtained by transforming $\approx$, in function of the set $Obs_\sigma(S, \approx)$. In particular, the authors define a new equivalence relation $S[\approx]$, called *observational equivalence*, such that two states are equivalent only if the possible traces leading from these states are indistinguishable under $\approx$:

$$\forall \sigma, \sigma' \in \Sigma\ .\ \langle \sigma, \sigma' \rangle \in S[\approx] \ \Leftrightarrow\ Obs_\sigma(S, \approx) \equiv Obs_{\sigma'}(S, \approx)$$

Hence, in the paper a system is said secure if all the $\approx$-equivalent states are observationally equivalent. In other words, there is no information flow to an observer with view $\approx$. This characterization is then used in order to declassify data in the system. The basic idea of declassification is that any system that leaks information can be thought of as containing declassification. A passive attacker may be able to learn some information by observing the system but, by assumption, that information leakage is allowed by the security policy [118]. In this way, the attacker is made *blind*, i.e., all the the information that the attacker can get from the execution of the program is declassified. Note that, in [118], robust declassification is defined in the more general case where the attacker can be *active*, namely it can interfere in the execution, for example being a program running concurrently. This work has been recently generalized in [96] in three ways. First, it is shown how to express the property in a language-based setting, for a simple imperative language. Second, the property has been generalized so that untrusted code and data are explicitly part of the system rather than appearing only when there is an active attacker. Third, a security guarantee, called *qualified robustness* has been introduced. This provides untrusted code with a limited ability to affect information release. The key point of this paper is the proof that both robust and qualified declassification can be enforced by a compile-time program analysis based on a simple type system.

More recently, explicit declassification is allowed by weakening the notion of non-interference, in particular in [103] the notion of *delimited information release* is introduced in order to type as secure also systems that admit explicit confidential information release. The idea behind this notion is that a given program is secure as long as updates to variables that are later declassified occur in a way that does not increase the information visible by the attacker [103]. In order to solve the same problem, in [83] the authors define the notion of *relaxed noninterference* . The basic idea is to treat downgrading policies as security levels in traditional information flow systems. Instead of having only two security classes, i.e., `H` and `L` the authors consider a much richer lattice of security levels where each point corresponds to a downgrading policy, describing how the data can be downgraded

from this level. Afterwards, the authors define a type system for enforcing the new notion of non-interference.

5.5.2 Constraining attackers

As noted before, the notion of non-interference introduced in this chapter, is based on the assumption that an attacker is able to observe public data, without any observational or complexity restriction. In particular, for some computational systems, disclose any kind of confidential properties require a particular number of statistical tests [41], or a particular computational complexity [82]. The idea is to characterize, in some ways, which has to be the power of the attacker that can disclose certain confidential properties form a given program.

A probabilistic approach.

The notion of non-interference is based on the concept of *indistinguishability* of behaviours: In order to establish that there is no information flow between two objects A and B, it is sufficient to establish that, for any pair of behaviours of the system that differ only in A's object, B's observations cannot distinguish these two behaviours. This suggest that it is possible to weaken this notion by *approximating* this indistinguishability relation [41]. In this paper, the authors replace the notion of indistinguishability by the notion of *similarity*. Therefore, two behaviours, though distinguishable, might still be considered as *effectively* non-interfering, provided that they are similar, i.e., their difference is below a threshold ϵ. A similarity relation can be defined by means of an appropriate notion of distance and provides information on how much two behaviours differ from each other. The power of the attacker is then measured since this quantitative measure of differences between behaviours is related with the number of statistical tests needed to distinguish the two behaviours.

A complexity-based approach.

As noted above, the standard notion of non-interference requires that the public output of the program do not contain *any* information (in the information-theoretic sense) about the confidential inputs. This corresponds to an *all-powerful* attacker who, in his quest to obtain confidential information, has no bounds on the resources (time and space) that it can use. Furthermore, in these definitions an "attacker" is represented by an arbitrary function, which does not even have to be a computable function; the attacker is permitted essentially arbitrary power [82]. The observation made in this paper is that, instead, realistic adversaries are bounded in the resources that they can use. For this reason the author provides a definition of secure information flow that corresponds to an adversary working in probabilistic polynomial time, together with a program analysis that allows to certify these kinds of information flows.

6

Abstract Non-Interference: Imperative languages

There are more things in heaven and earth, Horatio,
than are dreamt of in our philosophy.
William Shakespeare

The standard approach to the confidentiality problem, i.e., *non-interference*, is based on a characterization of attackers that does not impose any observational or complexity restriction on the attackers' power. This means that, in this model, the attackers are *all-powerful*, namely they are modeled without any limitation in their quest to obtain confidential information. For this reason non-interference, as defined in literature, is an extremely restrictive policy. The problem of refining this kind of security policies has been addressed by many authors as a major challenge in language-based information flow security [104]. Refining security policies means weakening standard non-interference checks, in such a way that these restrictions can be used in practice. Namely, in order to adapt security policies to practical cases, we need a weaker notion of non-interference where the power of the attacker (or external viewer) is bounded, and where intentional leakage of information is allowed. Our idea is to use this weaker notion in order to characterize the secrecy degree of programs by identifying the most powerful attacker that is not able to disclose confidential information by observing the execution of programs, but also in order to characterize the most abstract information released. This would allow to certify the security of programs parametrically on the attackers' power. In order to systematically derive these certifications, it is essential to understand how much an attacker may learn from the executions of programs, since this information characterizes their security level. In this chapter, we show in which way we can model attackers as static program analyzers, whose aim is to disclose confidential information by (statically) analyzing the input/output behavior of programs. Therefore, our goal is to automatically generate, given a security policy, a certifi-

cate specifying that the given program has only secure information flows, relatively to a given attacker's model.

Consider the following program, written in the simple language Imp (see Sect. 4.2.1), where the **while**-statement iterates until x_1 is 0. Suppose x_1 is a secret variable and x_2 is a public variable:

$$\textbf{while } x_1 \textbf{ do } x_2 := x_2 + 2;\ x_1 := x_1 - 1 \textbf{ endw}$$

Clearly, in the standard sense of non-interference, there is an implicit flow from x_1 to x_2, since, due to the **while**-statement, x_2 changes depending on the initial value of x_1. This represents the case where no restriction is considered on the power of an attacker. However, suppose that the attacker can observe only the parity of values (0 is even). It is worth noting that if x_2 is initially even, then it is still even independently from the execution of the while, and therefore from the initial value of x_1. Similarly, if x_2 is initially odd then it remains odd independently from the execution of the while, i.e., from the value of x_1. This means that there's no information flow concerning parity.
We said above that, in the same model, we want also to characterize the private information that flows in programs, due to the semantics and to the attacker's observational capability. In order to understand how we can characterize *what* flows, consider the following program fragment:

$$l := l * h^2$$

Suppose that the attacker can only observe the parity of the public variable l, then it is clear that if we are interested only in keeping private the sign of h, then the program is secure, since the only information disclosed, in this case, is its parity. In this expression, it is the semantics of the program that puts a *firewall* that hides the sign of h. Therefore, given the model of the attacker, we can characterize, not only *if* there is an information flow, but also *what* is flowing, when it turns out that the program is insecure.

In order to model these situations we need to weaken standard non-interference relatively to the properties about data that an attacker can statically observe on program information flows. Therefore, we introduce the notion of *abstract non-interference* by parameterizing standard non-interference relatively to what an attacker is able to observe and to what has not to be revealed about confidential inputs. As we have said above, we consider attackers as *static program analyzers* whose task is to reveal properties of secret data by statically analyzing public resources. Hence, a program ensures secrecy with respect to a given property, which can be statically analyzed by the attacker, if that property on confidential data cannot be disclosed by analyzing public data. For instance, in the first example above, any attacker looking at parity is unable to disclose secrets about confidential data. In this sense the program is secret for parity, while it is not secret relatively to stronger attackers, able to observe more concrete properties of data such as

how much a variable grows (e.g. by interval analysis [28]). Since static program analysis can be fully specified as the abstract interpretation of the semantics of the program [28], we can model *attackers as abstract interpretations*. The results presented in this chapter has been published in [52].

6.1 Defining abstract non-interference

In this section, we define a notion of non-interference defined in terms of attackers modelled as abstract interpretations of concrete data domains. In other words, we can say that attackers are static program analyzers of data properties. Let $[\![P]\!]$ be the denotational semantics of a program P (see Sect. 4.1.2). In order to analyze the variables of a program as regards the given set of security classes $\{\mathtt{H}, \mathtt{L}\}$, we consider a typing function $t \in \mathit{Var} \longrightarrow \{\mathtt{H}, \mathtt{L}\}$, which associates with each variable in a program its security class. In the following, if $x \in \mathit{Var}(P)$ then we denote $x : t(x)$ the corresponding security typing. Moreover, whenever $\mathtt{T} \in \{\mathtt{H}, \mathtt{L}\}$, $v \in \mathbb{V}^n$, and $n = |\{x \in \mathit{Var}(P) | t(x) = \mathtt{T}\}|$, we abuse notation by denoting $v \in \mathbb{V}^{\mathtt{T}}$ the fact that v is a possible value for the vector of variables with security type $\mathtt{T}$. At this point, we can reformulate Eq. 5.2, defining standard non-interference, as follows:

$$\boxed{\begin{array}{c}\text{A program } P \text{ is } \textit{secure} \text{ if} \\ \forall v \in \mathbb{V}^{\mathtt{L}}, \forall v_1, v_2 \in \mathbb{V}^{\mathtt{H}} \,.\, [\![P]\!](v_1, v)^{\mathtt{L}} = [\![P]\!](v_2, v)^{\mathtt{L}}\end{array}}$$

Now, let us consider this definition applied to the example seen before:

$$P \stackrel{\text{def}}{=} \textbf{while } x_1 \textbf{ do } x_2 := x_2 + 2;\ x_1 := x_1 - 1 \textbf{ endw}$$

and note that, for instance, we have

$$[\![P]\!](1,2)^{\mathtt{L}} = 4 \neq [\![P]\!](0,2)^{\mathtt{L}} = 2$$

which means that the program is not secure. Suppose now that, as we noted above, the attacker is able to observe the parity of public output, then we would have

$$\mathit{Par}([\![P]\!](1,2)^{\mathtt{L}}) = \mathit{even} = \mathit{Par}([\![P]\!](0,2)^{\mathtt{L}})$$

The idea that arises looking at this example is that of modeling secrecy relatively to some fixed observable property. We assume that any program variable must preserve secrecy only as regards a particular amount of information, and depending on what the attacker can observe. In other words, the program is secret as long as a given property of private data is not disclosed from the given attacker. Hence, we introduce an abstract notion of information flow, which models an attacker that can observe only some properties of public (i.e., $\mathtt{L}$) concrete values. Clearly, in general, we can suppose that the attacker has the same kind of limitations in observing the public inputs. For this reason, in order to be as general as possible,

we distinguish between the attacker's observational capability on the public inputs and on the public outputs of programs, by considering two distinct properties for respectively input and output $\mathtt{L}$-values. As usual, in abstract interpretation, a property is an *upper closure operator* on the concrete domain of computations (see Sect. 2.2), therefore we consider two closure operators on the domain for public values, $\eta, \rho \in \mathit{uco}(\wp(\mathbb{V}^{\mathtt{L}}))$, for modeling the attacker's power. This leads us to the first generalization of standard non-interference, called *narrow (abstract) non-interference* (NANI for short). The idea is that a program satisfies narrow abstract non-interference relatively to a typing on security classes and a pair of closures η and ρ, denoted $\langle \eta, \rho \rangle$-*NSecrecy*, if, whenever the $\mathtt{L}$ input values have the same property η then the $\mathtt{L}$ output values have the same ρ property. This captures precisely the intuition that η-indistinguishable input values provide ρ-indistinguishable results. The following definition introduces the notion of narrow abstract non-interference as a generalization of the standard one.

Definition 6.1. *Let $\eta, \rho \in \mathit{uco}(\wp(\mathbb{V}^{\mathtt{L}}))$. A program $P \in$ Imp is $\langle \eta, \rho \rangle$-NSecret if*

$$\begin{array}{c} \forall h_1, h_2 \in \mathbb{V}^{\mathtt{H}}, \forall l_1, l_2 \in \mathbb{V}^{\mathtt{L}} \ . \\ \eta(l_1) = \eta(l_2) \ \Rightarrow \ \rho([\![P]\!](h_1, l_1)^{\mathtt{L}}) = \rho([\![P]\!](h_2, l_2)^{\mathtt{L}}). \end{array}$$

Clearly, if ρ cannot observe non-termination, i.e., $\{\bot\} \notin \rho$, then this notion could be termination-sensitive, namely, if the attacker is able to observe non-termination, then some information could be released even if narrow abstract non-interference is satisfied. We write $\models \ [\eta]P \ (\rho)$ (or simply $\ [\eta]P \ (\rho)$) to say that a program P is $\langle \eta, \rho \rangle$-*NSecret*. If $\ [\eta]P \ (\rho)$ does not hold, written $\not\models \ [\eta]P \ (\rho)$, then the attacker may observe an interference due to confidential data-flow.

Example 6.2. Consider the property *Sign* and *Par* represented in Figure 6.1 and the program:

$$P \stackrel{\text{def}}{=} \ l := 2 * l * h^2;$$

with security typing: $t = \langle h : \mathtt{H}, l : \mathtt{L} \rangle$ and $\mathbb{V} = \mathbb{Z}$. Clearly in the standard notion of

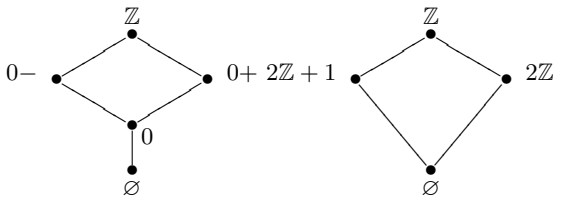


Fig. 6.1. The *Sign* and *Par* domains.

secrecy there is a flow of information from variable h to variable l, since l depends on the value of h, i.e., the statement is not secure. Let's see what happens for the $\langle \mathit{Sign}, \mathit{Par} \rangle$-*NSecrecy*. If the input is such that $\mathit{Sign}(l) = 0+$, then the possible

outputs, depending on h, are always in $2\mathbb{Z}$, indeed the result is always even because there is a multiplication by 2. The same holds if $\mathit{Sign}(l) = 0-$. Therefore any possible output value, with a fixed input l, has the same abstraction in Par, which is $2\mathbb{Z}$. Hence we have $\models\ [\mathit{Sign}]P\ (\mathit{Par})$.

It is worth noting that $\not\models\ [\eta]P\ (\rho)$ does not necessarily imply an information flow from `H` to `L` values. In fact, whenever narrow non-interference fails, it is possible that the revealed flow is a flow due to the η-undistinguished public values. Namely, what is revealed may not be an insecure information flow, which means that this flow may not convey private information into the public output. Indeed, whenever we can find two different public values l_1 and l_2 such that $\eta(l_1) = \eta(l_2)$, and a private value h such that $\rho([\![P]\!](h, l_1)^{\mathtt{L}}) \neq \rho([\![P]\!](h, l_2)^{\mathtt{L}})$, then narrow non-interference fails for what we call a *deceptive flow*. Since these flows are due to the fact that the public input ranges over sets of elements with the same property η. It is worth noting that the smaller are these sets, i.e., the more precise is η, and the less deceptive flows may arise.

Example 6.3. Consider the property Sign and Par represented in Figure 6.1 and the program in Example 6.2. Let us consider $\langle \mathit{Par}, \mathit{Sign}\rangle$-*NSecrecy*. In this case note that

$$\mathit{Par}(-2) = \mathit{Par}(4) = 2\mathbb{Z} \text{ but } \mathit{Sign}([\![P]\!](h, -2)^{\mathtt{L}}) = 0- \neq 0+ = \mathit{Sign}([\![P]\!](h, 4)^{\mathtt{L}})$$

In general we have a flow for each positive and negative number with the same abstraction in Par, i.e., for each pair of even or odd numbers with different sign. This means that $[\mathit{Par}]P\ (\mathit{Sign})$ doesn't hold due to deceptive flows generated by variations of low inputs having the same property in Par.

In order to avoid the presence of deceptive flows we consider the possibility of passing abstract `L` values as input to the program's semantics. The idea is to model only information flows generated by the variation of `H` values. This is obtained by computing the concrete semantics applied to abstract values for `L` inputs, denoting η-indistinguishable data.

Moreover, we said above, that we want to characterize also which properties of the private input can flow, or conversely which property of private input has to be kept confidential. Hence, we model when the change of a particular property in the private input has effects in what the attacker can observe concerning the properties of public output. This leads us to introduce a weaker notion of non-interference having no deceptive flows and such that, when the attacker is able to observe the property η of public input and the property ρ of public output, no information flow concerning the property ϕ of private input interferes in the observable property ρ of the public output, under the assumption that the public input property η doesn't change. In this case, the abstraction ϕ represents the property of private data that has not to flow into the public variables, given an attacker that can

observe η on public input and ρ on public output. We call this notion *abstract non-interference* (ANI for short), denoted $\langle \eta, \phi, \rho \rangle$-*Secrecy*, as regards the closures $\eta, \rho \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$, and $\phi \in uco(\wp(\mathbb{V}^{\mathtt{H}}))$.

Definition 6.4. *Let* $\eta, \rho \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$ *and* $\phi \in uco(\wp(\mathbb{V}^{\mathtt{H}}))$. $P \in$ Imp *is* $\langle \eta, \phi, \rho \rangle$-*Secret if*

$$\forall h_1, h_2 \in \mathbb{V}^{\mathtt{H}}, \forall l_1, l_2 \in \mathbb{V}^{\mathtt{L}} \ .$$
$$\eta(l_1) = \eta(l_2) \ \Rightarrow \ \rho([\![P]\!](\phi(h_1), \eta(l_1))^{\mathtt{L}}) = \rho([\![P]\!](\phi(h_2), \eta(l_2))^{\mathtt{L}}).$$

In the following, when a program P is $\langle \eta, \phi, \rho \rangle$-*Secret*, we will write $\models (\eta)P \ (\phi \rightsquigarrow\!\!\!\!| \rho)$ (or simply $(\eta)P \ (\phi \rightsquigarrow\!\!\!\!| \rho)$). Next example shows the difference between narrow and abstract non-interference.

Example 6.5. Consider the property *Par* represented in Figure 6.1, the sign property defined as $Sign \stackrel{\text{def}}{=} \{\top, +, -, \varnothing\}$ and the program in Example 6.2. Consider $\langle Par, id, Sign \rangle$-*Secrecy*. Then we have that

$$Sign([\![P]\!](h, Par(-2))^{\mathtt{L}}) = Sign([\![P]\!](h, Par(4))^{\mathtt{L}}) = Sign(2\mathbb{Z} * h^2) = \mathbb{Z}.$$

In general we can prove that $(Par)P \ (id \rightsquigarrow\!\!\!\!| Sign)$ holds. Hence no more deceptive flows can be revealed.

In order to understand which is the difference between $(\eta)P \ (id \rightsquigarrow\!\!\!\!| \rho)$ and $(\eta)P \ (\phi \rightsquigarrow\!\!\!\!| \rho)$, let us consider the example shown before.

Example 6.6. Consider the properties *Sign* and *Par* described in Fig. 6.1 and the program fragment used before:

$$P \ \stackrel{\text{def}}{=} \ l := l * h^2;$$

with security typing: $t = \langle h : \mathtt{H}, l : \mathtt{L} \rangle$ and $\mathbb{V} = \mathbb{Z}$. Consider $\langle id, id, Par \rangle$-*Secrecy*. Note that:

$$Par([\![P]\!](2, 1)^{\mathtt{L}}) = Par(4) = 2\mathbb{Z} \text{ while } Par([\![P]\!](3, 1)^{\mathtt{L}}) = Par(9) = 2\mathbb{Z} + 1$$

which are clearly different, therefore in this case $(id)P \ (id \rightsquigarrow\!\!\!\!| Par)$ doesn't hold. On the other hand, consider $\langle id, Sign, Par \rangle$-*Secrecy* and note that:

$$Par([\![P]\!](Sign(2), 1)^{\mathtt{L}}) = Par([\![P]\!](Sign(3), 1)^{\mathtt{L}}) = Par(0+) = \mathbb{Z}.$$

In this case, it is simple to check that $(id)P \ (Sign \rightsquigarrow\!\!\!\!| Par)$ holds.

It is clear that standard non-interference is exactly abstract non-interference with all identity maps, i.e., $\langle id, id \rangle$-*NSecrecy* and $\langle id, id, id \rangle$-*Secrecy*. More in general, the following proposition shows the relations existing among the given notions of non-interference.

Proposition 6.7.
$$\begin{array}{c} [id]P\ (id) \\ \Downarrow \\ [\eta]P\ (\rho) \Rightarrow\ (\eta)P\ (id \rightsquigarrow \rho) \Rightarrow\ (\eta)P\ (\phi \rightsquigarrow \rho) \end{array}$$

Proof. We prove the single implications separately. Consider $l, l_1, l_2 \in \mathbb{V}^{\mathtt{L}}$ and $h_1, h_2 \in \mathtt{H}$. The following proofs use the well-known property $\rho(\bigcup Y) = \rho(\bigcup \rho(Y))$ (see Prop. 2.57).

$\boxed{[id]P\ (id)\ \Rightarrow\ (\eta)P\ (id \rightsquigarrow \rho)}$ Suppose $[\![P]\!](h_1, l)^{\mathtt{L}} = [\![P]\!](h_2, l)^{\mathtt{L}}$ and consider $\eta(l_1) = \eta(l_2)$, we have to prove that $\rho([\![P]\!](h_1, \eta(l_1))^{\mathtt{L}}) = \rho([\![P]\!](h_2, \eta(l_2)^{\mathtt{L}})$:

$$\begin{array}{lll} \rho([\![P]\!](h_1, \eta(l_1))^{\mathtt{L}}) & = \rho\left(\bigcup_{l \in \eta(l_1)} [\![P]\!](h_1, l)^{\mathtt{L}}\right) & \\ & = \rho\left(\bigcup_{l \in \eta(l_2)} [\![P]\!](h_1, l)^{\mathtt{L}}\right) & (\text{since } \eta(l_1) = \eta(l_2)) \\ & = \rho\left(\bigcup_{l \in \eta(l_2)} [\![P]\!](h_2, l)^{\mathtt{L}}\right) & (\text{by hypothesis } [id]P\ (id)) \\ & = \rho([\![P]\!](h_2, \eta(l_2))^{\mathtt{L}}) & \end{array}$$

$\boxed{[\eta]P\ (\rho)\ \Rightarrow\ (\eta)P\ (id \rightsquigarrow \rho)}$ Suppose $\eta(l_1) = \eta(l_2)$ implies $\rho([\![P]\!](h_1, l_1)^{\mathtt{L}}) = \rho([\![P]\!](h_2, l_2)^{\mathtt{L}})$, we have to prove that $\rho([\![P]\!](h_1, \eta(l_1))^{\mathtt{L}}) = \rho([\![P]\!](h_2, \eta(l_2))^{\mathtt{L}})$ holds:

$$\begin{array}{lll} \rho([\![P]\!](h_1, \eta(l_1))^{\mathtt{L}}) & = \rho\left(\bigcup_{l \in \eta(l_1)} [\![P]\!](h_1, l)^{\mathtt{L}}\right) & \\ & = \rho\left(\bigcup_{l \in \eta(l_1)} \rho([\![P]\!](h_1, l)^{\mathtt{L}})\right) & (\text{by Prop. 2.57}) \\ & = \rho\left(\bigcup_{l \in \eta(l_2)} \rho([\![P]\!](h_2, l)^{\mathtt{L}})\right) & (\text{by hypothesis } [\eta]P\ (\rho)) \\ & = \rho\left(\bigcup_{l \in \eta(l_2)} [\![P]\!](h_2, l)^{\mathtt{L}}\right) & (\text{by Prop. 2.57}) \\ & = \rho([\![P]\!](h_2, \eta(l_2))^{\mathtt{L}}) & \end{array}$$

$\boxed{(\eta)P\ (id \rightsquigarrow \rho)\ \Rightarrow\ (\eta)P\ (\phi \rightsquigarrow \rho)}$ Suppose $(\eta)P\ (id \rightsquigarrow \rho)$, we prove that $\eta(l_1) = \eta(l_2) = \eta(l)$ implies $\rho([\![P]\!](\phi(h_1), \eta(l))^{\mathtt{L}}) = \rho([\![P]\!](\phi(h_2), \eta(l))^{\mathtt{L}})$:

$$\begin{array}{lll} \rho([\![P]\!](\phi(h_1), \eta(l))^{\mathtt{L}}) & = \rho\left(\bigcup_{h \in \phi(h_1)} [\![P]\!](h_1, \eta(l))^{\mathtt{L}}\right) & \\ & = \rho\left(\bigcup_{h \in \phi(h_1)} \rho([\![P]\!](h, \eta(l))^{\mathtt{L}})\right) & (\text{by Prop. 2.57}) \\ & = \rho\left(\bigcup_{h' \in \phi(h_2)} \rho([\![P]\!](h', \eta(l))^{\mathtt{L}})\right) & (\text{by } (\eta)P\ (id \rightsquigarrow \rho)) \\ & = \rho\left(\bigcup_{h' \in \phi(h_2)} [\![P]\!](h', \eta(l))^{\mathtt{L}}\right) & (\text{by Prop. 2.57}) \\ & = \rho([\![P]\!](\phi(h_2), \eta(l))^{\mathtt{L}}) & \end{array}$$

Note that, we don't have the implication $[id]P\ (id)\ \Rightarrow\ [\eta]P\ (\rho)$. The following example shows that this implication does not hold, due to deceptive flows.

Example 6.8. Consider $\eta = Sign$ and $\rho = Par$, defined in Fig. 6.1. Consider the program fragment:
$$P \stackrel{\text{def}}{=} l := l + 2$$

We can note that $\models\ [id]l := l + 2\ (id)$ since there cannot be insecure flows in P. On the other hand, we can note that $\not\models\ [Sign]l := l + 2\ (Par)$ since, for example, we have the following deceptive flow:

$$Par([\![l := l + 2]\!](h, 2)^{\mathtt{L}}) = Par(4) = 2\mathbb{Z} \qquad \text{while}$$
$$Par([\![l := l + 2]\!](h, 3)^{\mathtt{L}}) = Par(5) = 2\mathbb{Z} + 1$$

where $Sign(2) = Sign(3) = +$.

Abstract non-interference is parametric on program properties specified as closure operators. In order to better understand the meaning of input/output abstractions in the definitions above, we observe that the property $(\eta)P\ (\phi \rightsquigarrow \phi(\{\top\}))$ always holds. Indeed, if a closure makes equal some objects, then any more abstract closure will make equal at least the same objects. From these simple observations, we derive the following basic properties of narrow and abstract non-interference.

Proposition 6.9. *Let* $\eta, \rho \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$, $\phi \in uco(\wp(\mathbb{V}^{\mathtt{H}}))$, *and* $P \in$ Imp.

1. $[\eta]P\ (\rho) \Leftrightarrow \forall \beta \sqsupseteq \rho\,.\ [\eta]P\ (\beta)$;
2. $[\eta]P\ (\rho) \Leftrightarrow \forall \beta \sqsubseteq \eta\,.\ [\beta]P\ (\rho)$;
3. $\forall i.\ [\eta]P\ (\rho_i) \Rightarrow [\eta]P\ (\bigsqcap_i \rho_i)$ *and* $[\eta]P\ (\bigsqcup_i \rho_i)$;
4. $[id]P\ (\rho) \Leftrightarrow (id)P\ (id \rightsquigarrow \rho)$;
5. $(\eta)P\ (\phi \rightsquigarrow \rho) \Rightarrow \forall \beta \sqsupseteq \rho\,.\ (\eta)P\ (\phi \rightsquigarrow \beta)$;
6. $\forall i.\ (\eta)P\ (\phi \rightsquigarrow \rho_i) \Rightarrow (\eta)P\ (\phi \rightsquigarrow \bigsqcap_i \rho_i)$ *and* $(\eta)P\ (\phi \rightsquigarrow \bigsqcup_i \rho_i)$.

Proof. 1. If $\forall \beta \sqsupseteq \rho\,.\ [\eta]P\ (\beta)$, then $[\eta]P\ (\rho)$. Let us consider the other inclusion. Suppose $[\eta]P\ (\rho)$, namely $\forall l_1, l_2 \in \mathbb{V}^{\mathtt{L}}$ and $\forall h_1, h_2 \in \mathbb{V}^{\mathtt{H}}$, we have that $\eta(l_1) = \eta(l_2)$ implies the equality $\rho([\![P]\!](h_1, l_1)^{\mathtt{L}}) = \rho([\![P]\!](h_2, l_2)^{\mathtt{L}})$. Consider $\beta \sqsupseteq \rho$, then $\forall X, Y \in \wp(\mathbb{V}^{\mathtt{L}})\,.\ \rho(X) = \rho(Y) \Rightarrow \beta(X) = \beta(Y)$. This means that $\rho([\![P]\!](h_1, l_1)^{\mathtt{L}}) = \rho([\![P]\!](h_2, l_2)^{\mathtt{L}})$ implies $\beta([\![P]\!](h_1, l_1)^{\mathtt{L}}) = \beta([\![P]\!](h_2, l_2)^{\mathtt{L}})$ and therefore $[\eta]P\ (\beta)$.

2. If $\forall \beta \sqsubseteq \eta\,.\ [\beta]P\ (\rho)$, then $[\eta]P\ (\rho)$. Consider the other implication. Suppose $[\eta]P\ (\rho)$, and consider $\beta \sqsubseteq \eta$, then $\forall x, y \in \mathbb{V}^{\mathtt{L}}$ we have that $\beta(x) = \beta(y) \Rightarrow \eta(x) = \eta(y)$. By hypothesis $\forall l_1, l_2 \in \mathbb{V}^{\mathtt{L}}, \forall h_1, h_2 \in \mathbb{V}^{\mathtt{H}}$ we have that $\eta(l_1) = \eta(l_2)$ implies $\rho([\![P]\!](h_1, l_1)^{\mathtt{L}}) = \rho([\![P]\!](h_2, l_2)^{\mathtt{L}})$. Therefore $\beta(l_1) = \beta(l_2)$, which implies $\eta(l_1) = \eta(l_2)$, implies also $\rho([\![P]\!](h_1, l_1)^{\mathtt{L}}) = \rho([\![P]\!](h_2, l_2)^{\mathtt{L}})$. Hence we have that $[\beta]P\ (\rho)$ holds.

3. By hypotheses, for each $i \in I$, $\eta(l_1) = \eta(l_2)$ implies that $\rho_i([\![P]\!](h_1, l_1)^{\mathtt{L}}) = \rho_i([\![P]\!](h_2, l_2)^{\mathtt{L}})$, namely $\eta(l_1) = \eta(l_2)$ implies $\forall i \in I.\ \rho_i([\![P]\!](h_1, l_1)^{\mathtt{L}}) = \rho_i([\![P]\!](h_2, l_2)^{\mathtt{L}})$. But this corresponds to saying that $\eta(l_1) = \eta(l_2)$ implies that $\bigsqcap_i \rho_i([\![P]\!](h_1, l_1)^{\mathtt{L}}) = \bigsqcap_i \rho_i([\![P]\!](h_2, l_2)^{\mathtt{L}})$. $[\eta]P\ (\bigsqcup_i \rho_i)$ holds for the first point.

4. Straightforward by the definitions of narrow and abstract non-interference.

5. If $\forall \beta \sqsupseteq \rho\,.\ (\eta)P\ (\phi \rightsquigarrow \beta)$, then $(\eta)P\ (\phi \rightsquigarrow \rho)$. Let us consider the other inclusion. Suppose $(\eta)P\ (\phi \rightsquigarrow \rho)$, namely consider $\forall l_1, l_2 \in \mathbb{V}^{\mathtt{L}}, \forall h_1, h_2 \in \mathbb{V}^{\mathtt{H}}$ we have that $\eta(l_1) = \eta(l_2)$ implies $\rho([\![P]\!](\phi(h_1), \eta(l_1))^{\mathtt{L}}) = \rho([\![P]\!](\phi(h_2), \eta(l_2))^{\mathtt{L}})$.

Consider now $\beta \sqsupseteq \rho$, then $\forall X, Y \in \wp(\mathbb{V}^{\mathtt{L}}) \,.\, \rho(X) = \rho(Y) \;\Rightarrow\; \beta(X) = \beta(Y)$. This means that $\rho([\![P]\!](\phi(h_1), \eta(l_1))^{\mathtt{L}}) = \rho([\![P]\!](\phi(h_2), \eta(l_2))^{\mathtt{L}})$ implies $\beta([\![P]\!](\phi(h_1), \eta(l_1))^{\mathtt{L}}) = \beta([\![P]\!](\phi(h_2), \eta(l_2))^{\mathtt{L}})$ and therefore $(\eta)P\ (\phi \rightsquigarrow \!\!\!\downharpoonright \beta)$.

6. By hypotheses, for each $i \in I$ the condition $\eta(l_1) = \eta(l_2)$ implies that $\rho_i([\![P]\!](\phi(h_1), \eta(l_1))^{\mathtt{L}}) = \rho_i([\![P]\!](\phi(h_2), \eta(l_2))^{\mathtt{L}})$. This is equivalent to saying that $\eta(l_1) = \eta(l_2)$ implies $\rho_i([\![P]\!](\phi(h_1), \eta(l_1))^{\mathtt{L}}) = \rho_i([\![P]\!](\phi(h_2), \eta(l_2))^{\mathtt{L}})$, for each $i \in I$. This corresponds to saying that $\eta(l_1) = \eta(l_2)$ implies $\bigsqcap_i \rho_i([\![P]\!](\phi(h_1), \eta(l_1))^{\mathtt{L}}) = \bigsqcap_i \rho_i([\![P]\!](\phi(h_2), \eta(l_2))^{\mathtt{L}})$. Hence, $(\eta)P\ (\phi \rightsquigarrow \!\!\!\downharpoonright \bigsqcup_i \rho_i)$ holds for we proved in the first point.

6.2 Checking abstract non-interference

In this section, we derive the abstract non-interference semantics of a programming language by abstract interpretation of its maximal trace semantics. The abstract non-interference semantics of a program is the set of all the denotations, viz. functions, representing computations for P that satisfy $(\eta)P\ (\phi \rightsquigarrow \!\!\!\downharpoonright \rho)$. Formally, we define the domain $\mathbb{S}\ (\eta, \phi \rightsquigarrow \!\!\!\downharpoonright \rho)$ parametric on the abstract domains $\eta, \rho \in \mathit{uco}(\wp(\mathbb{V}^{\mathtt{L}}))$ and $\phi \in \mathit{uco}(\wp(\mathbb{V}^{\mathtt{H}}))$. This domain is an abstraction of the concrete domain of the angelic denotational semantics $\Sigma \to \wp(\Sigma)$ as introduced in [27], where $\Sigma = \mathbb{V}^n$ (see Sect. 4.1.2).

$$\mathbb{S}\ (\eta, \phi \rightsquigarrow \!\!\!\downharpoonright \rho) \stackrel{\text{def}}{=} \left\{ f \in \Sigma \to \wp(\Sigma) \;\middle|\; \begin{array}{l} \forall h_1, h_2 \in \mathbb{V}^{\mathtt{H}}, \forall l_1, l_2 \in \mathbb{V}^{\mathtt{L}} \,.\, \eta(l_1) = \eta(l_2) \;\Rightarrow \\ \rho(f(\phi(h_1), \eta(l_1))^{\mathtt{L}}) = \rho(f(\phi(h_2), \eta(l_2))^{\mathtt{L}}) \end{array} \right\}$$

The key point, in order to show that $\mathbb{S}\ (\eta, \phi \rightsquigarrow \!\!\!\downharpoonright \rho)$ is an abstraction of $\alpha^{\mathcal{D}}(\wp(\Sigma^\infty))$, is to note that if f doesn't satisfy *abstract non-interference*, i.e., $f \notin \mathbb{S}\ (\eta, \phi \rightsquigarrow \!\!\!\downharpoonright \rho)$, then there is no way to make it satisfy that property by adding traces, namely $\forall g \sqsupseteq f.\ g \notin \mathbb{S}\ (\eta, \phi \rightsquigarrow \!\!\!\downharpoonright \rho)$. On the other hand, if $f \in \mathbb{S}\ (\eta, \phi \rightsquigarrow \!\!\!\downharpoonright \rho)$, then $\forall g \sqsubseteq f$ we have $g \in \mathbb{S}\ (\eta, \phi \rightsquigarrow \!\!\!\downharpoonright \rho)$.

Proposition 6.10. *Let $\eta, \rho \in \mathit{uco}(\wp(\mathbb{V}^{\mathtt{L}}))$ and $\phi \in \mathit{uco}(\wp(\mathbb{V}^{\mathtt{H}}))$.*

1. *$\mathbb{S}\ (\eta, \phi \rightsquigarrow \!\!\!\downharpoonright \rho)$ is a Moore-family;*
2. *$(\mathbb{S}\ (\eta, \phi \rightsquigarrow \!\!\!\downharpoonright \rho), \alpha, \mathit{id}, \alpha^{\mathcal{D}}(\wp(\Sigma^\infty)))$ is a GI where, if $f \in \alpha^{\mathcal{D}}(\wp(\Sigma^\infty))$, then*

$$\alpha(f) = \begin{cases} \lambda x.\ \Sigma & \text{if } \exists \delta_{\vdash}, \sigma_{\vdash} \in \Sigma \,.\, \eta(\sigma^{\mathtt{L}}_{\vdash}) = \eta(\delta^{\mathtt{L}}_{\vdash}) \wedge \\ & \quad \rho(f(\phi(\sigma^{\mathtt{H}}_{\vdash}), \eta(\sigma^{\mathtt{L}}_{\vdash}))^{\mathtt{L}}) \neq \rho(f(\phi(\sigma^{\mathtt{H}}_{\vdash}), \eta(\delta^{\mathtt{L}}_{\vdash}))^{\mathtt{L}}) \\ f & \text{otherwise} \end{cases}$$

Proof. The proof that $\mathbb{S}\ (\eta, \phi \rightsquigarrow \!\!\!\downharpoonright \rho)$ is a Moore-family comes directly from the fact that the greatest lower bound of functions corresponds to the intersection of the target set the functions. This means that the greatest lower bound operation selects only a subset of the set of computations of both the functions. Therefore, since it is immediate to verify that by deleting computations we cannot add interference, the intersection function denote secret computations.

It is immediate to prove, by definition, that $\alpha^{\mathcal{D}}$ is monotone, extensive and idempotent.

An analogous result holds in the narrow case. Hence, to check $(\eta)P\ (\phi \rightsquigarrow\!\!\!\!| \rho)$ means checking whether $\alpha(\llbracket P \rrbracket) = \llbracket P \rrbracket$. This check boils down to a standard static program analysis. Let $P \in \textsc{Imp}$, $\eta, \rho \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$ and $\phi \in uco(\wp(\mathbb{V}^{\mathtt{H}}))$ be the observable properties that characterize the attacker. We observe that it is possible to monitor the possible flows of information from variables of type `H` to variables of type `L`, by considering the best correct approximation of $\llbracket P \rrbracket$ given by ρ, η, and ϕ.

Theorem 6.11. *Let $\eta, \rho \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$ and $\phi \in uco(\wp(\mathbb{V}^{\mathtt{H}}))$.*

- $[\eta]P\ (\rho)$ *iff* $\forall Y \in \eta(\wp(\mathbb{V}^{\mathtt{L}}))\,.\, \lambda x \in \mathbb{V}^{\mathtt{H}}, y \in Y.\, \rho(\llbracket P \rrbracket(x,y)^{\mathtt{L}})$ *is constant.*
- $(\eta)P\ (\phi \rightsquigarrow\!\!\!\!| \rho)$ *iff* $\forall Y \in \eta(\wp(\mathbb{V}^{\mathtt{L}}))\,.\, \lambda x \in \mathbb{V}^{\mathtt{H}}\,.\, \rho(\llbracket P \rrbracket(\phi(x), Y)^{\mathtt{L}})$ *is constant.*

Proof. Let us prove the two directions separately. Consider ($\Leftarrow$). Suppose that the function $F = \lambda x \in \mathbb{V}^{\mathtt{H}}\,.\, \rho(\llbracket P \rrbracket(\phi(x), Y)^{\mathtt{L}})$ is a constant map, which means that $\forall \overline{x}_1, \overline{x}_2 \in \mathbb{V}^h\,.\, F(\overline{x}_1) = F(\overline{x}_2)$. This means that $\forall \overline{x}_1, \overline{x}_2 \in \mathbb{V}^h\,.\, \rho\llbracket P \rrbracket(\phi(\overline{x}_1), \eta(\overline{y}))^{\mathtt{L}} = \rho\llbracket P \rrbracket(\phi(\overline{x}_2), \eta(\overline{y}))^{\mathtt{L}}$, and this holds for each $\overline{y} \in \mathbb{V}^l$, therefore we have secrecy. Consider ($\Rightarrow$). Suppose that $(\eta)P\ (\phi \rightsquigarrow\!\!\!\!| \rho)$, namely $\forall \overline{y} \in \mathbb{V}^l, \overline{x}_1, \overline{x}_2 \in \mathbb{V}^h$ we have the equality $\rho(\llbracket P \rrbracket(\phi(\overline{x}_1), \eta(\overline{y}))^{\mathtt{L}}) = \rho(\llbracket P \rrbracket(\phi(\overline{x}_2), \eta(\overline{y}))^{\mathtt{L}})$. But this holds for each $\overline{x}_1, \overline{x}_2$, therefore the function is constant. A similar proof can be done for the narrow case.

Hence, checking abstract non-interference corresponds to checking whether the best correct approximation of the concrete semantics of P, restricted to `H` variables in input and `L` variables in output, is constant. A similar result holds for narrow abstract non-interference, even though the checking process in the narrow case would not involve the best correct approximation of the concrete semantics but rather the concrete semantics itself. It is clear that, if ϕ and η are complete abstractions for ρ and $\llbracket P \rrbracket$ (see Sect. 2.2.4 [65]), then $[\eta]P\ (\rho)$ iff $(\eta)P\ (\phi \rightsquigarrow\!\!\!\!| \rho)$. Finally, note that abstract non-interference, as well as non-interference, is not in general a safety property [91] since it is a property of sets of traces and not a property of traces. It is a safety property only if no possible change is observable on `L` variables.

6.3 Deriving attackers

In this section, we define systematic methods for deriving attackers from programs by abstract interpretation. In particular, we are interested in characterizing the most concrete, viz. most precise, attacker for which a given program is secure. This is useful both in automatic program certification for abstract non-interference and in order to classify programs in terms of the properties that make them secure.

Since attackers are characterized as pairs of abstract domains, the idea is to define an abstract domain transformer, depending on the program to be analyzed, which is able to transform any non-secret abstraction ρ into the closest abstraction for which the program is secure. This corresponds to characterizing the most powerful *harmless* attacker for a given program. Here, harmless means that we look for the most powerful attacker whose observational capability is not sufficient for disclosing any confidential information.

This construction is significant in both, narrow and abstract non-interference, when we are intended to simplify domains: By Proposition 6.9, any refinement of non-secret output abstraction is still non-secret. Let P be a program, $\eta, \rho \in \mathit{uco}(\wp(\mathbb{V}^{\mathtt{L}}))$, and $\phi \in \mathit{uco}(\wp(\mathbb{V}^{\mathtt{H}}))$. Assume that the program P is not $\langle \eta, \rho \rangle$-*NSecret* [resp. $\langle \eta, \phi, \rho \rangle$-*Secret*]. We assume fixed the input-abstraction η and the private property ϕ. We know by Proposition 6.9 that the most concrete $\beta \sqsupseteq \rho$ such that $[\eta]P\ (\beta)$ [resp. $(\eta)P\ (\phi \rightsquigarrow\!\!\!\!\downarrow \beta)$], always exists unique. We call this domain the *narrow* [resp. *abstract*] *secret kernel of* ρ *for* P. As usual in systematic abstract domain design [31,61], we specify secret kernels by corresponding abstract domain transformers, $\mathcal{K}_{P,[\eta]}, \mathcal{K}_{P,(\eta),\phi} : \mathit{uco}(\wp(\mathbb{V}^{\mathtt{L}})) \longrightarrow \mathit{uco}(\wp(\mathbb{V}^{\mathtt{L}}))$:

$$\begin{array}{rcl} \mathcal{K}_{P,[\eta]} & \stackrel{\text{def}}{=} & \lambda\rho.\ \bigsqcap \left\{\ \beta \,\middle|\, \rho \sqsubseteq \beta \ \wedge\ [\eta]P\ (\beta)\ \right\} \\ \mathcal{K}_{P,(\eta),\phi} & \stackrel{\text{def}}{=} & \lambda\rho.\ \bigsqcap \left\{\ \beta \,\middle|\, \rho \sqsubseteq \beta \ \wedge\ (\eta)P\ (\phi \rightsquigarrow\!\!\!\!\downarrow \beta)\ \right\} \end{array}$$

In order to characterize these abstract domain transformers, we have to identify when a program property, viewed as a collection of values, is secure. Program properties are closure operators on sets of possible values. This means that we have to characterize the sets of values that can stay in an abstract domain ρ in order to guarantee $[\eta]P\ (\rho)$ [resp. $(\eta)P\ (\phi \rightsquigarrow\!\!\!\!\downarrow \rho)$].

6.3.1 Characterizing secret kernels

The idea is that of defining a predicate on sets of values that identifies exactly those elements that, in a given closure ρ, form the secret kernel, namely we would like to define a *core* of the kind $\mathcal{R}^{-}_{\pi}$ (see Sect. 3.1), where π is composed by all the closures whose elements guarantee secrecy. Let's try to understand which properties have to satisfy the elements of a closure in order to guarantee abstract non-interference. Consider the definition of narrow non-interference, it requires that all the objects $[\![P]\!](h, l)$, where the l's have the same property η and h is any possible private value, have the same property ρ. This means that, if we want to build the most concrete such ρ, we have to collect together, in the same abstract object, all such elements. For this reason, given a public value $l \in \mathbb{V}^{\mathtt{L}}$, we define the following set characterizing all the elements that have to be *indistinguishable* in order to guarantee NANI:

$$\Upsilon^{\eta}_{[\![P]\!]}(l) \stackrel{\text{def}}{=} \left\{\ [\![P]\!](h, y)^{\mathtt{L}} \,\middle|\, \eta(y) = \eta(l),\ h \in \mathbb{V}^{\mathtt{H}}\ \right\},$$

On the other hand, consider abstract non-interference, its definition requires that all the elements $[\![P]\!](\phi(h), \eta(l))$, where l is a given public value, while h can range over the domain for private values, have the same abstraction under ρ. As above, this means that, if we want to build the most concrete such abstract domain ρ, we have to collect together, in the same abstract object, all these elements. For this reason, given a public value $l \in \mathbb{V}^{\mathtt{L}}$, we define the following set characterizing all the elements that have to be *indistinguishable* in order to guarantee ANI:

$$\Upsilon^{\eta,\,\phi}_{[\![P]\!]}(l) \stackrel{\text{def}}{=} \left\{\ [\![P]\!](\phi(h), \eta(l))^{\mathtt{L}} \ \middle|\ h \in \mathbb{V}^{\mathtt{H}}\ \right\}$$

In other words, these sets represent the collections of the sets of values that any secure property ρ has not to distinguish, i.e., which have to be abstracted by ρ into the same object. At this point, if we want to reformulate the notion of abstract non-interference in terms of the sets defined above, we can say that any abstract domain ρ, that guarantee non-interference, has not to distinguish elements in each one of the sets $\Upsilon^{\eta}_{[\![P]\!]}$ [resp. $\Upsilon^{\eta,\,\phi}_{[\![P]\!]}$]. We formalize this intuition by defining a predicate *Secr*, such that $\mathit{Secr}(X)$, for X set of values, holds whenever X is not able to distinguish values in each one of the sets $\Upsilon^{\eta}_{[\![P]\!]}$ [resp. $\Upsilon^{\eta,\,\phi}_{[\![P]\!]}$]. So, let $\varepsilon = \eta$ or $\varepsilon = \eta, \phi$. We define the predicate $\mathit{Secr}^{\varepsilon}_{[\![P]\!]}(X)$ for any $X \subseteq \mathbb{V}^{\mathtt{L}}$:

$$\begin{array}{c} \mathit{Secr}^{\varepsilon}_{[\![P]\!]}(X) \text{ iff} \\ \forall l \in \mathbb{V}^{\mathtt{L}}\ .\ (\exists Z \in \Upsilon^{\varepsilon}_{[\![P]\!]}(l)\ .\ Z \subseteq X\ \Rightarrow\ \forall W \in \Upsilon^{\varepsilon}_{[\![P]\!]}(l)\ .\ W \subseteq X) \end{array}$$

In other words, $\mathit{Secr}^{\varepsilon}_{[\![P]\!]}(X)$ holds if X does not brake any $\Upsilon^{\varepsilon}_{[\![P]\!]}(l)$, namely for each $l \in \mathbb{V}^{\mathtt{L}}$, X contains all the sets in $\Upsilon^{\varepsilon}_{[\![P]\!]}(l)$ or none of them. In this case we also say that X is *secret*.

The following result proves that *Secr* precisely identifies all and only those sets of values that form the secret kernel.

Theorem 6.12. *Let $\eta \in \mathit{uco}(\wp(\mathbb{V}^{\mathtt{L}}))$ and $\phi \in \mathit{uco}(\wp(\mathbb{V}^{\mathtt{H}}))$.*

- $\mathcal{K}_{P,\,[\eta]}(id) = \left\{\ X \in \wp(\mathbb{V}^{\mathtt{L}}) \ \middle|\ \mathit{Secr}^{\eta}_{[\![P]\!]}(X)\ \right\}$;
- $\mathcal{K}_{P,\,(\eta),\,\phi}(id) = \left\{\ X \in \wp(\mathbb{V}^{\mathtt{L}}) \ \middle|\ \mathit{Secr}^{\eta,\phi}_{[\![P]\!]}(X)\ \right\}$.

Proof. In order to prove the thesis we have, first, to show that the two sets are Moore-families, i.e., closure operators. Afterwards, we have to show that they are the most concrete closures that make the program secret.
Let's consider the narrow case. In order to show that $\left\{\ X \ \middle|\ \mathit{Secr}^{\eta}_{[\![P]\!]}(X)\ \right\}$ is a Moore-family we show that it contains the intersection of its elements. Hence, consider $X, Y \in \left\{\ Z \ \middle|\ \mathit{Secr}^{\eta}_{[\![P]\!]}(Z)\ \right\}$ and consider $X \cap Y$. For the hypotheses on X and Y we have that

$$\begin{array}{c} \forall l \in \mathbb{V}^{\mathtt{L}}\ .\ (\exists Z \in \Upsilon^{\eta}_{[\![P]\!]}(l)\ .\ Z \subseteq X\ \Rightarrow\ \forall W \in \Upsilon^{\eta}_{[\![P]\!]}(l)\ .\ W \subseteq X \text{ and} \\ \exists Z \in \Upsilon^{\eta}_{[\![P]\!]}(l)\ .\ Z \subseteq Y\ \Rightarrow\ \forall W \in \Upsilon^{\eta}_{[\![P]\!]}(l)\ .\ W \subseteq Y) \end{array}$$

We have to prove that the same condition holds for $X \cap Y$. Therefore, suppose, for each $l \in \mathbb{V}^{\mathtt{L}}$, that $\exists Z \in \Upsilon^{\eta}_{[\![P]\!]}(l)\,.\, Z \subseteq X \cap Y$. This means that $Z \subseteq X$ and $Z \subseteq Y$. By the hypotheses, we have that $Secr^{\eta}_{[\![P]\!]}(X)$ and $Secr^{\eta}_{[\![P]\!]}(Y)$, therefore the previous conditions imply that $\forall W \in \Upsilon^{\eta}_{[\![P]\!]}(l)$ we have $W \subseteq X$ and $W \subseteq Y$, which is equivalent to saying that $W \subseteq X \cap Y$. Hence, we have proved $Secr^{\eta}_{[\![P]\!]}(X \cap Y)$. It is simple to verify that this result can be generalized to arbitrary intersections, therefore $\left\{\, X \,\middle|\, Secr^{\eta}_{[\![P]\!]}(X) \,\right\}$ is a Moore-family. Note that in the proof above we didn't used the fact that we were considering the narrow abstract non-interference, therefore the same result holds in the abstract case.
Now we have to prove that $\mathcal{S} \stackrel{\text{def}}{=} \left\{\, X \in \wp(\mathbb{V}^{\mathtt{L}}) \,\middle|\, Secr^{\eta}_{[\![P]\!]}(X) \,\right\}$ is such that $[\eta]P\ (\mathcal{S})$, i.e., such that $\eta(l_1) = \eta(l_2)$ implies $\mathcal{S}([\![P]\!](h_1, l_1)^{\mathtt{L}}) = \mathcal{S}([\![P]\!](h_2, l_2)^{\mathtt{L}})$. Suppose, towards a contradiction, that there exists h_1, h_2, l_1, l_2 such that $\eta(l_1) = \eta(l_2)$, with $\mathcal{S}([\![P]\!](h_1, l_1)^{\mathtt{L}}) \neq \mathcal{S}([\![P]\!](h_2, l_2)^{\mathtt{L}})$. Note that, both the elements $[\![P]\!](h_1, l_1)^{\mathtt{L}}, [\![P]\!](h_2, l_2)^{\mathtt{L}} \in \Upsilon^{\eta}_{[\![P]\!]}(l_1)$ being $\eta(l_1) = \eta(l_2)$, and, moreover, there exist $X_1, X_2 \in \wp(\mathbb{V}^{\mathtt{L}})$ such that $\mathcal{S}([\![P]\!](h_1, l_1)^{\mathtt{L}}) \stackrel{\text{def}}{=} X_1 \neq \mathcal{S}([\![P]\!](h_2, l_2)^{\mathtt{L}}) \stackrel{\text{def}}{=} X_2$, which means that $X_1, X_2 \in \mathcal{S}$, therefore $Secr^{\eta}_{[\![P]\!]}(X_1)$ and $Secr^{\eta}_{[\![P]\!]}(X_2)$ hold. At this point, if $X_1 \not\supseteq X_2$, then we have $[\![P]\!](h_1, l_1)^{\mathtt{L}} \in \Upsilon^{\eta}_{[\![P]\!]}(l_1)$ such that $[\![P]\!](h_1, l_1)^{\mathtt{L}} \subseteq X_1$, while $[\![P]\!](h_2, l_2)^{\mathtt{L}} \in \Upsilon^{\eta}_{[\![P]\!]}(l_1)$ such that $[\![P]\!](h_2, l_2)^{\mathtt{L}} \not\subseteq X_1$, which is a contradiction, for the hypothesis on X_1. If $X_1 \supseteq X_2$, then we have $X_2 \not\supseteq X_1$ and therefore we obtain the contradiction on X_2. Again, we can prove the same fact for abstract non-interference, simply by generalizing the proof above.
Finally, we have to prove that the secret closure obtained so far is the secret kernel, i.e., it is the most concrete closure that makes the program secret. Suppose this is false, towards a contradiction, namely suppose that there exists an abstract domain ρ such that $[\eta]P\ (\rho)$, with $\rho \not\subseteq \mathcal{S}$. We consider only the case $\rho \supseteq \mathcal{S}$, since otherwise we could consider the greatest lower bound $\rho \sqcap \mathcal{S}$ which is secret by Proposition 6.9(3). At this point, we can note that there exists $X \in \rho$ such that $X \notin \mathcal{S}$, i.e., such that $\neg Secr^{\eta}_{[\![P]\!]}(X)$. But this means that there exists a low input l such that $\exists Z \in \Upsilon^{\eta}_{[\![P]\!]}(l)$ with $Z \subseteq X$, and there exists $W \in \Upsilon^{\eta}_{[\![P]\!]}(l)$ such that $W \not\subseteq X$. Hence, we have that $Z = [\![P]\!](h_1, l_1)^{\mathtt{L}}$ for some $h_1, l_1 \in \mathbb{V}$, and $W = [\![P]\!](h_2, l_2)^{\mathtt{L}}$ for some $h_2, l_2 \in \mathbb{V}$, with $\eta(l_1) = \eta(l_2) = \eta(l)$, since both the sets are in $\Upsilon^{\eta}_{[\![P]\!]}(l)$. These facts imply that $\rho([\![P]\!](h_1, l_1)^{\mathtt{L}}) \subseteq X$, since $[\![P]\!](h_1, l_1)^{\mathtt{L}} \in X$, while $\rho([\![P]\!](h_2, l_2)^{\mathtt{L}}) \not\subseteq X$, since $[\![P]\!](h_2, l_2)^{\mathtt{L}} \notin X$. Therefore, $\rho([\![P]\!](h_1, l_1)^{\mathtt{L}}) \neq \rho([\![P]\!](h_2, l_2)^{\mathtt{L}})$, which is a contradiction, since we supposed that $[\eta]P\ (\rho)$. This means that such a closure cannot exists. Also in this case the proof for abstract non-interference is analogous.

In the following, we use the notation $\mathcal{S}^{\varepsilon}_{[\![P]\!]} : \wp(\wp(\mathbb{V}^{\mathtt{L}})) \longrightarrow \wp(\wp(\mathbb{V}^{\mathtt{L}}))$ to denote the predicate transformer $\mathcal{S}^{\varepsilon}_{[\![P]\!]}(\mathbb{X}) = \left\{\, X \in \mathbb{X} \,\middle|\, Secr^{\varepsilon}_{[\![P]\!]}(X) \,\right\}$, that transforms each domain $\mathbb{X}$ in the most concrete one that it contains and such that all its elements satisfy the predicate $Secr$. At this point we can define the secret kernels of a generic domain ρ in the following way:

Corollary 6.13. *Let $\eta \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$ and $\phi \in uco(\wp(\mathbb{V}^{\mathtt{H}}))$.*

- $\mathcal{K}_{P,\,[\eta]}(\rho) = \left\{\ X \in \rho \,\middle|\, \mathit{Secr}^{\eta}_{[\![P]\!]}(X)\ \right\}$;
- $\mathcal{K}_{P,\,(\eta),\,\phi}(\rho) = \left\{\ X \in \rho \,\middle|\, \mathit{Secr}^{\eta,\phi}_{[\![P]\!]}(X)\ \right\}$.

Proof. By definition, the secret kernels, parametric on the output observation ρ, are respectively: $\mathcal{K}_{P,\,[\eta]} = \lambda\rho.\, \mathcal{K}_{P,\,[\eta]}(id) \sqcup \rho$ and $\mathcal{K}_{P,\,(\eta),\,\phi} = \lambda\rho.\, \mathcal{K}_{P,\,(\eta),\,\phi}(id) \sqcup \rho$. We have to prove that this domains are, respectively, $\left\{\ X \in \rho \,\middle|\, \mathit{Secr}^{\eta}_{[\![P]\!]}(X)\ \right\}$ and $\left\{\ X \in \rho \,\middle|\, \mathit{Secr}^{\eta,\phi}_{[\![P]\!]}(X)\ \right\}$. This means that $\left\{\ X \in \rho \,\middle|\, \mathit{Secr}^{\eta}_{[\![P]\!]}(X)\ \right\}$ has to be equal to $\left\{\ X \in \wp(\mathbb{V}^{\mathtt{L}}) \,\middle|\, \mathit{Secr}^{\eta}_{[\![P]\!]}(X)\ \right\} \sqcup \rho$. Consider, now, an element belonging to this set, i.e., $Y \in \left\{\ X \in \rho \,\middle|\, \mathit{Secr}^{\eta}_{[\![P]\!]}(X)\ \right\}$, then it is in ρ and it is such that $\mathit{Secr}^{\eta}_{[\![P]\!]}(Y)$, namely it is in $\mathcal{K}_{P,\,[\eta]}(id) \sqcup \rho$. Analogously, the inverse inclusion holds, so we have the equality. Similarly, we can prove the same in the abstract case.

Therefore, we have that $\mathcal{K}$ is an abstract domain simplification (see Sect. 3.1, [61]), as we said above, namely $\mathcal{K} \in \mathit{uco}(\mathit{uco}(\wp(C)))$, mapping insecure abstract domains to the most concrete of their secure abstractions.

The problem with this characterization is that, if we want build this secret kernel, then we have to check the predicate *Secr* on all the possible sets of elements and this could be resource consuming, also for finite domains. Therefore, we would like to go deeper in the meaning of non-interference in order to identifies which elements are surely secret, which are never secret, and for which sets of elements we do have to check secrecy.

6.3.2 Deriving secret kernels

In this section, we analyze more deeply the meaning of non-interference in order to characterize three kind of sets: The sets of value that surely cannot generate interference, those that surely do generate interference and finally, those for which we have to check if they generate or not interference. As observed above, any secure abstraction has not to distinguish elements in the sets $\Upsilon^{\eta}_{[\![P]\!]}(l)$ [resp. $\Upsilon^{\eta,\,\phi}_{[\![P]\!]}(l)$], for each $l \in \mathbb{V}^{\mathtt{L}}$. In order to obtain this, we have to understand which are the smallest sets of values that need to be indistinguishable, and therefore that a secret abstraction has to contain in order to guarantee non-interference. In Fig. 6.2 (a) we depict what happens for narrow non-interference, indeed consider two elements $[\![P]\!](h_1, l_1)$ and $[\![P]\!](h_2, l_2)$ such that $h_1, h_2 \in \mathbb{V}^{\mathtt{H}}$, $l_1, l_2 \in \mathbb{V}^{\mathtt{L}}$ and $\eta(l_1) = \eta(l_2)$. Consider now an abstract domain ρ, then if we have an element $X \in \rho$, as depicted in the figure, we would have $\rho([\![P]\!](h_1, l_1)) = X \neq \rho([\![P]\!](h_2, l_2))$, and therefore we would have an interference. For this reason we note that the smallest elements, containing all the objects $[\![P]\!](h, l)$ with the same η property of l, that a secret abstract domain has to contain are $[\![P]\!](\mathbb{V}^{\mathtt{H}}, \eta(l))$. Each smaller set would distinguish objects that must be indistinguishable in order to guarantee non-interference. In Fig. 6.2 (b) we have a similar situation for abstract non-interference, and also in this case we note that the smallest elements, containing the all the objects $[\![P]\!](\phi(h), \eta(l))$ with h ranging over $\mathbb{V}^{\mathtt{H}}$, that a secret abstraction has to contain

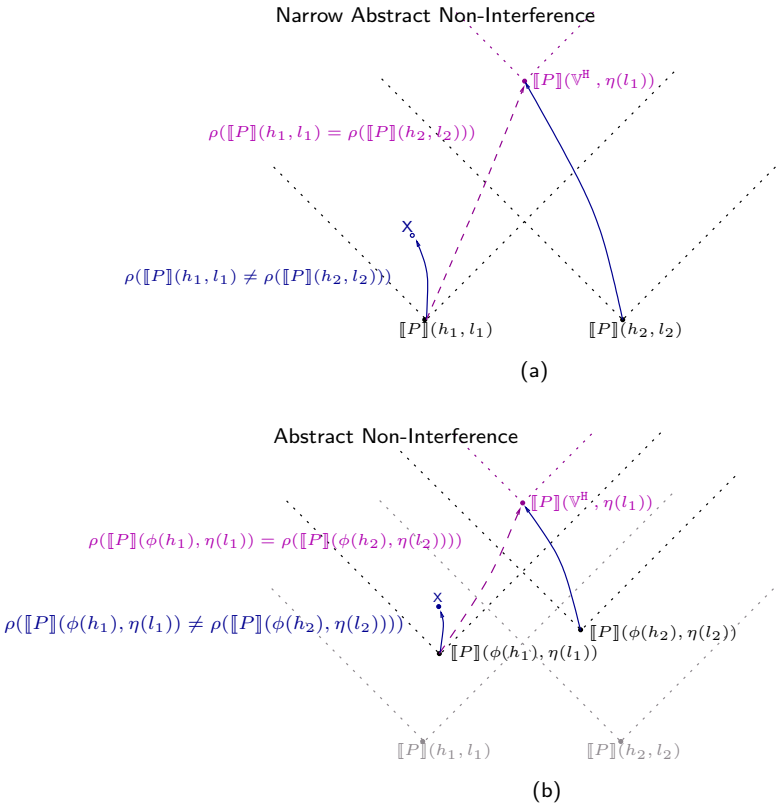


Fig. 6.2. Deriving attackers: The idea

are again $[\![P]\!](\mathbb{V}^{\mathtt{H}}, \eta(l))$. Let us collect all these sets of indistinguishable values, depending on public values, by defining, for any $\eta \in \mathit{uco}(\wp(\mathbb{V}^{\mathtt{L}}))$, the following family of sets of values:

$$\mathbb{D}_{[\![P]\!]}(\eta) \stackrel{\text{def}}{=} \left\{ \; [\![P]\!](\mathbb{V}^{\mathtt{H}}, \eta(y))^{\mathtt{L}} \;\middle|\; y \in \mathbb{V}^{\mathtt{L}} \; \right\}$$

The following example shows that the set $\mathbb{D}_{[\![P]\!]}(\eta)$ is not, in general, a Moore-family.

Example 6.14. Consider the program fragment

$$P \stackrel{\text{def}}{=} \textbf{while } h \textbf{ do } l := l * 2;\ h := 0 \textbf{ endw}$$

with typing $\langle h : \mathtt{H}, l : \mathtt{L} \rangle$ and $\mathbb{V} = \mathbb{N}$. Its single-step standard denotational semantics is:

$$[\![P]\!](h, l) = \begin{cases} (h, l) & \text{if } h = 0 \\ (0, l * 2) & \text{otherwise} \end{cases}$$

Then, for each $l \in \mathbb{N}$, we have $[\![P]\!](\mathbb{N}, l)^{\mathtt{L}} = \{l, 2l\}$, hence we can obtain the set $\mathbb{D}_{[\![P]\!]}(\mathit{id}) = \{\{0\}, \{1, 2\}, \{2, 4\}, \{3, 6\}, \{4, 8\}, \ldots\}$, which is not a Moore-family since, for example, it does not contain $\{1, 2\} \cap \{2, 4\} = \{2\}$.

The construction of the set $\mathbb{D}_{[\![P]\!]}$ allows to characterize which elements can never be contained in a secret abstract domain. Indeed each set of values that contains any $[\![P]\!](\phi(h), \eta(l))$ [$[\![P]\!](h, l)$ in the narrow case], but that does not contain the set $[\![P]\!](\mathbb{V}^{\mathtt{H}}, \eta(l))$, will surely cause an interference, making the program insecure. In Fig. 6.3 these point are those in the empty area between the two lines. On the other hand, all the sets that do not contain any $[\![P]\!](\phi(h), \eta(l))$ [$[\![P]\!](h, l)$ in the narrow case] cannot create any problem, since they are not able to distinguish values that must be indistinguishable. In Fig. 6.3 these points are those in the filled area.

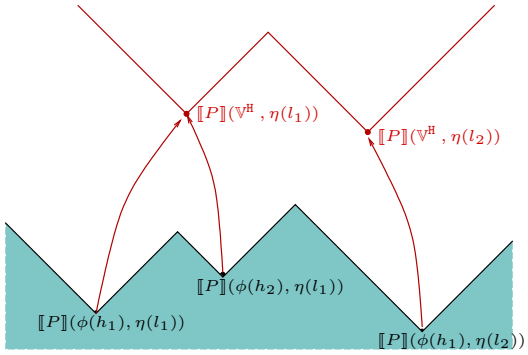


Fig. 6.3. Irrelevant elements

These latter elements of the concrete domain of values are always secret and need not to be involved in the verification of secrecy. We call these elements *irrelevant* and we can formally define them in the following way:

$$Irr^{\phi,\,\eta}_{[\![P]\!]} \stackrel{\text{def}}{=} \left\{\ X \in \wp(\mathbb{V}^{\mathtt{L}}) \,\middle|\, \forall h \in \mathbb{V}^{\mathtt{H}},\ l \in \mathbb{V}^{\mathtt{L}}\ .X \notin \uparrow([\![P]\!](\phi(h), \eta(l))^{\mathtt{L}})\ \right\}$$

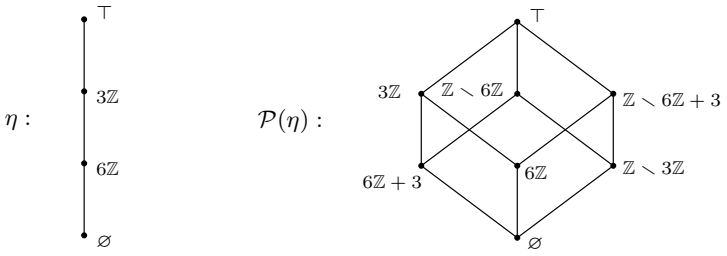
Irrelevant elements are also important in order to characterize the elements that reveal deceptive flows in narrow abstract non-interference. Suppose the output closure we are checking for secrecy contains an element $X \subset [\![P]\!](h, \eta(l))^{\mathtt{L}}$, for some $h \in \mathbb{V}^{\mathtt{H}}$, $l \in \mathbb{V}^{\mathtt{L}}$, and $X \neq \varnothing$, then clearly there exists $l_1 \in \eta(l)$ such that $[\![P]\!](h, l_1)^{\mathtt{L}} \in X$ and $l_2 \in \eta(l)$ such that $[\![P]\!](h, l_2)^{\mathtt{L}} \notin X$. In this situation, the revealed flow for $[\![P]\!](h, l_1)^{\mathtt{L}}$ and $[\![P]\!](h, l_2)^{\mathtt{L}}$, due to the presence of X in the closure, is clearly a deceptive flow. These sets, revealing deceptive flows, are in general irrelevant elements in the abstract non-interference case. This because in the abstract non-interference case we are interested in abstracting $[\![P]\!](h, \eta(l))^{\mathtt{L}}$. Since $X \subset [\![P]\!](h, \eta(l))^{\mathtt{L}} \subseteq [\![P]\!](\phi(h), \eta(l))^{\mathtt{L}}$, X falls in the set of irrelevant elements for abstract non-interference. In particular, the sets X, which reveal deceptive flows, are contained in $Irr^{id,\,\eta}_{[\![P]\!]} \smallsetminus Irr^{id,\,id}_{[\![P]\!]}$. Note also that, for the narrow case we have to consider the more concrete closure that induces the same partition of values as $\eta \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$, i.e., $\mathcal{P}(\eta) \stackrel{\text{def}}{=} \curlyvee\left(\left\{\ [l]_\eta \,\middle|\, l \in \mathbb{V}^{\mathtt{L}}\ \right\}\right)$, where $[l]_\eta \stackrel{\text{def}}{=} \left\{\ y \,\middle|\, \eta(l) = \eta(y)\ \right\}$.

This is essential in order get the secret kernel, i.e., the most concrete domain β such that $[\eta]P$ (β). The idea is that $\mathcal{P}(\eta)$ is the most concrete closure such that for any $y \in \mathcal{P}(\eta)(l)$: $\mathcal{P}(\eta)(l) = \mathcal{P}(\eta)(y)$, while in general $\eta(y) \subseteq \eta(l)$. The following example shows that, if we don't consider partitioning closures (Sect. 2.2.3) modeling the input, then the construction does not allow to obtain the secret kernel, since we are not sure to find the most concrete harmless attacker.

Example 6.15. Consider the following program fragment:

$$P \stackrel{\text{def}}{=} \textbf{while } h \textbf{ do } l := l + 6; h := h - 1 \textbf{ endw}, \qquad [\![P]\!](h, l) = l + 6h$$

with security typing $t = \langle h : \mathtt{H}, l : \mathtt{L} \rangle$ and $\mathbb{V} = \mathbb{Z}$. Let us consider $[\eta][\![P]\!]$ (id), where $\eta(\wp(\mathbb{Z})) = \{\mathbb{Z}, 3\mathbb{Z}, 6\mathbb{Z}, \varnothing\}$.



We show that if we consider η instead of $\mathcal{P}(\eta)$ we don't find the secret kernel of the program. First of all, note that

$$\begin{aligned} \forall l \in 6\mathbb{Z}\,.\ & \Upsilon^{\eta}_{[\![P]\!]}(l) = 6\mathbb{Z} \\ \forall l.\ \eta(l) = 3\mathbb{Z}.\ & \Upsilon^{\eta}_{[\![P]\!]}(l) = 6\mathbb{Z} + 3 \\ \forall l \notin 3\mathbb{Z}\,.\ & \Upsilon^{\eta}_{[\![P]\!]}(l) = \bigcup \{\, 6\mathbb{Z} + l \,|\, l \notin 3\mathbb{Z} \,\} = \mathbb{Z} \smallsetminus 3\mathbb{Z} \end{aligned}$$

At this point, $\mathbb{D}_{[\![P]\!]}(\eta) = \{6\mathbb{Z}, 3\mathbb{Z}, \mathbb{Z}\}$, and it is simple to verify that

$$\mathcal{S}^{\eta}_{[\![P]\!]}(\uparrow(\mathbb{D}_{[\![P]\!]}(\eta))) = \{\mathbb{Z}, \mathbb{Z} \smallsetminus (3(2\mathbb{Z}+1)), 3\mathbb{Z}, 6\mathbb{Z}\}$$

Let us consider now $\mathcal{P}(\eta) = \curlyvee(\{\mathbb{Z}, \mathbb{Z} \smallsetminus 3\mathbb{Z}, 3(2\mathbb{Z}+1), 6\mathbb{Z}, \varnothing\})$. Then it is simple to verify that $\mathbb{D}_{[\![P]\!]}(\mathcal{P}(\eta)) = \curlyvee(\{6\mathbb{Z}, 3(2\mathbb{Z}+1), \mathbb{Z} \smallsetminus 3\mathbb{Z}\})$. At this point, we have that $\mathcal{S}^{\eta}_{[\![P]\!]}(\uparrow(\mathbb{D}_{[\![P]\!]}(\mathcal{P}(\eta)))) = \mathbb{D}_{[\![P]\!]}(\mathcal{P}(\eta))$ which strictly contains the set $\mathcal{S}^{\eta}_{[\![P]\!]}(\uparrow(\mathbb{D}_{[\![P]\!]}(\eta)))$ obtained above.

Finally, we have to characterize the elements for which we do have to check noninterference. These are the sets that contain the objects of the kind $[\![P]\!](\mathbb{V}^{\mathtt{H}}, \eta(l))$, for which we cannot say anything a priori. We defined above the set $\mathbb{D}_{[\![P]\!]}$, then the sets for which we have to check secrecy are those contained in $\uparrow(\mathbb{D}_{[\![P]\!]}(\eta))$, which is the upper area in Fig. 6.3. Therefore, we have to verify, for each one of these sets, whether it satisfies the predicate *Secr*. The following example shows how to construct the set $\mathcal{S}^{\varepsilon}_{[\![P]\!]}(\uparrow(\mathbb{D}_{[\![P]\!]}(\eta)))$, namely how to detect the secret elements in $\uparrow(\mathbb{D}_{[\![P]\!]}(\eta))$.

Example 6.16. Let us consider the Example 6.14. We build $\mathcal{S}^{id}_{[\![P]\!]}(\uparrow(\mathbb{D}_{[\![P]\!]}(id)))$ by checking which $X \in \uparrow(\mathbb{D}_{[\![P]\!]}(id))$ are such that $Secr^{id}_{P}(X)$ holds. Note that, for each $l \in \mathbb{V}^{\mathtt{L}} = \mathbb{N}$, the elements that have to be indistinguishable under a secret abstraction, are collected in the sets:

$$\Upsilon^{id}_{[\![P]\!]}(l) = \{l, 2l\}.$$

Let us define $\{2\}^{\mathbb{N}} \stackrel{\text{def}}{=} \{\ 2^n \,|\, n \in \mathbb{N}\ \}$ [89]. We can prove that

$$\mathcal{S}^{\varepsilon}_{[\![P]\!]}(\uparrow(\mathbb{D}_{[\![P]\!]}(id))) = \bigvee\big(\{\ n\{2\}^{\mathbb{N}} \,|\, n \in 2\mathbb{N}+1\ \} \cup \{\{0\}\}\big).$$

Indeed, note that, if $n \in 2\mathbb{N}$, i.e., $n = 2k$ for some $k \in \mathbb{N}$, then $2k, k \in \Upsilon^{id}_{[\![P]\!]}(k)$, $2k \in n\{2\}^{\mathbb{N}}$, and $k \notin n\{2\}^{\mathbb{N}}$, therefore we always have that $\neg Secr^{id}_{[\![P]\!]}(2k\{2\}^{\mathbb{N}})$. Suppose, now,

$$X \in \uparrow\big(\{\ n\{2\}^{\mathbb{N}} \,|\, n \in \mathbb{N}\ \}\big)$$

If $X \supseteq 2k\{2\}^{\mathbb{N}}$, for some $k \in \mathbb{N}$, then X is not secret for the considerations above. Therefore, suppose $X \not\supseteq 2k\{2\}^{\mathbb{N}}$, for each $k \in \mathbb{N}$.
Consider, under these hypotheses, that $X \notin \bigvee\big(\{\ n\{2\}^{\mathbb{N}} \,|\, n \in 2\mathbb{N}+1\ \} \cup \{\{0\}\}\big)$. This means that there exists a set $m\{2\}^{\mathbb{N}}$ which is not entirely contained in X, i.e., such that $X \cap m\{2\}^{\mathbb{N}} \neq \varnothing$ and $X \not\supseteq m\{2\}^{\mathbb{N}}$. But this means that X is not secret since there must exists an element $h \in (X \cap m\{2\}^{\mathbb{N}})$ such that $2h \notin (X \cap m\{2\}^{\mathbb{N}})$, or viceversa, otherwise $X \supseteq m\{2\}^{\mathbb{N}}$, namely X "brakes" $\Upsilon(h)$.
Finally, if $X \in \bigvee\big(\{\ n\{2\}^{\mathbb{N}} \,|\, n \in 2\mathbb{N}+1\ \} \cup \{\{0\}\}\big)$, then it cannot brake any $\Upsilon(n)$. In particular, if n is odd, then $\Upsilon(n)$ is contained only in $n\{2\}^{\mathbb{N}}$, while if n is even, $\exists h, k\,.\, n = 2^k h$ with h odd, and in this case $\Upsilon(n)$ is contained only in $h\{2\}^{\mathbb{N}}$. Therefore, we proved that the secret elements of $\uparrow(\mathbb{D}_{[\![P]\!]}(id)))$ are $\mathcal{S}^{\varepsilon}_{[\![P]\!]}(\uparrow(\mathbb{D}_{[\![P]\!]}(id))) = \bigvee\big(\{\ n\{2\}^{\mathbb{N}} \,|\, n \in 2\mathbb{N}+1\ \} \cup \{\{0\}\}\big)$.

In Figure 6.4, we have an example on how $\mathcal{S}^{\varepsilon}_{[\![P]\!]}(\uparrow(\mathbb{D}_{[\![P]\!]}(\eta)))$ is constructed. Assume $\mathbb{D}_{P}(\eta) = \{d_1, d_2, d_3, d_4\}$. We note that d_2 brakes the set $\Upsilon^{\varepsilon}_{[\![P]\!]}(l_1)$ and d_3 cause the brake of $\Upsilon^{\varepsilon}_{[\![P]\!]}(l_4)$, therefore $Secr(d_2)$ and $Secr(d_3)$ don't hold. The operation $\mathcal{S}^{\varepsilon}_{[\![P]\!]}$ erases d_2 and d_3 from $\uparrow(\mathbb{D}_{[\![P]\!]}(\eta))$. With similar argument we erase from $\uparrow(\mathbb{D}_{[\![P]\!]}(\eta))$ all the points that do not satisfy $Secr$. The remaining points are circled in the picture, and they form the set $\mathcal{S}^{\varepsilon}_{[\![P]\!]}(\uparrow(\mathbb{D}_{[\![P]\!]}(\eta)))$. We call these points *relevant elements* of the secret kernel.

We are now in the position to characterize which are the elements that, following our construction, are secret and *should form* the secret kernel:

$$\begin{aligned} [\eta][\![P]\!]\ (id) &\stackrel{\text{def}}{=} \mathcal{S}^{\eta}_{[\![P]\!]}(\uparrow(\mathbb{D}_{[\![P]\!]}(\mathcal{P}(\eta)))) \cup Irr^{id,\,id}_{[\![P]\!]} \\ (\eta)[\![P]\!]\ (\phi \rightsquigarrow id) &\stackrel{\text{def}}{=} \mathcal{S}^{\eta,\,\phi}_{[\![P]\!]}(\uparrow(\mathbb{D}_{[\![P]\!]}(\eta))) \cup Irr^{\phi,\,\eta}_{[\![P]\!]} \end{aligned}$$

Note that these definitions introduce a slight abuse of notation: Indeed, while $(\eta)[\![P]\!]\ (\phi \rightsquigarrow id) \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$ is an abstract domain, $(\eta)P\ (\phi \rightsquigarrow id)$ is a program property. The same holds in the narrow case. The following result says that the

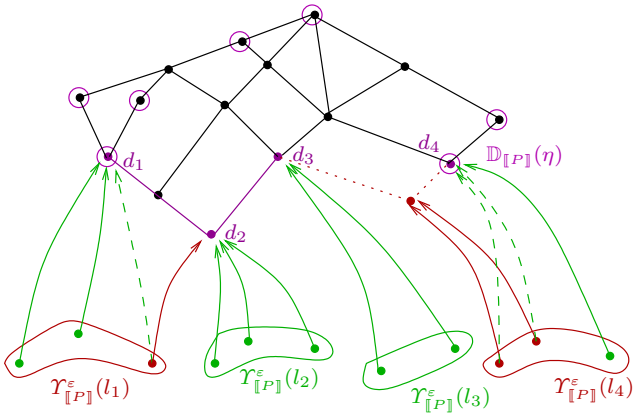


Fig. 6.4. The construction of $\mathcal{S}^{\varepsilon}_{[\![P]\!]}(\uparrow(\mathbb{D}_{[\![P]\!]}(\eta)))$.

domains $[\eta][\![P]\!]$ (id) and $(\eta)[\![P]\!]$ $(\phi \rightsquigarrow id)$ are exactly the secret kernels of a program P as regards a given attacker, i.e., a given input observation η for narrow non-interference, or a given input observation η and private property ϕ for abstract non-interference.

Theorem 6.17. *Let* $\eta, \rho \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$ *and* $\phi \in uco(\wp(\mathbb{V}^{\mathtt{H}}))$.

1. $\mathcal{K}_{P,\,[\eta]}(id) = [\eta][\![P]\!]$ (id);
2. $\mathcal{K}_{P,\,(\eta),\,\phi}(id) = (\eta)[\![P]\!]$ $(\phi \rightsquigarrow id)$.

Proof. Let us consider the narrow case. We can obtain the proof for abstract non-interference simply by substituting the closure $\mathcal{P}(\eta)$ with η. Consider narrow non-interference, we have to prove that $\mathcal{S}^{\eta}_{[\![P]\!]}(\uparrow(\mathbb{D}_{[\![P]\!]}(\mathcal{P}(\eta)))) \cup Irr_{[\![P]\!]}$ contains all and only the elements that are contained in the set $\left\{\ X \in \wp(\mathbb{V}^{\mathtt{L}}) \,\middle|\, Secr^{\eta}_{[\![P]\!]}(X)\ \right\}$. Let us prove the two inclusions separately.

($\supseteq$) Suppose that $Secr^{\eta}_{[\![P]\!]}(X)$, and suppose, towards a contradiction, that $X \notin \mathcal{S}^{\eta}_{[\![P]\!]}(\uparrow(\mathbb{D}_{[\![P]\!]}(\mathcal{P}(\eta)))) \cup Irr_{[\![P]\!]}$. This last fact holds iff $X \notin \mathcal{S}^{\eta}_{[\![P]\!]}(\uparrow(\mathbb{D}_{[\![P]\!]}(\mathcal{P}(\eta))))$ and $X \notin Irr_{[\![P]\!]}$. The first condition holds iff $X \notin \uparrow(\mathbb{D}_{[\![P]\!]}(\mathcal{P}(\eta)))$ or $\neg Secr(X)$, but since by hypothesis $Secr^{\eta}_{[\![P]\!]}(X)$, this implies $X \notin \uparrow(\mathbb{D}_{[\![P]\!]}(\mathcal{P}(\eta)))$. On the other hand, the second condition holds iff $X \supseteq [\![P]\!](h, l)$ for some $h, l \in \mathbb{V}$. Namely we have that X is such that $X \supseteq [\![P]\!](h, l)$ and $X \notin \uparrow(\mathbb{D}_{[\![P]\!]}(\mathcal{P}(\eta)))$ which implies, in particular that $X \notin \mathbb{D}_{[\![P]\!]}(\mathcal{P}(\eta))$. Moreover, $X \not\supseteq [\![P]\!](\mathbb{V}, \mathcal{P}(\eta)(l))$, since $X \notin \uparrow(\mathbb{D}_{[\![P]\!]}(\mathcal{P}(\eta)))$. This implies then that $\exists h' \in \mathbb{V}^{\mathtt{H}}, l' \in \mathbb{V}^{\mathtt{L}}$ such that $\mathcal{P}(\eta)(l') = \mathcal{P}(\eta)(l)$ and such that $X \not\supseteq [\![P]\!](h', l')^{\mathtt{L}}$. But $[\![P]\!](h', l')^{\mathtt{L}}, [\![P]\!](h, l) \in \Upsilon^{\eta}_{[\![P]\!]}(l)$ since $\eta(l) = \eta(l')$, therefore we have $\neg Secr(X)$, which is a contradiction.

($\subseteq$) Suppose $X \in \mathcal{S}^{\eta}_{[\![P]\!]}(\uparrow(\mathbb{D}_{[\![P]\!]}(\mathcal{P}(\eta)))) \cup Irr_{[\![P]\!]}$. If $X \in \mathcal{S}^{\eta}_{[\![P]\!]}(\uparrow(\mathbb{D}_{[\![P]\!]}(\mathcal{P}(\eta))))$ then $Secr^{\eta}_{[\![P]\!]}(X)$ by definition of $\mathcal{F}$. If, instead, $X \in Irr_{[\![P]\!]}$, then, by definition of

$Irr_{[\![P]\!]}$, for each $[\![P]\!](h,l)^{\mathtt{L}}$ we have that $X \not\supseteq [\![P]\!](h,l)^{\mathtt{L}}$, namely $\forall l \,.\, \forall Z \in \Upsilon^{\eta}_{[\![P]\!]} \,.\, Z \subseteq X$, which means that $Secr^{\eta}_{[\![P]\!]}(X)$ holds.

The following examples show the complete construction for both narrow and abstract non-interference.

Example 6.18. Consider the program fragment:

$$P \stackrel{\text{def}}{=} \textbf{while } h \textbf{ do } l := l * 2;\ h := h - 1 \textbf{ endw}$$

with security typing $\langle h : \mathtt{H}\,, l : \mathtt{L}\,\rangle$ and $\mathbb{V} = \mathbb{N}$. We can note that this fragment is secure as regards the output property *Sign*, while it is not secure as regards the output property *Parity*, since the semantics of this program may change the parity of l. For instance if the input l is odd, and the while is executed, then the output is even. We find here the most concrete property (that clearly has to contain *Sign*) that makes P secure. The denotational semantics of the program is:

$$[\![P]\!](h,l) = \begin{cases} (h,l) & \text{if } h = 0 \\ (0, l * 2^h) & \text{otherwise} \end{cases}$$

Let us compute the closure $[id][\![P]\!]\ (id)$. If $l = 1$ we have $[\![P]\!](\mathbb{V}^{\mathtt{H}}, 1)^{\mathtt{L}} = \{2\}^{\mathbb{N}}$, if $l = 2$ then $[\![P]\!](\mathbb{V}^{\mathtt{H}}, 2)^{\mathtt{L}} = 2\{2\}^{\mathbb{N}}$, and for each $l \in \mathbb{V}^{\mathtt{L}}$ we have $[\![P]\!](\mathbb{V}^{\mathtt{H}}, l)^{\mathtt{L}} = l\{2\}^{\mathbb{N}}$, this means that

$$\forall n \in \mathbb{N}.\, \Upsilon^{id}_{[\![P]\!]}(n) = n\{2\}^{\mathbb{N}} \text{ and } \mathbb{D}_{[\![P]\!]}(id) = \left\{\ n\{2\}^{\mathbb{N}} \,\middle|\, n \in \mathbb{N}\ \right\}$$

Note that, the only secret elements in $\mathbb{D}_{[\![P]\!]}(id)$ are $\left\{\ n\{2\}^{\mathbb{N}} \,\middle|\, n \in 2\mathbb{N}+1\ \right\} \cup \{\{0\}\}$. In fact, for each n we have $n\{2\}^{\mathbb{N}} \supseteq 2n\{2\}^{\mathbb{N}}$, which implies that $n \in \Upsilon(n)$. But $n \notin 2n\{2\}^{\mathbb{N}}$, while $2n \in \Upsilon(n)$ and $2n \in 2n\{2\}^{\mathbb{N}}$. Hence $\neg Secr^{id}_{[\![P]\!]}(2n\{2\}^{\mathbb{N}})$. Hence $\mathcal{S}^{id}_{[\![P]\!]}$ can only select elements from $\uparrow\left(\left\{\ n\{2\}^{\mathbb{N}} \,\middle|\, n \in 2\mathbb{N}+1\ \right\} \cup \{\{0\}\}\right)$.
Following what we showed in Example 6.16, we obtain that the set resulting from the application of $\mathcal{S}^{id}_{[\![P]\!]}$ is exactly the set

$$\mathcal{S}^{id}_{[\![P]\!]}(\uparrow(\mathbb{D}_{[\![P]\!]}(id))) = \bigvee\left(\left\{\ n\{2\}^{\mathbb{N}} \,\middle|\, n \in 2\mathbb{N}+1\ \right\} \cup \{\{0\}\}\right)$$

In particular, consider $X \in \uparrow\left(\left\{\ n\{2\}^{\mathbb{N}} \,\middle|\, n \in \mathbb{N}\ \right\}\right)$. If $X \supseteq 2n\{2\}^{\mathbb{N}}$, then we proved above that X is not secret. If $X \notin \bigvee\left(\left\{\ n\{2\}^{\mathbb{N}} \,\middle|\, n \in 2\mathbb{N}+1\ \right\} \cup \{\{0\}\}\right)$, then for each $k \in \mathbb{N}$ we have $X \not\supseteq 2k\{2\}^{\mathbb{N}}$ and $X \supseteq n\{2\}^{\mathbb{N}}$, with n odd. This means that there exists a set $m\{2\}^{\mathbb{N}}$ such that $X \cap m\{2\}^{\mathbb{N}} \neq \varnothing$ and $X \not\supseteq m\{2\}^{\mathbb{N}}$. But this implies that X is not secret since it brakes the set $\Upsilon(m)$. Finally, if $X \in \bigvee\left(\left\{\ n\{2\}^{\mathbb{N}} \,\middle|\, n \in 2\mathbb{N}+1\ \right\} \cup \{\{0\}\}\right)$, then it does not brake any $\Upsilon(m)$. Indeed, if m is odd, then it is contained only in $m\{2\}^{\mathbb{N}}$, while if it is even, then $m = 2k$ with k odd, and it is contained only in $k\{2\}^{\mathbb{N}}$. Moreover, note that in this case the set of irrelevants is $\{\varnothing\}$. The resulting set, by Theorem 6.17, is the most concrete domain making P secure.

Example 6.19. Let us consider the program fragment:

$$P \stackrel{\text{def}}{=} l := l + (h \ mod \ 3);$$

where *mod* is the rest of the integer division, the security typing is $\langle h : \mathtt{H}, l : \mathtt{L} \rangle$ and $\mathbb{V} = \mathbb{Z}$. We would like that $(\eta)P\ (id \rightsquigarrow\!\!\!\!| id)$, where η is the abstract domain $\eta(\wp(\mathbb{Z})) = \{\mathbb{Z}, [2,4], [5,8], \{5\}, \varnothing\}$. In order to build the set $\mathbb{D}_{[\![P]\!]}(id)$, we have to compute the elements $[\![P]\!](\mathbb{V}^{\mathtt{H}}, \eta(l))$:

$$\begin{aligned} [\![P]\!](\mathbb{Z}, \{5\})^{\mathtt{L}} &= [5,7] \\ [\![P]\!](\mathbb{Z}, [2,4])^{\mathtt{L}} &= [2,6] \\ [\![P]\!](\mathbb{Z}, [5,8])^{\mathtt{L}} &= [5,10] \\ [\![P]\!](\mathbb{Z}, \mathbb{Z})^{\mathtt{L}} &= \mathbb{Z} \end{aligned}$$

Therefore $\mathbb{D}_{[\![P]\!]}(\eta) = \{[5,7], [2,6], [5,10], \mathbb{Z}\}$.
On the other hand, we have that

$$\begin{aligned} \forall l.\, \eta(l) = [2,4].\, \Upsilon^{\eta,\,id}_{[\![P]\!]}(l) &= \{[2,4], [3,5], [4,6]\} \\ \forall l.\, \eta(l) = [5,8].\, \Upsilon^{\eta,\,id}_{[\![P]\!]}(l) &= \{[5,8], [6,9], [7,10]\} \\ \Upsilon^{\eta,\,id}_{[\![P]\!]}(5) &= \{5,6,7\} \\ \forall l.\, \eta(l) \notin \{[2,4], [5,8], \{5\}\}.\, \Upsilon^{\eta,\,id}_{[\![P]\!]}(l) &= \{\mathbb{Z}\} \end{aligned}$$

Note that these last sets cannot create problems to secrecy. It is possible to verify that the resulting abstract domain $\mathcal{S}^{\eta,\,id}_{[\![P]\!]}(\uparrow(\mathbb{D}_{[\![P]\!]}(\eta)))$ is the following collection of objects:

$$\left\{ Y \in \wp(\mathbb{V}) \;\middle|\; \begin{array}{l} (Y \supseteq [2,10]) \vee \\ (Y \not\supseteq [2,10],\ Y \supseteq [2,7],\ Y \cap [5,10] \notin \{[5,9], [5,8], [5,8] \cup \{10\}\}) \vee \\ (Y \not\supseteq [2,10],\ Y \supseteq [5,10],\ Y \cap [2,7] \notin \{[4,7], [3,7], [4,7] \cup \{2\}\}) \end{array} \right\}$$

These objects guarantee non-interference since they are all the elements that do not brake the sets of indistinguishable value, i.e., the sets $\{[2,4], [3,5], [4,6]\}$, $\{[5,8], [6,9], [7,10]\}$, and $\{5,6,7\}$.

6.3.3 Approximating the secret kernel

It is clear that the construction above is really complex since it requires the denotational semantics of the program. An approximation can be introduced in the derivation of the secret kernel by separately analyzing program fragments. In the following, we show two methods for approximating the secret kernel of a closure ρ for a program P, by inductively deriving the kernels of program components.

Bounded iterations.

Let $[\![P]\!]^{\langle n \rangle}$ represents the partial semantics of the program at the n-th step of evaluation, supposing that all **while**'s are unfolded: If $P = c_0; c_1; \ldots; c_m$, then for any n define:

$$\begin{array}{lll} [\![P]\!]^{\langle 0\rangle} & \stackrel{\text{def}}{=} & [\![c_0]\!] \\ [\![P]\!]^{\langle n+1\rangle} & \stackrel{\text{def}}{=} & [\![c_{n+1}]\!] \circ [\![P]\!]^{\langle n\rangle} \end{array}$$

For instance, $[\![P]\!]^{\langle 2\rangle}$ is the semantics of $c_0; c_1; c_2$, i.e., $[\![P]\!]^{\langle 2\rangle} = [\![c_2]\!] \circ [\![c_1]\!] \circ [\![c_0]\!]$. We define an abstract domain transformer $\mathcal{K}^{\text{par}}_{P,\,[\eta]}$ [resp. $\mathcal{K}^{\text{par}}_{P,\,(\eta),\,\phi}$] denoting the common abstraction among all the domains ρ_i such that the first i-steps are $\langle \eta, \rho_i\rangle$-*NSecret*, [resp. $\langle \eta, \phi, \rho_i\rangle$-*Secret*] i.e., for any $i \leq n$: $\rho_i = \ [\eta][\![P]\!]^{\langle i\rangle}\ (id)$ [resp. $(\eta)[\![P]\!]^{\langle n\rangle}\ (\phi \rightsquigarrow \rho_i)$], it is defined in the following way:

$$\begin{array}{l} \mathcal{K}^{\text{par}}_{P,\,[\eta]} \stackrel{\text{def}}{=} \lambda\rho.\ \rho \sqcup \bigsqcup_{i\leq n}\ [\eta][\![P]\!]^{\langle i\rangle}\ (id) \\ \mathcal{K}^{\text{par}}_{P,\,(\eta),\,\phi} \stackrel{\text{def}}{=} \lambda\rho.\ \rho \sqcup \bigsqcup_{i\leq n}\ (\eta)[\![P]\!]^{\langle i\rangle}\ (\phi \rightsquigarrow id) \end{array}$$

It is clear that $\mathcal{K}_{P,\,[\eta]} \sqsubseteq \mathcal{K}^{\text{par}}_{P,\,[\eta]}$. By Proposition 6.9, $[\eta]P\ (\mathcal{K}^{\text{par}}_{P,\,[\eta]}(\rho))$ still holds. The same holds in the abstract non-interference case. In this case we find the most concrete output attacker which is harmless as regards each step of computation of the program. As we will see, when we introduce timing channels, this corresponds to saying that the program is secure even if the attacker is time-sensitive, i.e., if it is able to observe also the time elapsed (see Sect. 5.4.2 for timing channels and Chap. 9 for timed abstract non-interference).

Independent composition.

An even coarser approximation of the secret kernel of ρ can be obtained by considering the most concrete abstraction making all statements in P $\langle \eta, \rho\rangle$-*NSecret* [resp. $\langle \eta, \phi, \rho\rangle$-*Secret*]. Suppose that for each statement c in P, $[\![c]\!]$ is the denotational semantics of c. Let $P = c_0; c_1; \ldots; c_m$ and define:

$$\begin{array}{l} \mathcal{K}^{\text{sbs}}_{P,\,[\eta]} \stackrel{\text{def}}{=} \lambda\rho.\ \rho \sqcup \bigsqcup_{i\leq m}\ [\eta][\![c_i]\!]\ (id) \\ \mathcal{K}^{\text{sbs}}_{P,\,(\eta),\,\phi} \stackrel{\text{def}}{=} \lambda\rho.\ \rho \sqcup \bigsqcup_{i\leq n}\ (\eta)[\![c_i]\!]\ (\phi \rightsquigarrow id) \end{array}$$

Intuitively $\mathcal{K}^{\text{sbs}}_{P,\,[\eta]}$ [resp. $\mathcal{K}^{\text{sbs}}_{P,\,(\eta),\,\phi}$] is the property which is not disclosed with respect to each fragment of the program P relatively to sequential composition. Note that, in this case, if there is at least one statement c such that $[\eta][\![c]\!]\ (id) = \{\top\}$ then $\mathcal{K}^{\text{sbs}}_{P,\,[\eta]} = \{\top\}$. Also in this case we have $\mathcal{K}_{P,\,[\eta]} \sqsubseteq \mathcal{K}^{\text{sbs}}_{P,\,[\eta]}$. It is clear that this is an upper approximation of both $\mathcal{K}_{P,\,[\eta]}$ and $\mathcal{K}^{\text{par}}_{P,\,[\eta]}$ as shown in Figure 6.5:

$$\mathcal{K}_{P,\,[\eta]} \sqsubseteq \mathcal{K}^{\text{par}}_{P,\,[\eta]} \sqsubseteq \mathcal{K}^{\text{sbs}}_{P,\,[\eta]}$$

The same holds in the abstract non-interference case. By Proposition 6.9, we have that the predicates $[\eta]P\ (\mathcal{K}^{\text{sbs}}_{P,\,[\eta]}(\rho))$ and $(\eta)P\ (\phi \rightsquigarrow \mathcal{K}^{\text{sbs}}_{P,\,[\eta]}(\rho))$ still hold. The following example shows, in the narrow abstract non-interference case, where the relations $\mathcal{K}_{P,\,[\eta]} \sqsubseteq \mathcal{K}^{\text{par}}_{P,\,[\eta]} \sqsubseteq \mathcal{K}^{\text{sbs}}_{P,\,[\eta]}$ are strict inclusions.

Example 6.20. Consider the program fragment

$$P \ \stackrel{\text{def}}{=}\ l := 2 * h;\ l := l * h$$

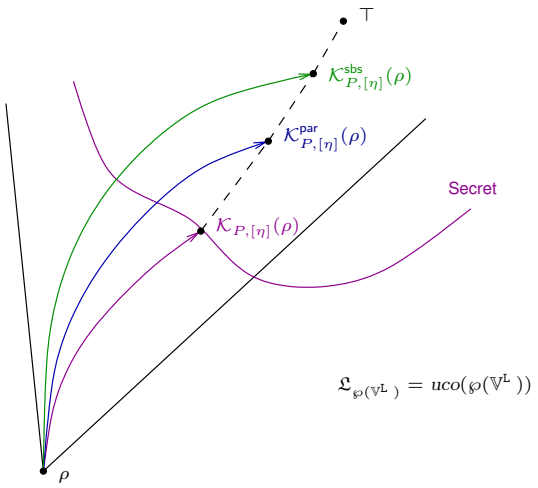


Fig. 6.5. Deriving secret kernels

with typing $\langle h : \mathtt{H}, l : \mathtt{L} \rangle$ and $\mathbb{V} = \mathbb{Z}$ and $[\![P]\!](h, l) = (h, 2{*}h^2)$. Consider $[id]P\ (id)$, then we can show that

$$\mathbb{D}_{[\![P]\!]}(id) = \{\!\{\ 2n^2 \,|\, n \in \mathbb{Z}\ \}\!\}$$

Moreover, in this case the operator $\mathcal{S}^{id}_{[\![P]\!]}$ is the identity on $\uparrow(\mathbb{D}_{[\![P]\!]}(id))$. Hence, we obtain the secret kernel by computing the irrelevant elements, which are

$$Irr_{[\![P]\!]} = \{\ X \,|\, \forall h, l\,.\, X \not\subseteq [\![P]\!](h, l)\ \}$$

This is the set $\{\ X \,|\, \forall n \in \mathbb{N}\,.\, 2n^2 \notin X\ \}$. It is immediate to verify that this set is equal to the set $\curlyvee\left(\{\!\{\ \{m\} \,\middle|\, m \in \mathbb{Z} \setminus \bigcup \{\!\{\ 2n^2 \,|\, n \in \mathbb{Z}\ \}\!\}\right)$. At this point, we can conclude that the secret kernel of P is the domain

$$\rho = \uparrow\left(\{\!\{\ 2n^2 \,|\, n \in \mathbb{Z}\ \}\!\} \cup \curlyvee\left(\{\!\{ \{m\} | m \in \mathbb{Z} \setminus \bigcup \{\!\{\ 2n^2 \,|\, n \in \mathbb{Z}\ \}\!\}\right)\right)$$

Consider now $\mathcal{K}^{\mathrm{par}}_{P,[id]}(id)$. In order to obtain this domain we have to find the closure ρ_1 such that $[id]l := 2h\ (\rho_1)$. First of all, we compute $\uparrow\left(\mathbb{D}^{\mathrm{par}}_{[\![P]\!]}(id)\right)$, which is the set $\{2\mathbb{Z}\}$. At this point, we have to find the set of irrelevants which is $\{\ X \,|\, \forall n \in \mathbb{N}\,.\, 2n \notin X\ \}$, namely it is the set $\wp(2\mathbb{Z}+1)$. Therefore the domain ρ_1 is the closure with fixpoints

$$\rho_1 = \uparrow(\{2\mathbb{Z}\}) \cup \wp(2\mathbb{Z}+1)$$

Clearly $[\![P]\!]^{\langle 2 \rangle} = [\![P]\!]$, therefore $\mathcal{K}^{\mathrm{par}}_{P,[id]}(id) = \rho_1 \sqcup \rho$. It is straightforward that $\rho_1 \sqsupset \rho$, hence we have $\mathcal{K}^{\mathrm{par}}_{P,[id]}(id) = \rho_1 \sqsupset \rho$. Finally, consider $\mathcal{K}^{\mathrm{sbs}}_{P,[id]}(id)$. The secret kernel for the first statement is exactly ρ_1, obtained in the previous case. We have to

compute ρ_2 such that $[id]l := l * h\ (\rho_2)$. It is straightforward that $\mathbb{D}_{[\![P]\!]}(id)$ is the set of congruences of the kind $n\mathbb{Z}$, with $n \in \mathbb{Z}$. At this point, the operator $\mathcal{S}^{\varepsilon}_{[\![P]\!]}$ determines the domain $\rho_2 = \{\mathbb{Z}\}$ since each congruence of this type intersects with any other one, breaking the corresponding set Υ. On the other hand, the set of irrelevants is $\{\varnothing\}$. Moreover we have that $\rho_2 \sqsupset \rho_1$, therefore we conclude that $\mathcal{K}^{\text{sbs}}_{P,\,[id]}(id) = \rho_1 \sqcup \rho_2 = \rho_2 \sqsupset \mathcal{K}^{\text{par}}_{P,\,[id]}(id)$.

6.3.4 Canonical attackers

The construction that we have seen above, consider a fixed input observation and finds the most concrete output observation that makes the program secret. Anyway, an attacker is characterized by both the input and output observations, therefore we would like to characterize the most concrete harmless attacker, able to observe the same property both in input and in output. Hence, we want to characterize the most concrete abstract domain ρ such that $[\rho]P\ (\rho)\ [\ (\rho)P\ (\phi \rightsquigarrow \rho)]$, i.e., which represents a possible attacker unable to disclose confidential data by analyzing the same property on input/output data. We call it *canonical* attacker for P, since it allows to compare the relative security of different programs, in the lattice of abstract interpretation. Namely we can say that a program P is more secure than a program Q if the canonical attacker of P is more abstract than the canonical attacker for Q. This problem is, clearly, slightly more complex and requires an iterative solution. The idea is to consider an iterative application of the construction given in the previous sections in order to derive the canonical attacker as the *fixpoint* of this process. The following theorem provides a first simple and immediate characterization of canonical attackers, given in terms of the previous construction.

Theorem 6.21. *Let* $\rho \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$ *and* $\phi \in uco(\wp(\mathbb{V}^{\mathtt{H}}))$.

- $[\rho]P\ (\rho)$ *iff* $\rho = [\rho][\![P]\!]\ (id)$;
- $(\rho)P\ (\phi \rightsquigarrow \rho)$ *iff* $\rho = (\rho)[\![P]\!]\ (\phi \rightsquigarrow id)$.

Proof. By construction an by Theorem 6.17.

Narrow abstract non-interference.

In order to constructively characterize the harmless attackers of a program, we consider standard domain-theoretic arguments. It is worth noting that in the narrow case $[\eta][\![P]\!]\ (id)$, the set of irrelevants does not depend on the input observation η. Therefore, the change of the input property does not have any effect on the irrelevants $Irr^{id,\,id}_{[\![P]\!]}$. Moreover, by construction, the set $\mathbb{D}_{[\![P]\!]}$ monotonically depends upon the input property η. Therefore, we have the following result.

Proposition 6.22. *Let* P *be a program. The function* $\lambda X \in uco(\wp(\mathbb{V}))$. $[X][\![P]\!]\ (id)$ *is monotone on the domain* $\langle uco(\wp(\mathbb{V}^{\mathtt{L}})), \sqsubseteq\rangle$.

Proof. Let us prove that $\lambda X.\ [X][\![P]\!]\ (id)$ is monotone by showing that if $\eta_1 \sqsubseteq \eta$ then we have the inclusion $\left\{\ X \,\middle|\, \mathit{Secr}^{\eta}_{[\![P]\!]}(X)\ \right\} \subseteq \left\{\ X \,\middle|\, \mathit{Secr}^{\eta_1}_{[\![P]\!]}(X)\ \right\}$. Namely, we have to prove that for each X, the implication $\mathit{Secr}^{\eta}(X) \Rightarrow \mathit{Secr}^{\eta_1}(X)$ holds. Suppose that $\mathit{Secr}^{\eta}(X)$, i.e., $\forall l\ .\ (\exists Z \in \Upsilon^{\eta}(l)\ .\ Z \subseteq X\ \Rightarrow\ \forall W \in \Upsilon^{\eta}(l)\ .\ W \subseteq X)$. Note that $\Upsilon^{\eta}_{[\![P]\!]}(l) = \left\{\ [\![P]\!](h,y)^{\mathtt{L}}\ \middle|\ \eta(y) = \eta(l),\ h \in \mathbb{V}^{\mathtt{H}}\ \right\}$ and $\Upsilon^{\eta_1}(l) \subseteq \Upsilon^{\eta}(l)$, since $\eta_1(l) = \eta_1(l')$ implies $\eta(l) = \eta(l')$. Hence, if $\exists Z \in \Upsilon^{\eta_1}(l)$ such that $Z \subseteq X$, then we have $Z \in \Upsilon^{\eta}(l)$. Therefore, $\forall W \in \Upsilon^{\eta_1}(l)$ we have $W \in \Upsilon^{\eta}(l)$, which means that $W \subseteq X$. We can conclude that $\mathit{Secr}^{\eta_1}(X)$ holds.

By Tarski fixpoint theorems, this function has least fixpoint, and it can be obtained as fixpoint of the iterative application of the function itself (see Sect. 2.1.2). Then we can note that the least fixpoint of this function is, by definition, the most concrete canonical attacker. Indeed it is canonical being a fixpoint by Theorem 6.21, it is the most concrete being the *least* fixpoint. Therefore, the idea for deriving the most concrete secure attacker for a program P, in the narrow case, is that of taking $\rho_0 = id$ and then finding, at each step n, the most concrete domain ρ_n that satisfies $[\rho_{n-1}]P\ (\rho_n)$, which is $\mathcal{K}_{P,\,[\rho_{n-1}]}(id)$. By Proposition 6.22, given a program P, we have that $\lambda X.\ (\ [X][\![P]\!]\ (id)) = \lambda X.\ \mathcal{K}_{P,\,[X]}(id)$ is monotone on $\mathit{uco}(\wp(\mathbb{V}^{\mathtt{L}}))$. The most concrete secure attacker for the narrow case is therefore the least fixpoint of $\lambda X.\ \mathcal{K}_{P,\,[X]}(id)$.

Corollary 6.23. *The closure $\mathit{lfp}^{\sqsubseteq}_{id}\lambda X.\ (\ [X][\![P]\!]\ (id))$ is the (unique) most concrete narrow secure attacker for P.*

In the following, we denote $\mathcal{F}_P \stackrel{\text{def}}{=} \mathit{lfp}^{\sqsubseteq}_{id}\lambda X.\ (\ [X][\![P]\!]\ (id))$. The following examples show how we can apply this fixpoint construction of canonical attackers. Moreover, we show that the approximations used before can be also applied to canonical attackers, preserving the same precision relation.

Example 6.24. Consider the fragment given in Example 6.18 together with its denotational semantics. In order to find the canonical attacker we consider

$$\begin{array}{l} \rho_0 \stackrel{\text{def}}{=} id \\ \rho_1 \stackrel{\text{def}}{=} [\rho_0][\![P]\!]\ (id) = [id][\![P]\!]\ (id) \end{array}$$

In Example 6.18 we derived this closure, obtaining the following closure ρ_1, at this point we can find the successive closure ρ_2:

$$\begin{array}{l} \rho_1 = \mathcal{S}^{\varepsilon}_{[\![P]\!]}(\uparrow(\mathbb{D}_{[\![P]\!]}(id))) = \curlyvee\left(\left\{\ n\{2\}^{\mathbb{N}}\ \middle|\ n \in 2\mathbb{N}+1\ \right\} \cup \{\{0\}\}\right) \\ \rho_2 \stackrel{\text{def}}{=} [\rho_1][\![P]\!]\ (id) \end{array}$$

Therefore, we have to find the most concrete closure that makes P secure with input observation ρ_1. Note that $\rho_1 = \mathcal{P}(\rho_1)$. In order to compute this closure we have to compute the elements $[\![P]\!](\mathbb{V}^{\mathtt{H}}, Y)$ for each $Y \in \rho_1$:

$$\begin{array}{l} Y = \{2\}^{\mathbb{N}}\ \Rightarrow [\![P]\!](\mathbb{V}^{\mathtt{H}}, \{2\}^{\mathbb{N}})^{\mathtt{L}} = \{2\}^{\mathbb{N}} \\ Y = 3\{2\}^{\mathbb{N}} \Rightarrow [\![P]\!](\mathbb{V}^{\mathtt{H}}, 3\{2\}^{\mathbb{N}})^{\mathtt{L}} = 3\{2\}^{\mathbb{N}} \\ \ldots \end{array}$$

It is clear that, by using this results we obtain again the closure ρ_1, which means that we reached the fixpoint. Therefore, in this example, we obtain the closure $\mathcal{F}_P = \curlyvee\left(\left\{\ n\{2\}^{\mathbb{N}} \,\middle|\, n \in 2\mathbb{N}+1\ \right\} \cup \{\{0\}\}\right)$.

We now apply bounded iterations to approximate the closure $\mathcal{F}_P$, denoted $\mathcal{F}_P^{\text{par}}$. The single-step semantics of P is:

$$[\![P]\!](h,l) = \begin{cases} (h,l) & \text{if } h = 0 \\ (h-1, l*2) & \text{otherwise} \end{cases}$$

For each value $l \in \mathbb{V}^{\mathtt{L}}$, we have $[\![P]\!](\mathbb{V}^{\mathtt{H}}, l)^{\mathtt{L}} = \{l, 2l\}$. Hence, we can verify that

$$\mathcal{S}^{id}_{[\![P]\!]}(\uparrow(\mathbb{D}_{[\![P]\!]^1}(id))) = \curlyvee\left(\left\{\ n\{2\}^{\mathbb{N}} \,\middle|\, n \in 2\mathbb{N}+1\right\} \cup \{\{0\}\}\right)$$

Consider the second step. For each $l \in \mathbb{V}^{\mathtt{L}}$ we have that $[\![P]\!]^{\langle 2\rangle}(\mathbb{V}^{\mathtt{H}}, l)^{\mathtt{L}} = \{l, 2l, 4l\}$. It is possible to verify that $\curlyvee\left(\left\{\ n\{2\}^{\mathbb{N}} \,\middle|\, n \in 2\mathbb{N}+1\right\} \cup \{\{0\}\}\right)$, which is the closure $\mathcal{F}_P^{\text{par}}$, is again the resulting domain. In this case, the irrelevants are $\{\varnothing\}$, since the range of $[\![P]\!]$ covers the whole domain of values.

In the example above we reached the fixpoint in one step. Let us see an example where an infinite iteration is necessary.

Example 6.25. Consider the program fragment:

$$P \stackrel{\text{def}}{=} l := l^2 + (h \bmod l)$$

with security typing $t = \langle h : \mathtt{H}, l : \mathtt{L}\rangle$, and $\mathbb{V} = \mathbb{N}$. The denotational semantics $[\![P]\!]$ of P is immediate from its definition, and it is

$$[\![P]\!](h,l) = (h, l^2 + (h \bmod l))$$

Let us consider the iteration process introduced above:

$$\begin{array}{l} \rho_0 \stackrel{\text{def}}{=} id \\ \rho_1 \stackrel{\text{def}}{=} [\rho_0][\![P]\!]\ (id) = [id][\![P]\!]\ (id) \end{array}$$

In particular let us derive $\mathbb{D}_{[\![P]\!]}(id)$:

$$\begin{array}{l} l = 1 \Rightarrow [\![P]\!](\mathbb{V}^{\mathtt{H}}, 1)^{\mathtt{L}} = \{1\} \\ l = 2 \Rightarrow [\![P]\!](\mathbb{V}^{\mathtt{H}}, 2)^{\mathtt{L}} = [4,5] \\ l = 3 \Rightarrow [\![P]\!](\mathbb{V}^{\mathtt{H}}, 3)^{\mathtt{L}} = [9,11] \\ l = 4 \Rightarrow [\![P]\!](\mathbb{V}^{\mathtt{H}}, 4)^{\mathtt{L}} = [16,19] \\ \ldots \end{array}$$

Therefore $\mathbb{D}_{[\![P]\!]}(id) = \left\{\ [n^2, n^2+n-1] \,\middle|\, n \in \mathbb{N}\ \right\}$. These intervals are all disjoint, therefore $\mathcal{S}^{id}_{[\![P]\!]}$ on this set behaves like the identity map. This means that the resulting closure ρ_1 contains $\curlyvee(\mathbb{D}_{[\![P]\!]}(id))$ together with all the elements that are

not contained in any of these intervals, i.e., the irrelevants. Therefore we can continue the process

$$\begin{aligned}\rho_1 = &\ \mathbb{D}_{[\![P]\!]}(id) = \{\ [n^2, n^2+n-1] \mid n \in \mathbb{N}\ \} \cup Irr_{[\![P]\!]}\\ \rho_2 \stackrel{\text{def}}{=} &\ [\rho_1][\![P]\!]\ (id)\end{aligned}$$

In order to find ρ_2, as before, we have to derive $\mathbb{D}_{[\![P]\!]}(id)$:

$$\begin{array}{l}
l = 1 \Rightarrow \rho_1(1) = 1\ \wedge\ [\![P]\!](\mathbb{V}^{\mathtt{H}}, 1)^{\mathtt{L}} = \{1\}\\
l = 2 \Rightarrow \rho_1(2) = 2\ \wedge\ [\![P]\!](\mathbb{V}^{\mathtt{H}}, 2)^{\mathtt{L}} = [4,5]\\
l = 3 \Rightarrow \rho_1(3) = 3\ \wedge\ [\![P]\!](\mathbb{V}^{\mathtt{H}}, 3)^{\mathtt{L}} = [9,11]\\
l = 4 \Rightarrow \rho_1(4) = [4,5]\ \wedge\ [\![P]\!](\mathbb{V}^{\mathtt{H}}, [4,5])^{\mathtt{L}} = [16,19] \cup [25,29]\\
\ldots\\
l = 9 \Rightarrow \rho_1(9) = [9,11]\ \wedge\ [\![P]\!](\mathbb{V}^{\mathtt{H}}, [9,11])^{\mathtt{L}} = [81,89] \cup [100,109] \cup [121,131]\\
\ldots
\end{array}$$

We may continue in this way, until the fixpoint characterizing the canonical harmless attacker. Note that $\mathcal{S}^{id}_{[\![P]\!]}$ on these domains is always the identity, since the generated intervals are always disjoint.

Abstract non-interference.

At this point, we would like to derive the same construction for abstract non-interference. Unfortunately, we cannot obtain it in the same way since the function that we should iterate in the abstract case is not monotone. In particular, there are two facts that avoid monotonicity. First of all, the set of irrelevants *grows* when we abstract η; second the set of relevants is *not comparable* with the set of relevants obtained by abstracting η. Therefore, in general, $\lambda X.\ (X)[\![P]\!]\ (\phi \rightsquigarrow\!\!\| id)$ is not monotone on $\langle uco(\wp(\mathbb{V}^{\mathtt{L}})), \sqsubseteq\rangle$ as we can see in the following example.

Example 6.26. Let us consider the program fragment P in Example 6.19. Let us consider two closures $\eta \sqsubseteq \beta$ such that $\eta(\mathbb{Z}) = \{\mathbb{Z}, [2,4], [5,8], \{5\}, \varnothing\}$ and $\beta(\mathbb{Z}) = \{\mathbb{Z}, [2,4], [5,8], \varnothing\}$. Consider η, in Example 6.19 we obtained that $\mathbb{D}_{[\![P]\!]}(\eta) = \{[5,7], [2,6], [5,10], \mathbb{Z}\}$. We can note that $\neg Secr^{\eta,id}_{[\![P]\!]}([2,6])$ since $[2,6]$ contains some elements of $\Upsilon^{\eta,\,id}_{[\![P]\!]}(5)$ (see Example 6.19), but not all of them. This means that surely $[2,6]$ is not included in the final domain, as seen in Example 6.19. Consider now β, then we have

$$\begin{aligned}Y = [2,4] &\Rightarrow [\![P]\!](\mathbb{Z}, [2,4])^{\mathtt{L}} = [2,6]\\ Y = [5,8] &\Rightarrow [\![P]\!](\mathbb{Z}, [5,8])^{\mathtt{L}} = [5,10]\\ Y = \mathbb{Z} &\Rightarrow [\![P]\!](\mathbb{Z}, \mathbb{Z})^{\mathtt{L}} = \mathbb{Z}\end{aligned}$$

Therefore $\mathbb{D}_{[\![P]\!]}(\eta) = \{[2,6], [5,10], \mathbb{Z}\}$. While

$$\begin{aligned}\Upsilon^{\eta,\,id}_{[\![P]\!]}([2,4]) &= \{[2,4], [3,5], [4,6]\}\\ \Upsilon^{\eta,\,id}_{[\![P]\!]}([5,8]) &= \{[5,8], [6,9], [7,10]\}\\ \forall y.\ \beta(y) \notin \{[2,4], [5,8]\}.\ \Upsilon^{\eta,\,id}_{[\![P]\!]}(y) &= \{\mathbb{Z}\}\end{aligned}$$

In this case $Secr^{\eta,id}_{[\![P]\!]}([2,6])$, which implies that $[2,6]$ is now left in the domain.

This example shows that in abstract non-interference, in general, by further abstracting the input observation we don't necessarily further abstract the output observation. Hence, in the abstract non-interference case, the fact that $\lambda X.\ (X)[\![P]\!]\ (\phi \rightsquigarrow\!\!\upharpoonright id)$ is not monotone, avoids us to characterize the canonical harmless attacker by means of Tarski fixpoint theorems. In order to constructively characterize an harmless attacker as the limit of a possible transfinite upper iteration sequence of $\lambda X.\ (X)[\![P]\!]\ (\phi \rightsquigarrow\!\!\upharpoonright id)$, we force this function to be extensive on $\langle uco(\wp(\mathbb{V}^{\mathtt{L}})), \sqsubseteq\rangle$, i.e., we consider the function:

$$\lambda X.\ (X)[\![P]\!]\ (\phi \rightsquigarrow\!\!\upharpoonright id) \sqcup X$$

By a well known result in lattice theory we have that any extensive function has a fixpoint [37]. The following result proves that any fixpoint of $\lambda X.\ (X)[\![P]\!]\ (\phi \rightsquigarrow\!\!\upharpoonright id) \sqcup X$ is a harmless attacker for P.

Theorem 6.27. *Let $\rho \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$ and $\phi \in uco(\wp(\mathbb{V}^{\mathtt{H}}))$. Then*

$$\rho = (\rho)[\![P]\!]\ (\phi \rightsquigarrow\!\!\upharpoonright id) \sqcup \rho \ \Rightarrow\ (\rho)P\ (\phi \rightsquigarrow\!\!\upharpoonright \rho).$$

Proof. Consider $\rho = (\rho)[\![P]\!]\ (\phi \rightsquigarrow\!\!\upharpoonright id) \sqcup \rho$, this means that, if $\rho' = (\rho)[\![P]\!]\ (\phi \rightsquigarrow\!\!\upharpoonright id)$, then $\rho' \sqsubseteq \rho$. By construction we have that $(\rho)P\ (\phi \rightsquigarrow\!\!\upharpoonright \rho')$ holds, and by Proposition 6.9(5) we conclude that $(\rho)P\ (\phi \rightsquigarrow\!\!\upharpoonright \rho)$, being $\rho' \sqsubseteq \rho$.

Unfortunately, in this case, we cannot guarantee that we obtain the "most concrete" harmless attacker, since the limit of an extensive function may not be the "least" fixpoint of the function, we only know that surely it is a fixpoint.

6.4 Abstract declassification

Declassifying information means downgrading the sensitivity of data in order to accommodate with (intentional) information leakage. In this section, we show how abstract non-interference can be used also to characterize which is the maximal amount of information about private data that surely flows and that we have to declassify in order to guarantee secrecy. Namely, we want to characterize a property that contains all the possible variations of private inputs that generate insecure information flows. More formally speaking, let $P \in$ Imp be a program and $\eta, \rho \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$ be abstract domains, we want to derive the closures ϕ such that

$$\forall h_1, h_2 \in \mathbb{V}^{\mathtt{H}}.\ (\phi(h_1) = \phi(h_2)\ \Rightarrow\ \forall l \in \mathbb{V}^{\mathtt{L}}.\ \rho([\![P]\!](h_1, \eta(l)^{\mathtt{L}}) = \rho([\![P]\!](h_2, \eta(l)^{\mathtt{L}})) \tag{6.1}$$

In other words, if we have two private values that show a flow of information, i.e., whose variation interferes in the public output, then they are distinguished by ϕ. Any property ϕ satisfying the relation 6.1 will be called *private observable* of the program P.

Clearly, a private observable, could distinguish private data too much. Namely, a property that satisfies 6.1, could distinguish also private values whose variation does not cause any insecure information flow. For this reason, we are also interested in characterizing the abstractions $\phi \in uco(\wp(\mathbb{V}^{\mathtt{H}}))$ such that any pair of values which can be distinguished by ϕ violates $\langle \eta, \phi, \rho \rangle$-*Secrecy*.

Definition 6.28. *A closure* $\phi \in \mathcal{P}(uco(\wp(\mathbb{V}^{\mathtt{H}})))$ *is called* flow-irredundant *with respect to a pair of input/output abstractions* $\langle \eta, \rho \rangle$ *if*

$$\forall h_1, h_2 \in \mathbb{V}^{\mathtt{H}} \,.\, \phi(h_1) \neq \phi(h_2). \exists l \in \mathbb{V}^{\mathtt{L}} \,.\, \rho([\![P]\!](\phi(h_1), \eta(l))^{\mathtt{L}}) \neq \rho([\![P]\!](\phi(h_2), \eta(l))^{\mathtt{L}})$$

The most concrete flow-irredundant domain $\phi \in \mathcal{P}(uco(\wp(\mathbb{V}^{\mathtt{H}})))$, when it exists relatively to $\langle \eta, \rho \rangle$, defines the *maximal abstract interference* of P, denoted $(\eta)P\ (\phi \Rightarrow \rho)$.
It is clear that a property that is both the most abstract private observable, and the maximal abstract interference, provides precisely the maximal amount of information concerning $\mathtt{H}$-values that flows in P when the attacker is modeled by the pair of input/output abstractions $\langle \eta, \rho \rangle$. Indeed, if ϕ is the most abstract private observable, then it identifies all the insecure information flow situations, while if ϕ is the most concrete non-redundant property that flows, then it identifies exactly those situations where insecure information flows happen.

The idea for deriving the maximal amount of private information that flows, is that of building the most abstract private observable collecting together only those values that cannot generate insecure information flows. We will show that, in this way, we obtain also the maximal abstract interference. Consider the following partition of values:

$$\Pi_{\mathtt{P}}(\eta, \rho) \stackrel{\text{def}}{=} \left\{ \left\langle \left\{ h \in \mathbb{V}^{\mathtt{H}} \,\middle|\, \rho([\![P]\!](\langle h, \eta(l)\rangle)^{\mathtt{L}}) = A \right\}, \eta(l) \right\rangle \,\middle|\, l \in \mathbb{V}^{\mathtt{L}},\ A \in \rho \right\}$$

This is the set of all the pairs $\langle H, L \rangle \in \wp(\mathbb{V}^{\mathtt{H}}) \times \wp(\mathbb{V}^{\mathtt{L}})$, such that, whenever $\eta(l) = L$, then for any $h_1, h_2 \in H$, no information flows are revealed. We use this set for deriving a partition of private data that guarantees secrecy.
For each $L \in \eta$, we define the partition $\Pi_{\mathtt{P}}(\eta, \rho)_{|L} \stackrel{\text{def}}{=} \left\{ H \,\middle|\, \langle H, L \rangle \in \Pi_{\mathtt{P}}(\eta, \rho) \right\}$, of private data. The partition obtained corresponds to the most abstract property, containing all the possible flows, whenever the property of public input is L.

At this point, we use these partitions, in order to find the most abstract private observable ϕ[1], representing the maximal abstract interference:

$$\phi \stackrel{\text{def}}{=} \mathcal{P}\left(\prod_{L \in \eta} \mathcal{M}(\Pi_{\mathtt{P}}(\eta, \rho)_{|L}) \right)$$

Then, the following result holds.

[1] Note that $\mathcal{P}(\phi)$ is the most concrete domain that induces the same partition as ϕ, and because in abstract non-interference we abstract single values, it is clear that we can consider $\phi \in \mathcal{P}(uco(\wp(\mathbb{V}^{\mathtt{H}})))$.

Theorem 6.29. *If* $\not\models \ (\eta)P \ (id \rightsquigarrow \rho)$ *and* ϕ *is defined as shown above, then it is the most abstract private observable and* $(\eta)P \ (\phi \Rightarrow \rho)$*, i.e., it is also the maximal abstract interference.*

Proof. First of all, we have to prove that ϕ, as defined above, is a private observable and it is the most abstract. Consider $h_1, h_2 \in \mathbb{V}^{\mathtt{H}}$, and suppose that $\phi(h_1) = \phi(h_2)$. By definition of ϕ, this means that for each $L \in \eta$, we have that h_1 and h_2 are in the same equivalence class induced by $\Pi_{\mathtt{P}}(\eta, \rho)_{|L}$. But, by definition of $\Pi_{\mathtt{P}}(\eta, \rho)_{|L}$, this means that, for each $L \in \eta$, $\rho([\![P]\!](h_1, L)^{\mathtt{L}}) = \rho([\![P]\!](h_2, L)^{\mathtt{L}})$, and this con be rewritten as $\forall l \in \mathbb{V}^{\mathtt{L}} \,.\, \rho([\![P]\!](h_1, \eta(l))^{\mathtt{L}}) = \rho([\![P]\!](h_2, \eta(l))^{\mathtt{L}})$. So we proved that ϕ is a private observable. Let us prove that it is the most abstract. Suppose that there exists a private observable $\phi' \sqsupset \phi$. This implies that, in particular, there exist two private values h_1 and h_2 such that $\phi'(h_1) = \phi'(h_2)$ and $\phi(h_1) \neq \phi(h_2)$. By definition of ϕ, this means that there exists $L \in \eta$ such that $\Pi_{\mathtt{P}}(\eta, \rho)_{|L}$ distinguishes the two private values, but if $\Pi_{\mathtt{P}}(\eta, \rho)_{|L}$ distinguishes h_1 and h_2 then, by construction, $\rho([\![P]\!](h_1, L)^{\mathtt{L}}) \neq \rho([\![P]\!](h_2, L)^{\mathtt{L}})$. Namely, $\exists l \in \mathbb{V}^{\mathtt{L}} \,.\, \rho([\![P]\!](h_1, \eta(l))^{\mathtt{L}}) \neq \rho([\![P]\!](h_2, \eta(l))^{\mathtt{L}})$, which is a contradiction since ϕ' was a private observable.

Now, we have to prove that ϕ is flow-irredundant and that is the most concrete one, as regards the closures $\eta, \rho \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$.

First, we have to prove that ϕ is flow-irredundant, namely $\forall x_1, x_2 \in \mathbb{V}^{\mathtt{H}}$ such that $\phi(x_1) \neq \phi(x_2)$, there exists $y \in \mathbb{V}^{\mathtt{L}}$ such that $\rho([\![P]\!](\phi(x_1), \eta(y))^{\mathtt{L}}) \neq \rho([\![P]\!](\phi(x_2), \eta(y))^{\mathtt{L}})$. Let $[x_1]_\phi$ and $[x_2]_\phi$ be the corresponding equivalence classes of the partition induced by ϕ, i.e., $[x_1]_\phi = \phi(x_1)$ and $[x_2]_\phi = \phi(x_2)$. By construction of these equivalence classes for each $[x]_\phi$ and for each $y \in [x]_\phi$ (i.e., $y \in \phi(x_1)$) we have $\rho([\![P]\!](y, \eta(y))^{\mathtt{L}}) = \rho([\![P]\!](x, \eta(y))^{\mathtt{L}})$, which implies that for each $y \in [x]_\phi$, $\rho([\![P]\!](y, \eta(y))^{\mathtt{L}}) = \rho([\![P]\!]([x]_\phi, \eta(y))^{\mathtt{L}})$ (*). At this point, by definition of partition, if $z \in [x_1]_\phi$ then $z \notin [x_2]_\phi$ and viceversa. This means that for each $z \in [x_1]_\phi$ there exists $y \in \mathbb{V}^{\mathtt{L}}$ such that $\rho([\![P]\!](z, \eta(y))^{\mathtt{L}}) \neq \rho([\![P]\!]([x_2]_\phi, \eta(y))^{\mathtt{L}})$, since $\rho([\![P]\!](x_2, \eta(y))^{\mathtt{L}}) = \rho([\![P]\!]([x_2]_\phi, \eta(y))^{\mathtt{L}})$. But we have also that for each $z \in [x_1]_\phi$, $\rho([\![P]\!](z, \eta(y))^{\mathtt{L}}) = \rho([\![P]\!]([x_1]_\phi, \eta(y))^{\mathtt{L}})$, therefore the property is flow-irredundant, namely $\rho([\![P]\!]([x_1]_\phi, \eta(y))^{\mathtt{L}}) \neq \rho([\![P]\!]([x_2]_\phi, \eta(y))^{\mathtt{L}})$.

Now, we have to prove that ϕ is the most concrete flow-irredundant closure, as regards the two observables properties $\eta, \rho \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$. Suppose that there exists a closure φ more concrete than ϕ or non comparable with ϕ, which is flow-irredundant. Suppose they are incomparable, let us prove that the hypotheses above implies that $\phi' \stackrel{\text{def}}{=} \phi \sqcap \varphi$ is flow-irredundant. Being ϕ flow-irredundant we have that $\forall x_1, x_2 . \phi(x_1) \neq \phi(x_2)$ there exists y such that $\rho([\![P]\!](\phi(x_1), \eta(y))^{\mathtt{L}}) \neq \rho([\![P]\!](\phi(x_2), \eta(y))^{\mathtt{L}})$. Being $\phi' \sqsubseteq \phi$ we have also that $\phi(x_1) \neq \phi(x_2)$ implies $\phi'(x_1) \neq \phi'(x_2)$, let us prove that $\rho([\![P]\!](\phi'(x_1), \eta(y))^{\mathtt{L}}) \neq \rho([\![P]\!](\phi'(x_2), \eta(y))^{\mathtt{L}})$. By definition of reduced product we have that $\phi'(x) = \phi(x) \cap \varphi(x)$. Hence

$$\begin{aligned}
\rho([\![P]\!](\phi'(x_1), \eta(y))^{\mathtt{L}}) &= \rho([\![P]\!](\phi(x_1) \cap \varphi(x_1), \eta(y))^{\mathtt{L}}) \\
&= \rho\left(\bigcup_{h \in \phi(x_1) \cap \varphi(x_1)} [\![P]\!](h, \eta(y))^{\mathtt{L}}\right) \\
&= \rho([\![P]\!](\phi(x_1), \eta(y))^{\mathtt{L}}) \text{ (for the condition (*), being } h \in \phi(x_1)) \\
&\neq \rho([\![P]\!](\phi(x_2), \eta(y))^{\mathtt{L}}) = \rho\left(\bigcup_{h \in \phi(x_2) \cap \varphi(x_2)} [\![P]\!](h, \eta(y))^{\mathtt{L}}\right) \\
&= \rho([\![P]\!](\phi(x_2) \cap \varphi(x_2), \eta(y))^{\mathtt{L}}) = \rho([\![P]\!](\phi'(x_2), \eta(y))^{\mathtt{L}})
\end{aligned}$$

This means that we can consider only $\varphi \sqsubseteq \phi$. This hypothesis implies that the partition induced by φ is a refinement than the one induced by ϕ. Let $[x]_\varphi$ the new equivalence classes and consider two values $x_1, x_2 \in \mathbb{V}^{\mathtt{H}}$ such that $[x_1]_\varphi \neq [x_2]_\varphi$ and such that $x_1 \in [x_2]_\phi$, two such classes in φ exist since this closure is more concrete than ϕ. But if $x_1 \in [x_2]_\phi$, then by construction we have $\forall y \in \mathbb{V}^{\mathtt{L}}, \forall x' \in [x_1]_\phi, x'' \in [x_2]_\phi . \rho([\![P]\!](x', \eta(y))^{\mathtt{L}}) = \rho([\![P]\!](x'', \eta(y))^{\mathtt{L}})$ and therefore, since $\forall x. [x]_\varphi \subseteq [x]_\phi$ we have $\rho([\![P]\!]([x_1]_\varphi, \eta(y))^{\mathtt{L}}) = \rho([\![P]\!]([x_2]_\varphi, \eta(y))^{\mathtt{L}})$, which is a contradiction, since it would mean that φ is not flow-irredundant.

The following example shows how we can use the transformer for characterizing the most abstract private observable.

Example 6.30. Consider the program fragment:

$$P = \quad l := l * h^2;$$

We can compute

$$\Pi_{\mathtt{P}}\,(id, Par) = \left\{\ \langle \mathbb{Z}, l\rangle \,\middle|\, l \in 2\mathbb{Z}\ \right\} \cup \left\{\ \langle 2\mathbb{Z}, l\rangle \,\middle|\, l \in 2\mathbb{Z}+1\ \right\} \cup \left\{\ \langle 2\mathbb{Z}+1, l\rangle \,\middle|\, l \in 2\mathbb{Z}+1\ \right\}$$

Therefore by using the notation above we have that

$$\begin{aligned}
l \in 2\mathbb{Z} \quad &\Rightarrow \pi_l = \mathbb{Z} \\
l \in 2\mathbb{Z}+1 \Rightarrow \pi_l &= \{2\mathbb{Z}, 2\mathbb{Z}+1\}
\end{aligned}$$

This means that $\mathcal{P}(\sqcap_{l \in \mathbb{Z}} \mathcal{M}(\pi_l)) = Par$. In other words, we have that by looking at the low variables the only information that leaks about the high variables is its parity.

Note that, this construction is related with declassification, since it says that all the properties that distinguish values that are distinguished by the most abstract private observable generate interference. Indeed, if we want to guarantee non-interference by declassifying private properties, we have at least to declassify all the properties that induce a partition which is as precise as the one induced by the most abstract private observable. Note that, we are considering only the partition induced by the private observable, since non-interference is defined by applying the property on single values.

6.5 Enriching the semantics

In this section, we consider two slight extensions of the notion of abstract non-interference introduced in the previous section. In particular, these extensions, are obtained by enriching the semantics. In the first case, we consider the trace semantics instead of the denotational one. In this way we, clearly, enrich the observational capability of the attacker, which is now able to observe the entire memory of the system during the whole execution of the program. We will see, in the following, that this extension is necessary in order to make abstract non-interference time-insensitive. In the second extension, we add non-determinism, by considering the non-deterministic denotational semantics defined in [27] as abstraction of the maximal trace semantics of non-deterministic systems.

6.5.1 Abstract non-interference on traces

In this section, we describe how it is possible to strengthen the given notions of abstract non-interference, based on denotational (I/O) semantics, by defining them in terms of trace semantics. This means that at *each step* of computation we require that, what an attacker may observe does not depend on private input. Since abstract non-interference is based on the distinction between input and output, the simpler way to extend the two notions is to consider, as output, the results of all the partial computations, and as input only the initial values (namely the initial state). With this assumption we can consider again only two closures, η for the public input, and ρ for the public output[2].
Consider first narrow abstract non-interference. We can formulate the notion of non-interference by saying that all the execution traces of a program starting from states with different confidential input and the same η property of public input, have to provide the same public observations. In particular, it consists in guaranteeing that starting from a state with the low property η, then all the possible observations of the states during the computation have the same low property ρ. Therefore, the new notion of narrow non-interference consists simply in abstracting each state of the computational traces. In the following, we will denote the trace semantics of P as $\langle\!|P|\!\rangle$, in order to distinguish it from the denotational semantics $[\![P]\!]$. Let us introduce this notion through an example. Consider the standard semantics where states are simply tuples of values for the variables, and consider for example the concrete trace (each state is $\langle h, l\rangle$):

$$\langle 3,1\rangle \longrightarrow \langle 2,2\rangle \longrightarrow \langle 1,3\rangle \longrightarrow \langle 0,4\rangle \longrightarrow \langle 0,4\rangle$$

Now, suppose to observe the parity of public data, i.e., *Par*, both in input and in output, then intuitively the abstraction, i.e., the observation of this trace through the property *Par* is:

[2] Note that in the most general case we could consider a family of "output" observations.

$$\langle 3, 2\mathbb{Z}+1\rangle \longrightarrow \langle 2, 2\mathbb{Z}\rangle \longrightarrow \langle 1, 2\mathbb{Z}+1\rangle \longrightarrow \langle 0, 2\mathbb{Z}\rangle \longrightarrow \langle 0, 2\mathbb{Z}\rangle$$

Namely, we define the abstraction of a computational trace of P, i.e., $\sigma \in \langlebar P \ranglebar$, relatively to the property ρ, as done in [118], with the only difference that we abstract, explicitly, only the low values:

$$\forall \sigma \in \langlebar P \ranglebar . \forall i \leq |\sigma| \,.\, \sigma_i^\rho = \langle \sigma^{\mathtt{H}} \,, \rho(\sigma_i^{\mathtt{L}}) \rangle$$

At this point, consider only the angelic trace semantics of P, then we can define the semantics of the abstract traces. Consider $\rho \in uco(\mathbb{V}^{\mathtt{L}})$ and $X \in \wp(\Sigma^+)$:

$$\begin{array}{l} \langlebar P \ranglebar^\eta_\rho = \alpha^\eta_\rho(\langlebar P \ranglebar) \\ \alpha^\eta_\rho(X) = \left\{ \; s^\eta \delta^\rho \;\middle|\; s \in \Sigma,\; s\delta \in X \; \right\} \end{array}$$

This is clearly an abstraction since it is additive by construction. Moreover, it is immediate to determine the corresponding concretization as the set of all the possible traces that have the same abstract trace. We can define narrow non-interference for traces simply by changing the semantics:

A program P is *secure* if
$\forall h_1, h_2 \in \mathbb{V}^{\mathtt{H}}, \forall l_1, l_2 \in \mathbb{V}^{\mathtt{L}} \;.\; \langlebar P \ranglebar^\eta_\rho(h_1, l_1)^{\mathtt{L}} = \langlebar P \ranglebar^\eta_\rho(h_2, l_2)^{\mathtt{L}}$.

where, as usual, $\eta, \rho \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$. The interesting aspect of this extension is that we can apply the transformers defined on abstract non-interference (seen in the previous section) simply by considering the approximation based on bounded iteration (see page 133). Bounded iteration, in fact, proves I/O non-interference by requiring a stronger condition, i.e., it requires that all the partial computations provide the same public output. This is clearly narrow non-interference on traces, since we compare abstract *observations* of concrete *computations*.

It is worth noting that, in order to define narrow non-interference we keep the concrete semantics and we change its observation. If, instead, we want to define abstract non-interference on traces, then we have to change also the concrete semantics by considering as initial state the set of all the states with the same public input property. In this case we have to consider a lift of the transition relation to sets: Consider the transition system $\langle \Sigma, \longrightarrow \rangle$, we define the lift $\twoheadrightarrow \subseteq \wp(\Sigma) \times \wp(\Sigma)$ as follows

$$X \twoheadrightarrow \left\{ \; y \in \Sigma \;\middle|\; \exists x \in X.\, x \longrightarrow y \; \right\}$$

Consider now the lifted transition system $\langle \wp(\Sigma), \twoheadrightarrow \rangle$, and consider the trace semantics obtained from this transition system. Let us denote also this semantics as $\langlebar P \ranglebar$, since it is clear from the input (depending on the fact that it is a state or a set of states) which semantics we have to consider. Again let us consider the input and output abstractions and the abstract trace semantics $\langlebar P \ranglebar^\eta_\rho$, on the lifted transition system, then we can define abstract non-interference on traces as follows:

A program P is *secure* if
$\forall h_1, h_2 \in \mathbb{V}^{\mathtt{H}}, \forall l \in \mathbb{V}^{\mathtt{L}} \ . \ \langle\!|P|\!\rangle^{\eta}_{\rho}(\phi(h_1), \eta(l))^{\mathtt{L}} = \langle\!|P|\!\rangle^{\eta}_{\rho}(\phi(h_2), \eta(l))^{\mathtt{L}}$.

where, as usual, $\phi \in uco(\wp(\mathbb{V}^{\mathtt{H}}))$ and $\eta, \rho \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$.

6.5.2 Abstract non-interference for non-deterministic languages

In the following, we consider the simple imperative language with non deterministic choice, ND-IMP, introduced in Sect. 4.2.1. As in the deterministic case, the operational semantics naturally induces a transition relation on a set of states Σ, denoted $\longrightarrow$, specifying the relation between a state and *the set* of its possible successors. Consider the Cousot's construction (see Sect. 4.1.2), where Σ^+ and $\Sigma^{\omega} \stackrel{\text{def}}{=} \mathbb{N} \to \Sigma$ denote respectively the set of finite nonempty and infinite sequences of symbols in Σ as in deterministic case. In the following $[\![P]\!]$ denotes the input/output relation for the program P also in the non-deterministic case, therefore $[\![P]\!](s)$ denotes the set of all the states reachable by executing P starting from the state s.

In this context, consider the *possibilistic* non-interference defined in [108] for non deterministic programs: A program is secure if given two states s_1 and s_2 such that $s_1^{\mathtt{L}} = s_2^{\mathtt{L}}$, then for each computation σ with $\sigma_{\vdash}^{\mathtt{L}} = s_1^{\mathtt{L}}$ there exists a computation δ with $\delta_{\vdash}^{\mathtt{L}} = s_2^{\mathtt{L}}$, such that $\sigma_{\dashv}^{\mathtt{L}} = \delta_{\dashv}^{\mathtt{L}}$ (see Sect 5.3.2). This notion can be formulated as in Eq. 5.2 with the only semantic difference that now $[\![P]\!](s)^{\mathtt{L}}$ are sets of values instead of a single value. Unfortunately, the generalization is not so straightforward, indeed if we don't consider additive closure for the output observation, then the notion of abstract non-interference as given above, is not precise. In fact, missing additivity means that the property of a set is not the union of the properties of its elements. In the context of non-interference, this means that the collection of all the observations of the single computations, does not corresponds to the observation of the set of all the possible results. Indeed, we recall that possibilistic non-interference is based on the assumption that the attacker can observe and collect all the possible system behaviours. Therefore, if it is able to observe the property ρ of the output, then the natural non-deterministic extension of abstract non-interference would say that the attacker can collect the set of all the ρ observations of the possible system behaviours, which is in general different from the ρ property of the set of all the possible system behaviours. Therefore, in order to define abstract non-interference for non-deterministic system simply by considering the non-deterministic denotational semantics as defined in [27] we can only consider additive properties for the output observation. Therefore, when ρ is *additive* we define abstract non-interference exactly as we have done for deterministic systems:

A program P is *secure* if
$\forall h_1, h_2 \in \mathbb{V}^{\mathtt{H}}, \forall l \in \mathbb{V}^{\mathtt{L}} \ . \ \rho([\![P]\!](\phi(h_1), \eta(l))^{\mathtt{L}}) = \rho([\![P]\!](\phi(h_2), \eta(l))^{\mathtt{L}})$

Anyway, we can overcome the limitation of requiring additive properties for the public output, by considering the semantics obtained as denotational abstraction (see Table 4.1) of the trace semantics abstracted by η and ρ, as seen in the previous section. Therefore, we consider the abstract semantics $\langle\!|P|\!\rangle^{\eta}_{\rho}$ ($\langle\!|P|\!\rangle$ is the standard trace semantics or the lifted one depending on the fact that we are defining narrow or abstract non-interference) and we take its denotational abstraction:

$$[\![P]\!]^{\eta}_{\rho} \stackrel{\text{def}}{=} \alpha^{\mathcal{D}}(\langle\!|P|\!\rangle^{\eta}_{\rho})$$

Then we can define abstract non-interference for non-deterministic systems simply by considering this semantics:

A non-deterministic program P is *secure* for narrow non-interference if
$\forall h_1, h_2 \in \mathbb{V}^{\mathtt{H}}, \forall l_1, l_2 \in \mathbb{V}^{\mathtt{L}} \;.\; \eta(l_1) = \eta(l_2) \;\Rightarrow\; [\![P]\!]^{\eta}_{\rho}(h_1, l_1)^{\mathtt{L}} = [\![P]\!]^{\eta}_{\rho}(h_2, l_2)^{\mathtt{L}}$.

A non-deterministic program P is *secure* for abstract non-interference if
$\forall h_1, h_2 \in \mathbb{V}^{\mathtt{H}}, \forall l \in \mathbb{V}^{\mathtt{L}} \;.\; [\![P]\!]^{\eta}_{\rho}(\phi(h_1), \eta(l))^{\mathtt{L}} = [\![P]\!]^{\eta}_{\rho}(\phi(h_2), \eta(l))^{\mathtt{L}}$.

In order to understand the difference between the two definitions of abstract non-interference given for non-deterministic systems, let us consider the following example.

Example 6.31. Let us consider the program fragment

$$P \stackrel{\text{def}}{=} \textbf{while } h \textbf{ do } l := l + 1; x := x - 1 \textbf{ endw } \square\ l := 0;$$

with security typing $t = \langle h : \mathtt{H}, l : \mathtt{L} \rangle$, and $\mathbb{V} = \mathbb{N}$. The concrete trace semantics of this system is the set of traces

$$\begin{array}{l}\langle\!|P|\!\rangle = \left\{\ \langle 0, l\rangle \longrightarrow \langle 0, l\rangle \,\middle|\, l \in \mathbb{N}\ \right\} \cup \left\{\ \langle h, l\rangle \longrightarrow \langle h, 0\rangle \,\middle|\, h, l \in \mathbb{N}\ \right\} \cup \\ \qquad \left\{\ \langle h, l\rangle \longrightarrow \langle h-1, l+1\rangle \longrightarrow \ldots \longrightarrow \langle 0, l+h\rangle \,\middle|\, h, l \in \mathbb{N}, h \neq 0\ \right\}\end{array}$$

Consider for example $h = 3$ and $l = 1$, then we have

$$[\![P]\!](3,1) = \alpha^{\mathcal{D}}(\langle\!|P|\!\rangle)(3,1) = \{\langle 0, 4\rangle, \langle 3, 0\rangle\} \text{ hence } [\![P]\!](3,1)^{\mathtt{L}} = \{4, 0\}$$

On the other hand, consider the property $Par \stackrel{\text{def}}{=} \{\top, 2\mathbb{N} \smallsetminus \{0\}, 2\mathbb{N}+1, \{0\}, \varnothing\}$, clearly not additive, then the abstract semantics $\langle\!|P|\!\rangle^{Par}_{Par}$ is

$$\begin{array}{l}\langle\!|P|\!\rangle^{Par}_{Par} = \left\{\ \langle 0, Par(l)\rangle \longrightarrow \langle 0, Par(l)\rangle \,\middle|\, l \in \mathbb{N}\ \right\} \cup \\ \qquad \left\{\ \langle h, Par(l)\rangle \longrightarrow \langle h, \{0\}\rangle \,\middle|\, h, l \in \mathbb{N}\ \right\} \cup \\ \qquad \left\{ \begin{array}{l} \langle h, Par(l)\rangle \longrightarrow \langle h-1, Par(l+1)\rangle \longrightarrow \cdots \\ \longrightarrow \langle 0, Par(l) \oplus Par(h)\rangle \end{array} \,\middle|\, h, l \in \mathbb{N}, h \neq 0 \right\}\end{array}$$

At this point we consider the denotational abstraction of this trace semantics obtaining:

$$[\![P]\!]^{Par}_{Par} = \alpha^{\mathcal{D}}(\langle\!|P|\!\rangle^{Par}_{Par})(3,1)^{\mathtt{L}} = \{2\mathbb{N} \smallsetminus \{0\}, \{0\}\} \neq Par(\{4,0\}) = \top$$

In the example above, it is worth noting that, the two notions of abstract non-interference, provided for non-deterministic systems, are different, since they could classify as secure different programs. Moreover, it is worth noting that the latter is more precise, i.e., stronger.

6.6 Related works

As we have noted in the previous chapter, there exist different works whose task is to weaken non-interference, in order to make it less restrictive. In particular, we have found two works, that more than others, are strictly related with the idea of abstract non-interference. In fact both, the PER model [106] and robust declassification [118] model the observational capability of the attacker by using *equivalence relations*, that, as shown in Sect. 2.2.3, are particular a kind of upper closure operators, used in abstract non-interference.

6.6.1 Abstract non-interference vs PER model

In Sect. 5.1.3, we described the semantic approach to non-interference based on partial equivalence relations. Due to the straightforward isomorphism existing between equivalence relations and partitioning closure operators (Sect. 2.2.3), the relation between this model of non-interference and our abstract non-interference is almost immediate. Consider a map $f : C \longrightarrow C$ and consider $\mathtt{R}$, $\mathtt{Q}$, equivalence relations on C. For what we proved in Sect. 2.2.3, the following partial equivalence relations (PER) on functions are equivalent:

$$\forall x_1, x_2 \in C \,.\, x_1 \,\mathtt{R}\, x_2 \;\Rightarrow\; f(x_1)\,\mathtt{Q}\; g(x_2)$$
$$\forall x_1, x_2 \in C \,.\; \mathit{Clo}^{\mathtt{R}}(x_1) = \mathit{Clo}^{\mathtt{R}}(x_2) \;\Rightarrow\; \mathit{Clo}^{\mathtt{Q}}(f(x_1)) = \mathit{Clo}^{\mathtt{Q}}(g(x_2))$$

Since the second one, with f instead of g, is narrow abstract non-interference, we can express narrow abstract non-interference inside the PER model, and in particular by substituting equivalence relations with closure operators. It is worth noting that this correspondence is only with narrow non-interference since the semantics f is computed on the concrete values, namely the output relation is applied to the results of the concrete computations.

Moreover, the relation between PER model and the notion of narrow abstract non-interference is even deeper when we consider non-deterministic systems. Indeed, there is a correspondence between additive closures and power domain relations, both introduced to cope with non-determinism ([106] for PER model, Sect. 6.5.2 for NANI).
Indeed, recall that the extension of a relation $\mathtt{R}$ on a powerdomain is $\mathcal{P}[\mathtt{R}\,]$ defined as

$$X\,\mathcal{P}[\mathtt{R}\,]\,Y \;\Leftrightarrow\; \forall x \in X.\,\exists y \in Y \,.\, x\,\mathtt{R}\; y \text{ and } \forall y \in Y.\,\exists x \in X \,.\, x\,\mathtt{R}\; y$$

If we consider a generic closure η inducing $\mathtt{R}$, we can rewrite this definition as

$$X\ \mathcal{P}[\mathtt{R}\,]\ Y \ \Leftrightarrow\ \forall x \in X.\, \exists y \in Y\,.\, \eta(x) = \eta(y) \text{ and } \forall y \in Y.\, \exists x \in X\,.\, \eta(x) = \eta(y)$$

which means, for each pair of set properties X and Y, we have

$$X\ \mathcal{P}[\mathtt{R}\,]\ Y \ \Leftrightarrow\ \bigcup_{x \in X} \eta(x) = \bigcup_{y \in Y} \eta(y)$$

At this point, if η is additive, then it commutes with the union, and therefore we obtain

$$X\ \mathcal{P}[\mathtt{R}\,]\ Y \ \Leftrightarrow\ \eta(X) = \eta(Y)$$

This equivalence shows that power domain relations in the PER model correspond to additive closure operators in narrow abstract non-interference. Moreover, both power domain relations in PER and additive closures in abstract non-interference are introduced, in the respective model, to cope with non-determinism.

6.6.2 Abstract non-interference vs robust declassification

Robust declassification has been introduced in [118] as a systematic method to drive declassification by characterizing what information flows from confidential to public variables, as we have introduced in Sect 6.4. In this section, we show the relation between robust declassification and the notion of abstract declassification introduced in Sect. 6.4, and such that, in the narrow case boils down to robust declassification with passive attackers [118], simply called declassification.

In order to adapt the declassification, introduced in [118], to the abstract declassification, defined before in terms of denotational semantics, we take a semantic variation of what is proposed in [118], which considers passive attackers only and a semantics observing only the initial and the final states of computations in deterministic systems. We follow [118] in defining the information leaked by an equivalence relation transformer $S[\eta, \rho]$ on Σ for each $\eta, \rho \in \mathit{uco}(\wp(\mathbb{V}^{\mathtt{L}}))$. Recall that (see Sect. 5.5) the transformer defined in [118], given an equivalence relation $\approx$, is $S[\approx]$ defined as follows:

$$\begin{array}{l}\forall s_1, s_2 \in \Sigma\,.\, \langle s_1, s_2\rangle \in S[\approx] \ \Leftrightarrow\ \mathit{Obs}_{s_1}(S, \approx) \equiv \mathit{Obs}_{s_2}(S, \approx)\\ \text{where } \mathit{Obs}_s(S, \approx) \stackrel{\text{def}}{=} \left\{\ \tau/\approx \ \middle|\ \tau \text{ trace of } S \text{ starting in } s\ \right\}\\ \text{and } \forall i < |\tau|\,.\, (\tau/\approx)_i = [\tau_i]_\approx\end{array}$$

First of all, let us consider the denotational abstraction of the set $\mathit{Obs}_s(S, \approx)$:

$$\mathit{Obs}^{\mathcal{D}}_s(S, \approx) \stackrel{\text{def}}{=} \lambda[s]_\approx.\ \left\{\ [\sigma_\dashv]_\approx \ \middle|\ \sigma \text{ finite trace and } \sigma_\vdash = s\ \right\}$$

We can further abstract this set in order to distinguish the relation observable on inputs and on outputs, exactly as we have done in abstract non-interference. For this reason we consider two closures on public value $\eta, \rho \in \mathit{uco}(\mathbb{V}^{\mathtt{L}})$ and we define:

$$Obs^{\mathcal{D}}_{s}(S,\langle\eta,\rho\rangle) \stackrel{\text{def}}{=} \lambda[s]_\eta.\ \left\{\ [\sigma_{\dashv}]_\rho \,\middle|\, \sigma \text{ finite trace and } \sigma_{\vdash} = s\ \right\}$$
$$\text{where } s_1 \in [s]_\eta \ \Leftrightarrow\ \eta(s_1^{\mathtt{L}}) = \eta(s^{\mathtt{L}})$$

At this point we have all we need for defining abstract declassification:

$$\begin{aligned} s_1 S[\eta,\rho] s_2 &\Leftrightarrow Obs^{\mathcal{D}}_{s_1}(S,\langle\eta,\rho\rangle) = Obs^{\mathcal{D}}_{s_2}(S,\langle\eta,\rho\rangle) \\ &\Leftrightarrow \lambda[s_1]_\eta.\ \left\{\ [\sigma_{\dashv}]_\rho \,\middle|\, \sigma \text{ finite trace and } \sigma_{\vdash} = s_1\ \right\} = \\ &\qquad\qquad \lambda[s_2]_\eta.\ \left\{\ [\sigma_{\dashv}]_\rho \,\middle|\, \sigma \text{ finite trace and } \sigma_{\vdash} = s_2\ \right\} \end{aligned}$$

which, in deterministic systems and by definition of $[\cdot]_\eta$ and $[\cdot]_\rho$, becomes:

$$s_1 S[\eta,\rho] s_2 \text{ iff } s_1^{\mathtt{L}} \approx_\eta s_2^{\mathtt{L}} \quad \text{and}(\forall \sigma,\delta \in \Sigma^+ .\ \sigma_{\vdash} = s_1\ \wedge\ \delta_{\vdash} = s_2\ \Rightarrow\ \sigma_{\dashv}^{\mathtt{L}} \approx_\rho \delta_{\dashv}^{\mathtt{L}}))$$

where $s_1, s_2 \in \Sigma$. It is simple to verify that $s_1 S[\eta,\rho] s_2$ if and only if $\eta(s_1^{\mathtt{L}}) = \eta(s_2^{\mathtt{L}})$ and, moreover, $\rho([\![P]\!](s_1)^{\mathtt{L}}) = \rho([\![P]\!](s_2)^{\mathtt{L}})$. Note that, this is true also with non-deterministic systems, whenever ρ is additive. This means that abstract declassification *a la* [118] is based on a formulation of non-interference which is stronger then abstract non-interference, since it may allow also deceptive flows, as we have in the narrow abstract non-interference case. This means that abstract declassification, for passive attackers, characterizes the information leaked in narrow abstract non-interference. The following examples show that in the narrow abstract non-interference the two methods provide the same partition of states, while in the abstract non-interference, our method is more precise.

Example 6.32. Consider the program fragment given in Example 6.30. The corresponding transition system is $\langle \Sigma, \longrightarrow\rangle$, where $\Sigma = \mathbb{V}^{\mathtt{H}} \times \mathbb{V}^{\mathtt{L}}$ and $\langle h, l\rangle \longrightarrow \langle h, l * h^2\rangle$. Then $\forall h_1, h_2 \in \mathbb{V}^{\mathtt{H}}$ and $\forall l_1, l_2 \in \mathbb{V}^{\mathtt{L}}$ we have that

$$\langle h_1, l_1\rangle S[id, Par]\langle h_2, l_2\rangle \text{ iff } l_1 = l_2 \text{ and } Par(l_1 * h_1^2) = Par(l_2 * h_2^2)$$

Clearly, this second condition is always true when $Par(l_1) = 2\mathbb{Z}$, but when $Par(l_1) = 2\mathbb{Z} + 1$ implies that $Par(h_1) = Par(h_2)$. Therefore, we conclude that

$$\langle h_1, l_1\rangle S[id, Par]\langle h_2, l_2\rangle \text{ iff } l_1 = l_2 \text{ and} Par(l_1) = 2\mathbb{Z}\ \Rightarrow\ Par(h_1) = Par(h_2)$$

Then, the induced partition of states is the set

$$\left\{\ \langle \mathbb{Z}, l\rangle \,\middle|\, l \in 2\mathbb{Z}\ \right\} \cup \left\{\ \langle 2\mathbb{Z}, l\rangle \,\middle|\, l \in 2\mathbb{Z}+1\ \right\} \cup \left\{\ \langle 2\mathbb{Z}+1, l\rangle \,\middle|\, l \in 2\mathbb{Z}+1\ \right\}$$

which is the same partition induced in Example 6.30. The leaked information is parity of h when l is odd.

Example 6.33. Consider the program fragment in Example 6.19 and consider the closure $\eta(\mathbb{Z}) = \{\mathbb{Z}, [2,4], [5,8], \{5\}, \varnothing\}$. We want to compute the most concrete closure ϕ, flow-irredundant, such that $(\eta)P\ (\phi \Rightarrow id)$. Note that for each $h_1, h_2 \in \mathbb{V}^{\mathtt{H}}$, and for each $Y \in \eta(\mathbb{Z})$, we have

$$[\![P]\!](h_1, Y) = [\![P]\!](h_2, Y) \text{ iff } h_1\ mod\ 3 = h_2\ mod\ 3$$

This means that, for each $Y \in \eta(\mathbb{Z})$, we have $\pi_Y = \{3\mathbb{Z}, 3\mathbb{Z}+1, 3\mathbb{Z}+2\}$, and therefore $\phi = \mathcal{P}(\{3\mathbb{Z}, 3\mathbb{Z}+1, 3\mathbb{Z}+2\})$.
Consider the transition system associated with the program, which has transition relation $\langle h, l\rangle \longrightarrow \langle h, l + (h \ mod\ 3)\rangle$. Then

$$\langle h_1, l_1\rangle S[\eta, id]\langle h_2, l_2\rangle \text{ iff } \eta(l_1) = \eta(l_2) \text{ and } l_1 + (h_1 \ mod\ 3) = l_2 + (h_2 \ mod\ 3)$$

This in particular implies that the two states $\langle 3, 2\rangle$ and $\langle 6, 4\rangle$ which are in the relation $\Pi(\eta, id)$ cannot be in the relation $S[\eta, id]$, which in this case distinguish 3 from 6.

6.7 Discussion

In this chapter, we have defined a weakening of the notion of non-interference in language-based security, called abstract non-interference. In particular, this notion is based on the idea of modeling attackers as abstract interpretations of the program's semantics. This model of program security, provided us the framework where studying the secrecy level of programs in the lattice of abstract interpretations. This is obtained by defining a domain transformer which, given the semantics of a program, characterizes the most concrete property that can be observed in the public output, without disclosing confidential properties. In the following we will call this abstraction the most concrete *public observer*, which represents the most powerful harmless attacker. We showed also that this construction can be used for deriving the most concrete canonical attacker, by using a fixpoint construction, allowing us to compare the secrecy degree of programs by comparing the canonical attackers for which they are secure. Finally we showed that, the same model, allows us to characterize the maximal amount of private information that flows, which will be also called the most abstract *private observable*. This abstraction can be used for declassification, since it represents the most abstract property that we have to declassify in order to guarantee non-interference. Weakening non-interference is not new in language-based security. In particular, two works have strong relations with abstract non-interference. The first is [106], and it models the observational capability of attackers by using equivalence relations, which can be associated with particular closure operators as we have seen in Chap. 2. The other work is the one on robust declassification [118], where attackers are characterized again by equivalence relations, and where the released information is identified by transforming these equivalence relations.
It is worth noting that the more powerful is the attacker and the more precise is the private property released by the program. This means that the more concrete is the public observer and the more abstract is the information that can be kept private. This strong relation between these two transformers will be formalized in the following (in Chap. 8) as an adjunction in the algebra of domain transformers described in Chap. 3.

One of the key aspects of the notion of abstract non-interference is the fact that it is a *semantic* property. On one side, this makes the notion of abstract non-interference not very practical, since it requires the computation of the complete denotational semantics. This problem can be solved by defining a compositional proof system that allows to certify programs inductively on the syntax of the programming language, as we will see later on, in the following chapter.
On the other side, this semantic nature, makes abstract non-interference a very versatile notion, since it is sufficient to change the semantics in order to change the enforced notion of non-interference, as it is shown, for example, in this chapter for non deterministic programs and as we will see in Chap. 9 for avoiding timing channels.

7

Proving Abstract Non-Interference

The knowledge follows different ways [...]
we cannot see beyond next turn.
Khalil Gibran

Abstract non-interference is based on the idea that the model of an attacker is an abstract interpretation of the semantics of the program. A program satisfies abstract non-interference relatively to some given abstraction (attacker) if the abstraction *obfuscates* any possible interference between confidential and public data. In the previous chapter, we introduce a step-by-step abstraction of standard non-interference by specifying abstract non-interference as a property of the semantics of the program. The idea of modeling attackers as abstract domains provides advanced methods for deriving attackers by systematically transforming the corresponding abstract domains. Any abstraction for which the program satisfies abstract non-interference, is both a model of an harmless attacker and a certificate for the security degree of the program. However, the original definition of abstract non-interference is not specified inductively on program's syntax but rather it is derived as an abstraction of the concrete semantics of the whole program. This makes the use of abstract non-interference hard in automatic program certification mechanisms, such as in proof-carrying code architectures [97] and in type-based verification algorithms. The logical approach to secure information flow is not new. In [36] dynamic logic is used for characterizing secure information flows, deriving a theorem prover for checking programs. In [7] and in [6] axiomatic approaches for checking secure information flows are provided (see Sect. 5.2.4). However, these works don't characterize the power of the attacker.

In this chapter, we introduce a compositional proof system whose aim is to certify abstract non-interference in programming languages, which means proving that the program satisfies an abstract non-interference constraint relatively to some

given abstraction of its input/output. Abstractions are specified in the standard abstract interpretation [28] framework. Assertions in the proof system have the form of Hoare triples: $(\eta)P\,(\rho)$ where P is a program fragment and η and ρ are abstractions of program's data. However, the interpretation of abstract non-interference assertions is rather different from partial correctness assertions (see [9]): $(\eta)P\,(\rho)$ means that P is unable to disclose secrets if input and output values on public variables are approximated respectively in η and ρ. Hence, abstract non-interference assertions specify the secrecy of a program relatively to a given model of an attacker and the proof system specifies how these assertions can be composed in a syntax-directed *a la* Hoare deduction of secrecy. We introduce two proof systems for checking abstract non-interference. The first deals with a narrow abstract non-interference. The advantage of narrow abstract non-interference is in the simplicity of the proof system and in its natural derivation from the operational semantics of the language. This proof system is necessary in order to derive a proof system for the more general notion of abstract non-interference assertions. We prove that the proof systems are sound relatively to the standard semantics of an imperative programming language. Both proof systems provide a deeper insight in abstract non-interference, by specifying how assertions concerning secrecy compose with each other. This is essential for any static semantics for secrecy devoted to derive certificates specifying the degree of secrecy of a program. The results presented in this chapter has been published in [54].

7.1 Axiomatic abstract non-interference

In this section, we introduce a proof system for certifying abstract non-interference of programs. We assume a set Φ of basic formulas which can be freely generated from some given set of predicates on $\mathbb{V}^{\mathtt{L}}$ with the basic connectives $\wedge$, $\vee$ and $\neg$. An abstract domain $\rho \in \mathit{uco}(\wp(\mathbb{V}^{\mathtt{L}}))$ can therefore be represented as a $\wedge$-closed set of formulas in Φ. The semantics of a set of formulas is the corresponding abstract domain. The interpretation of $\bigsqcap$ and $\bigsqcup$ are therefore straightforward. In the following of this chapter, we always consider $\phi = \mathit{id}$ in abstract non-interference, for this reason we will use a simplified notation, i.e., $(\eta)P\,(\rho) \stackrel{\text{def}}{=} (\eta)P\,(\mathit{id} \rightsquigarrow \rho)$. Moreover, as noticed in the PER model (see Sect. 6.6.1), if we have to handle a tuple of variable, then a property of this tuple is indeed a tuple of properties, one for each element of the tuple. Therefore, given a tuple of values $l \stackrel{\text{def}}{=} \langle l_1, l_2, \ldots, l_n\rangle$ then we suppose that a property ρ of this tuple is indeed $\rho(l) = \langle \rho_1(l_1), \rho_2(l_2), \ldots, \rho_n(l_n)\rangle$, where ρ_i are properties on the domain of data. In the following, if x is a variable in the tuple l, then we denote by ρ_x, the component on x of the property ρ.

7.1.1 Proof system for invariants

In order to certify secrecy when implicit flows may occur, we need to model the properties that are invariant during the execution of programs. Intuitively, an

$$\textbf{I1:}\ \{\top\}_{\mathtt{L}}\ c\ \{\top\}_{\mathtt{L}} \qquad \textbf{I2:}\ \{\rho\}_{\mathtt{L}}\ \textbf{nil}\ \{\rho\}_{\mathtt{L}} \qquad \textbf{I3:}\ \frac{x : \mathtt{H}}{\{\rho\}_{\mathtt{L}}\ x := e\ \{\rho\}_{\mathtt{L}}}$$

$$\textbf{I4:}\ \frac{\{\rho\}\ \langle e, x\rangle\ \{\rho\},\ x : \mathtt{L}}{\{\rho\}_{\mathtt{L}}\ x := e\ \{\rho\}_{\mathtt{L}}} \qquad \textbf{I5:}\ \frac{\{\rho\}_{\mathtt{L}}\ c_1\ \{\rho\}_{\mathtt{L}},\ \{\rho\}_{\mathtt{L}}\ c_2\ \{\rho\}_{\mathtt{L}}}{\{\rho\}_{\mathtt{L}}\ c_1; c_2\ \{\rho\}_{\mathtt{L}}}$$

$$\textbf{I6:}\ \frac{\{\rho\}_{\mathtt{L}}\ c\ \{\rho\}_{\mathtt{L}}}{\{\rho\}_{\mathtt{L}}\ \textbf{while}\ x\ \textbf{do}\ c\ \textbf{endw}\ \{\rho\}_{\mathtt{L}}} \qquad \textbf{I7:}\ \frac{\{\rho'\}_{\mathtt{L}}\ c\ \{\rho'\}_{\mathtt{L}},\ \rho' \sqsubseteq \rho}{\{\rho\}_{\mathtt{L}}\ c\ \{\rho\}_{\mathtt{L}}}$$

Table 7.1. Derivation of public invariants of programs.

abstraction is invariant for a program fragment P, written $\{\rho\}_{\mathtt{L}}\ P\ \{\rho\}_{\mathtt{L}}$, when by observing the property ρ of public inputs, we are not able to observe any difference in the ρ property of the corresponding public outputs. In other words, $\{\rho\}_{\mathtt{L}}\ P\ \{\rho\}_{\mathtt{L}}$ means that P is observably equivalent to **nil** as regards the observable property ρ. This information is essential in order to certify the lack of implicit flows relatively to an abstraction. These invariant abstractions are obtained with an *a la* Hoare proof system, where assertions are invariant properties of the form $\{\rho\}_{\mathtt{L}}\ P\ \{\rho\}_{\mathtt{L}}$, with $\rho \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$. Invariants of expressions are parametric on a public variable, the public variable to which they can be assigned.

Definition 7.1. *Let e be an expression in the language* Imp *and x a variable. The property ρ of the variable x is invariant in e, written $\models\ \{\rho\}\ \langle e, x\rangle\ \{\rho\}$, if:*

$$\forall l \in \mathbb{V}^{\mathtt{L}}, \forall h \in \mathbb{V}^{\mathtt{H}}\ .\ \rho_x([\![e]\!](h, l)) = \rho_x(l_{|_x})$$

where for any expression e, $[\![e]\!] : \Sigma \longrightarrow \mathbb{V}$ is the standard semantics of expressions and where $l_{|_x}$ is the value for x in the tuple l.

The intuition is that e does not change the property ρ of the value of x inside l. We extend this definition of invariant properties of expressions in order to define invariants of statements/programs.

Definition 7.2. *Let P be an* Imp *program. A property ρ is invariant in the program P, written $\models\ \{\rho\}_{\mathtt{L}}\ P\ \{\rho\}_{\mathtt{L}}$, if*

$$\forall l \in \mathbb{V}^{\mathtt{L}},\ \forall h \in \mathbb{V}^{\mathtt{H}}\ .\ \rho([\![P]\!](h, l)^{\mathtt{L}}) = \rho(l)$$

Public invariants for programs can be derived by induction on the syntax of Imp by using the proof system $\mathcal{I} = \{\textbf{I1}, \ldots, \textbf{I7}\}$ whose rules are defined in Table 7.1 and explained in the following.

[**I1**] Rule **I1** says that the property $\top$ is invariant for any program. This holds since $\top$ is the property unable to distinguish any difference among values. Therefore, any change due to the execution of a program cannot be observed through the property $\top$.

[**I2**] Rule **I2** says that any property is invariant for the program **nil**. This holds since **nil** does not change public data, and therefore public data properties are left unchanged.

[**I3**] When we have an assignment to private variables, then the semantics behaves as **nil** relatively to public values, therefore rule **I3** is similar to **I2**, since, by definition, invariants are defined only for low variables.

[**I4**] In **I4** if a property is invariant for the evaluation of an expression as regards the low variable x, then it is invariant for the assignment of the expression to x. Consider for example the expression $l + 2$, then the property *Sign* (which abstracts the sign of an integer variable) is not invariant, since if we consider the input value $l = -1$, then we have that $\mathit{Sign}(l + 2) = \mathit{Sign}(1) = + \neq \mathit{Sign}(l) = -$. On the other hand, we have that *Par* (which abstracts the parity of an integer variable) is invariant for this expression as regards the variable l, since the operation $l + 2$ doesn't change the parity of the value assigned to l. Now, if the statement is $l := l + 2$, then $\{\mathit{Par}\}_{\mathtt{L}}\ l := l + 2\ \{\mathit{Par}\}_{\mathtt{L}}$. Note that, in this rule, it is important to consider, if $\mathbb{V}^{\mathtt{L}} = \mathbb{V}_1 \times \ldots \times \mathbb{V}_n$, only properties $\rho \in \mathit{uco}(\wp(\mathbb{V}^{\mathtt{L}}))$ such that $\rho(\langle x_1, \ldots, x_n\rangle) = \langle \rho_1(x_1), \ldots, \rho_n(x_n)\rangle$, where $\forall i.\ \rho_i \in \mathit{uco}(\wp(\mathbb{V}_i))$.

[**I5**] Rule **I5** says that the invariants distribute on the sequential composition. Hence, if for example we consider the program $l := l + 2;\ h := h - 1$, then we know, by the rule **I3**, that $\{\mathit{Par}\}_{\mathtt{L}}\ h := h - 1\ \{\mathit{Par}\}_{\mathtt{L}}$ and, by the rule **I4**, that $\{\mathit{Par}\}_{\mathtt{L}}\ l := l + 2\ \{\mathit{Par}\}_{\mathtt{L}}$. Therefore, we obtain the invariant assertion $\{\mathit{Par}\}_{\mathtt{L}}\ l := l + 2;\ h := h - 1\ \{\mathit{Par}\}_{\mathtt{L}}$.

[**I6**] Rule **I6** states that, given a **while** statement, if a property is invariant for the body, then the same property is invariant for the whole statement. This rule holds since the only modifications of variables made by the **while**, are made by its body.

[**I7**] Weakening (**I7**) says that any more abstract property of an invariant is still invariant.

A derivation in the proof system of public invariants in Table 7.1 is denoted $\vdash_{\mathcal{I}}$. The following theorem shows that the proof system for invariants is *sound* as regards the given definition of invariant properties.

Theorem 7.3. *Let* $P \in$ IMP *and* $\mathbb{V}^{\mathtt{L}} = \mathbb{V}_1 \times \ldots \times \mathbb{V}_n$, $n = \left|\left\{\ x \in \mathit{Var} \,\middle|\, x : \mathtt{L}\ \right\}\right|$. *Let* $\rho_i \in \mathit{uco}(\wp(\mathbb{V}_i))$, *and* $\rho = \langle \rho_1, \ldots, \rho_n\rangle$. *If* $\vdash_{\mathcal{I}} \{\rho\}_{\mathtt{L}}\ P\ \{\rho\}_{\mathtt{L}}$ *then* $\models \{\rho\}_{\mathtt{L}}\ P\ \{\rho\}_{\mathtt{L}}$.

Proof. The proof is by induction on the rules in Table 7.1. The rules **I1** and **I2** are trivial since the closure $\top$ makes each element equal to the element $\top$, while $[\![\mathbf{nil}]\!](h_1, l) = l$ by definition of **nil** and therefore for each ρ we have $\rho([\![\mathbf{nil}]\!](h_1, l)^L) = \rho(l)$. As regards **I3**, the semantics of assignment guarantees that $x : \mathtt{H}$ implies $\rho([\![x := e]\!](h, l)^{\mathtt{L}}) = \rho(l)$, being l left unchanged by the assignment. Consider **I4**, then the hypothesis $\{\rho\}\ \langle e, x\rangle\ \{\rho\}$ says that $\forall l \in \mathbb{V}^{\mathtt{L}}\ .\ \rho_x(l_{|_x}) = \rho_x([\![e]\!](h, l))$. We have to prove that, with this hypothesis, $\rho([\![x := e]\!](h, l)^{\mathtt{L}}) = \rho(l)$ holds. Since we have that the abstraction ρ of a

$$\textbf{N0:}\ \frac{[\eta][\![c]\!]\ (id) \sqsubseteq \rho}{[\eta]c\ (\rho)} \qquad \textbf{N1:}\ [\eta]c\ (\top) \qquad \textbf{N2:}\ \frac{\Pi(\eta) \sqsubseteq \Pi(\rho)}{[\eta]\textbf{nil}\ (\rho)}$$

$$\textbf{N3:}\ \frac{x \models\ [\eta]e\ (\rho),\ [\ Rel^{\eta} \sqsubseteq\ Rel^{\rho}],\ x : \mathtt{L}}{[\eta]x := e\ (\rho)} \qquad \textbf{N4:}\ \frac{x : \mathtt{H}\,,\quad Rel^{\eta} \sqsubseteq\ Rel^{\rho}}{[\eta]x := e\ (\rho)}$$

$$\textbf{N5:}\ \frac{[\eta]c_1\ (\rho),\quad [\rho]c_2\ (\beta)}{[\eta]c_1; c_2\ (\beta)} \qquad \textbf{N6:}\ \frac{\{\rho\}_{\mathtt{L}}\ c\ \{\rho\}_{\mathtt{L}}}{[\rho]\textbf{while}\ x\ \textbf{do}\ c\ \textbf{endw}\ (\rho)}$$

$$\textbf{N7:}\ \frac{[\eta']c\ (\rho'),\ \eta \sqsubseteq \eta',\ \rho' \sqsubseteq \rho}{[\eta]c\ (\rho)} \qquad \textbf{N8:}\ \frac{\forall i \in I\,.\ [\eta]c\ (\rho_i)}{[\eta]c\ (\bigsqcup_{i\in I} \rho_i)} \qquad \textbf{N9:}\ \frac{\forall i \in I\,.\ [\eta]c\ (\rho_i)}{[\eta]c\ (\bigsqcap_{i\in I} \rho_i)}$$

Table 7.2. Axiomatic narrow (abstract) non-interference

tuple of values is a tuple of abstractions, then after the execution of $x := e$ we obtain $\rho(l[x \mapsto [\![e]\!](h,l)]) = \rho(l)[x \mapsto \rho_x(l_{|_x})] = \rho(l)$. Consider now **I5** and suppose that $\{\rho\}_{\mathtt{L}}\ c_1\ \{\rho\}_{\mathtt{L}}$ and $\{\rho\}_{\mathtt{L}}\ c_2\ \{\rho\}_{\mathtt{L}}$, namely $\forall l \in \mathbb{V}^{\mathtt{L}}\,,\ h \in \mathbb{V}^{\mathtt{H}}$ we have $\rho([\![c_1]\!](h,l)^{\mathtt{L}}) = \rho(l)$ and $\rho([\![c_2]\!](h,l)^{\mathtt{L}}) = \rho(l)$. We have the following equalities $\rho([\![c_1;c_2]\!](h,l)^{\mathtt{L}}) = \rho([\![c_2]\!]([\![c_1]\!](h,l))^{\mathtt{L}}) = \rho([\![c_2]\!](h',l')^{\mathtt{L}}) = \rho(l')$ with $l' = [\![c_1]\!](h,l)^{\mathtt{L}}$ therefore $\rho(l') = \rho(l)$, so we have the thesis. **I6** holds since the only modifications that the statement **while** x **do** c **endw** can do are made by c. Also **I7** is straightforward from the definition of invariants.

7.1.2 Proof system for Narrow non-interference

We can now introduce a proof system for narrow abstract non-interference. In this case we look for sufficient conditions that allow to deduce that a statement satisfies narrow non-interference, simply by statically analyzing the syntax of the statement. The rules of this proof system are specified in Table 7.2. In particular, as we will see later on, in the proof system we introduce a semantic rule, based on the derivation of secret kernels introduced in the previous chapter. The reason that lead us to consider this construction as a rule in the proof system is that, the only axiom in the proof system is not so much significant, therefore without the semantic rule, it is difficult to derive significant non-interference properties of generic programs. Therefore, the idea is to use the semantic construction on the single statements of a program, and then to combine the different non-interference properties by using the other rules of the proof system. This observation makes also clear why we are interested in proving that the proof system, without the semantic rule, is at least sound. Let us explain these rule:

[**N0**] Rule **N0** derives from Th. 6.17. It states that given a program c and an input observation η we can derive the most concrete output observation that makes the program secret. This corresponds to finding the strongest post-condition (viz. the most concrete abstract domain) for the program c with precondition

η such that narrow abstract non-interference holds. This is a "semantic rule", because it involves the construction of the abstract domain $[\eta][\![c]\!]\ (id)$, which is equivalent to compute the concrete semantics of the command c. However, this rule allows us to include in the narrow abstract non-interference proofs, also assertions which can be systematically derived as an abstract domain transformation as shown in Sect. 6.3.2.

[**N1**] Rules **N1** says that if the output observation is the property $\top$, then the input can be any property. Again, this holds because $\top$ is not able to distinguish different public data.

[**N2**]] Rule **N2** says that **nil** is secret for any possible attacker such that the partition induced by input observation is more concrete than the one induced by the output observation. This condition is necessary since in this case abstract non-interference corresponds to saying $\forall l_1, l_2 \,.\, \eta(l_1) = \eta(l_2) \;\Rightarrow\; \rho(l_1) = \rho(l_2)$ which holds iff $Rel^{\eta} \sqsubseteq Rel^{\rho}$.

[**N3**] Rule **N3** considers a notion of secrecy extended to expressions, depending on a fixed variable to which the expression has to be assigned. Formally, we can define this secrecy for expression as follows:

$$x \models [\eta]e\ (\rho) \text{ iff } \forall l_1, l_2 \in \mathbb{V}^{\mathtt{L}} \,.\, \eta(l_1) = \eta(l_2) \;\Rightarrow\; \forall h_1, h_2 \in \mathbb{V}^{\mathtt{H}} \,. \\ \rho_x([\![e]\!](h_1, l_1)]) = \rho_x([\![e]\!](h_2, l_2)])$$

Being the variable x public, the secrecy of the expression distributes on the assignment when the partition induced by the input observation is more concrete than the output one. This condition, on the induced partitions, is necessary only when there are public variables for which the assignment behaves as **nil** (see **N2**), i.e., when there are more than one public variable.

Example 7.4. For instance, consider the program fragment

$$l_1 := 2 * h * l_2$$

with $l_1, l_2 : \mathtt{L}$, then $\not\models [\top]l_1 := 2 * h * l_2\ (Par)$ since

$$Par([\![l_1 := 2 * h * l_2]\!](h, \langle l, 3\rangle)^{\mathtt{L}}) = \langle 2\mathbb{Z}, 2\mathbb{Z} + 1\rangle \text{ while} \\ Par([\![l_1 := 2 * h * l_2]\!](h, \langle l, 2\rangle)^{\mathtt{L}}) = \langle 2\mathbb{Z}, 2\mathbb{Z}\rangle.$$

This because $Rel^{\top} \not\sqsubseteq Rel^{Par}$, and therefore $\top(3) = \top(2)$ doesn't imply $Par(3) = Par(2)$.

Anyway, the condition between the two closures results unnecessary when the program contains only one low variable. Consider $l := h * 2$, we have that $l \models [\top]h * 2\ (Par)$, namely the multiplication by 2 hides the parity property of the computed value. This implies that $[\top]l := h * 2\ (Par)$.

[**N4**] Rule **N4** says that an assignment to a high variable is always secret when the partition induced by the input observation is more concrete than the one induced by the output observation, since an assignment to private variables

behaves as **nil** for the public variables. For instance, note that if we have the statement $h := h + 1$, then clearly

$$\rho([\![h := h + 1]\!](h, l_1)^{\mathtt{L}}) = \rho(l_1) \text{ and } \rho([\![h := h + 1]\!](h, l_2)^{\mathtt{L}}) = \rho(l_2)$$

This means that also in this case narrow non-interference corresponds to requiring that $\eta(l_1) = \eta(l_2) \;\Rightarrow\; \rho(l_1) = \rho(l_2)$.

[**N5**] Rule **N5** shows how we can compose attackers in presence of sequential composition of programs. In particular, two programs c_1 and c_2 compose when c_1 is secret for the output observation which is the input one that makes c_2 secret.

[**N6**] Rule **N6** controls the **while** statement. In particular, $\{\rho\}_{\mathtt{L}}\ c\ \{\rho\}_{\mathtt{L}}$ states that the program c is not acting on the property ρ of the public data, namely ρ is invariant in the execution of c, in the sense that the property ρ of public data is not changed by the execution of c. If this happens, then the behaviour of c observed from ρ is the same as the program **nil**, and therefore the fact that the **while** is executed or not is not distinguishable from an observer. We apply this rule also when the guard is a low variable, because narrow non-interference may observe also deceptive flows.

[**N7**] Rule **N7** is the consequence rule, which states that we can concretize the input observation and we can abstract the output one (see Prop. 6.9).

[**N8**,**N9**] The rules **N8** and **N9** say that both the least upper bound and the greatest lower bound of output observations making a program secret, still make the program secret.

We denote by $\mathcal{N} = \mathcal{I} \cup \{\mathbf{N0}, \ldots, \mathbf{N9}\}$ the proof system for narrow abstract non-interference and by $\mathcal{N}_0 = \mathcal{I} \cup \{\mathbf{N1}, \ldots, \mathbf{N9}\}$ the same proof system without the semantic rule **N0**. Being the rule **N0** a semantic rule, as we explained above, we are also interested in the properties of proof system, without this rule. In particular next result specifies that the proof system, without **N0**, is sound.

Theorem 7.5. *Let* $P \in \textsc{Imp}$ *and* $\mathbb{V}^{\mathtt{L}} = \mathbb{V}_1 \times \ldots \times \mathbb{V}_n$, $n = \left|\{\ x \in \mathit{Var} \,|\, x : \mathtt{L}\ \}\right|$. *Let* $\rho_i \in \mathit{uco}(\wp(\mathbb{V}_i))$, *and* $\rho = \langle \rho_1, \ldots, \rho_n \rangle$. *If* $\vdash_{\mathcal{N}_0} [\eta]P\ (\rho)$ *then* $\models [\eta]P\ (\rho)$.

Proof. We prove the soundness of the system inductively on the rules in Table 7.2. Consider $l_1, l_2 \in \mathbb{V}^{\mathtt{L}}$ and $h_1, h_2 \in \mathbb{V}^{\mathtt{H}}$. The first three rules hold from Proposition 6.9. Consider **N3**, the hypothesis says that $\eta(l_1) = \eta(l_2)$ implies $\rho_x([\![e]\!](h_1, l_1)) = \rho_x([\![e]\!](h_2, l_2))$, we have to prove that $\eta(l_1) = \eta(l_2)$ implies that $\rho([\![x := e]\!](h_1, l_1)^{\mathtt{L}}) = \rho([\![x := e]\!](h_2, l_2)^{\mathtt{L}})$. Hence, suppose $\eta(l_1) = \eta(l_2)$ and note that, being $\eta \sqsubseteq \rho$ then for each x and y we have that $\eta(x) = \eta(y)$ implies $\rho(x) = \rho(y)$. Let $l_1 = \langle x_1, \ldots, x, \ldots, x_n \rangle$ and $l_2 = \langle y_1, \ldots, y_n \rangle$, with $n \in \mathbb{N}$. For the condition above, the hypothesis $\eta(l_1) = \eta(l_2)$ means that $\forall i \leq n\ .\ \eta_i(x_i) = \eta_i(y_i)$ that implies that $\forall i \leq n\ .\ \rho_i(x_i) = \rho_i(y_i)$. Therefore the following equalities hold:

$$\begin{aligned}\rho([\![x := e]\!](h_1, l_1)^{\mathtt{L}}) &= \rho(\langle x_1, \ldots, [\![e]\!](h_1, l_1), \ldots, x_n\rangle)\\ &= \langle \rho_1(x_1), \ldots, \rho_x([\![e]\!](h_1, l_1)), \ldots, \rho_n(x_n)\rangle\\ &= \langle \rho_1(y_1), \ldots, \rho_x([\![e]\!](h_2, l_2)), \ldots, \rho_n(y_n)\rangle\\ &= \rho(\langle y_1, \ldots, [\![e]\!](h_2, l_2), \ldots, y_n\rangle) = \rho([\![x := e]\!](h_2, l_2)^{\mathtt{L}})\end{aligned}$$

Consider **N4**, suppose $\eta(l_1) = \eta(l_2)$, being $Rel^{\eta} \sqsubseteq Rel^{\rho}$, we have also that $\rho(l_1) = \rho(l_2)$. Moreover, note that $\rho([\![x := e]\!](h_1, l_1)^{\mathtt{L}}) = \rho(l_1)$, being $x : \mathtt{H}$. Analogously, we can note that $\rho([\![x := e]\!](h_2, l_2)^{\mathtt{L}}) = \rho(l_2)$, and since $\rho(l_1) = \rho(l_2)$ we have the thesis. Consider rule **N5**. The hypotheses of the rule say that $\forall l_1, l_2 \,.\, \eta(l_1) = \eta(l_2)$ we have $\forall h_1, h_2 \,.\, \rho([\![c_1]\!](h_1, l_1)^{\mathtt{L}}) = \rho([\![c_1]\!](h_2, l_2)^{\mathtt{L}})$ and $\forall l_1, l_2 \,.\, \rho(l_1) = \rho(l_2)$ we have $\forall h_1, h_2 \,.\, \beta([\![c_2]\!](h_1, l_1)^{\mathtt{L}}) = \beta([\![c_2]\!](h_2, l_2)^{\mathtt{L}})$. Suppose $\eta(l_1) = \eta(l_2)$ then the following implications hold.

$$\begin{aligned}\beta([\![c_1; c_2]\!](h_1, l_1)^{\mathtt{L}}) &= \beta([\![c_2]\!]([\![c_1]\!](h_1, l_1))^{\mathtt{L}}) = \beta([\![c_2]\!]([\![c_1]\!](h_1, l_1)^{\mathtt{H}}, [\![c_1]\!](h_1, l_1)^{\mathtt{L}})^{\mathtt{L}})\\ &= \beta([\![c_2]\!]([\![c_1]\!](h_1, l_1)^{\mathtt{H}}, [\![c_1]\!](h_2, l_2)^{\mathtt{L}})^{\mathtt{L}})\\ &= \beta([\![c_2]\!]([\![c_1]\!](h_2, l_2)^{\mathtt{H}}, [\![c_1]\!](h_2, l_2)^{\mathtt{L}})^{\mathtt{L}})\\ &= \beta([\![c_2]\!]([\![c_1]\!](h_2, l_2))^{\mathtt{L}}) = \beta([\![c_1; c_2]\!](h_2, l_2)^{\mathtt{L}})\end{aligned}$$

In order to show the soundness of **N6** we prove that $\{\rho\}_{\mathtt{L}}\, c\, \{\rho\}_{\mathtt{L}}$, i.e., $\rho([\![c]\!](h, l)^{\mathtt{L}}) = \rho(l)$, implies non-interference on the statement, namely

$$\rho([\![\textbf{while } x \textbf{ do } c \textbf{ endw}]\!](h_1, l_1)^{\mathtt{L}}) = \rho([\![\textbf{while } x \textbf{ do } c \textbf{ endw}]\!](h_2, l_2)^{\mathtt{L}})$$

for any $l_1, l_2 \in \mathbb{V}^{\mathtt{L}}$ and $h_1, h_2 \in \mathbb{V}^{\mathtt{H}}$ such that $\rho(l_1) = \rho(l_2)$. Let us denote $c_1 \stackrel{\text{def}}{=} \textbf{while } x \textbf{ do } c \textbf{ endw}$. At this point, we have to prove, by induction on the semantics, that $\rho([\![c_1]\!](h, l)^{\mathtt{L}}) = \rho(l)$ for any h, l. If $[\![x]\!](h, l) = \mathit{false}$, then by definition we have that $[\![c_1]\!] = [\![\textbf{nil}]\!]$ and therefore we have the thesis. Suppose now that it holds for **while**'s with a number of loops less or equal to n, we prove it for **while**'s with $n + 1$ iterations. Consider $[\![c_1]\!] = [\![c; c_1]\!]$ where c_1 has n iterations, namely we can apply the inductive hypothesis on c_1. Then

$$\begin{aligned}\rho([\![c; c_1]\!](h, l)^{\mathtt{L}}) &= \rho([\![c_1]\!]([\![c]\!](h, l))^{\mathtt{L}}) = \rho([\![c_1]\!]([\![c]\!](h, l)^{\mathtt{H}}, [\![c]\!](h, l)^{\mathtt{L}})^{\mathtt{L}})\\ &= \rho([\![c]\!](h, l)^{\mathtt{L}}) \quad \text{(by inductive hypothesis)}\\ &= \rho(l) \quad \text{(by the hypothesis of the rule on } c)\end{aligned}$$

Finally **N7**, **N8** and **N9** hold by Prop. 6.9.

Example 7.6. Consider the closure *Par* which observes parity, depicted in Fig. 6.1, and the program:

$$P \stackrel{\text{def}}{=} l := 2 * h;\ \textbf{while } h \textbf{ do } l := l + 2;\ h := h - 1 \textbf{ endw}$$

with security typing: $t = \langle h : \mathtt{H}, l : \mathtt{L}\rangle$ and $\mathbb{V}^{\mathtt{H}} = \mathbb{V}^{\mathtt{L}} = \mathbb{Z}$. Then we have that $l \models [\top]2 * h\ (\rho_1)$, where ρ_1 is the closure which is not able to distinguish even numbers, i.e., $\rho_1 = \curlyvee\left(\{2\mathbb{Z}\} \cup \left\{\ \{n\}\ \middle|\ n \text{ odd}\ \right\}\right)$. Therefore, by **N3**, $[\top]l := 2{*}h\ (\rho_1)$ (note that, since there is only one low variable we ignore the condition $Rel^{\eta} \sqsubseteq$

Rel^{ρ}). Consider now the **while** statement. We note that the operation $l+2$ leaves unchanged the parity of l, this means that if the input is even the output is even, and similarly if it is odd. Namely for each n such that $Par(n) = Par(l)$ then $Par([\![l+2]\!](h,n)) = Par(n+2) = Par(n) = Par(l)$. Therefore $\{Par\}\,\langle l+2, l\rangle\ \{Par\}$ which implies

$$\frac{\{Par\}\ \langle l+2, l\rangle\ \ \{Par\}}{\{Par\}_{\mathtt{L}}\ l := l+2\ \ \{Par\}_{\mathtt{L}}} \qquad \frac{h : \mathtt{H}}{\{Par\}_{\mathtt{L}}\ h := h+1\ \ \{Par\}_{\mathtt{L}}}$$

Therefore, by **I5**, we have that $\{Par\}_{\mathtt{L}}\ l := l+2;\ h := h-1\ \ \{Par\}_{\mathtt{L}}$. Now we can apply rule **N6** obtaining

$$\frac{\{Par\}_{\mathtt{L}}\ l := l+2;\ h := h-1\ \ \{Par\}_{\mathtt{L}}}{[Par]\mathbf{while}\ h\ \mathbf{do}\ l := l+2;\ h := h-1\ \mathbf{endw}\ (Par)}$$

Finally, note that $\rho_1 \sqsubseteq Par$ hence by **N7** we have also that $[\top]l := 2 * h\ (Par)$, therefore we can apply rule **N5** and we obtain that $[\top]P\ (Par)$.

Unfortunately, the system $\mathcal{N}_0$ is not complete, and in particular **N5** is the rule that introduces incompleteness.

Example 7.7. Consider the property *Par* observing parity, and the program:

$$P \stackrel{\text{def}}{=} l := 4 * h^2 + 4;\ \mathbf{while}\ h\ \mathbf{do}\ l := l\ mod\ 4;\ h := 0\ \mathbf{endw}$$

with security typing: $t = \langle h : \mathtt{H}\,, l : \mathtt{L}\,\rangle$ and $\mathbb{V}^{\mathtt{H}} = \mathbb{V}^{\mathtt{L}} = \mathbb{Z}$. Let us denote the **while** statement as $c \stackrel{\text{def}}{=} \mathbf{while}\ h\ \mathbf{do}\ l := l\ mod\ 4;\ h := 0\ \mathbf{endw}$. We can prove that

$$\models\ [\top]l := 4h^2 + 4\ (\rho_1) \text{ and } \models\ [\top]P\ (\rho_1)$$

where ρ_1 is defined in Example 7.6. These facts hold since the result of the assignment is always an even number multiple of 4, independently from the value of h (so the first fact holds). At this point, the **while** receives a multiple of 4 and therefore the result is always 0, implying the second fact. On the other hand, we have

$$\not\models\ [\rho_1]c\ (\rho_1)$$

since without the assignment, the **while** can receive any number, in particular it can receive as input numbers that are not multiples of 4. For these numbers the statement is not secret, for instance $\rho_1([\![c]\!](0,5)^{\mathtt{L}}) = 5 \neq \rho_1([\![c]\!](1,5)^{\mathtt{L}}) = 1$. This means that $\not\vdash_{\mathcal{N}\setminus\{\mathbf{N1}\}}\ [\top]P\ (\rho_1)$.

It is clear that rule **N0** makes the proof system complete. This is a straight consequence of Th. 6.17.

Corollary 7.8. *The proof system $\mathcal{N}$ is complete.*

Proof. If $\models\ [\eta]P\ (\rho)$ then $\rho \sqsupseteq\ [\eta][\![P]\!]\ (id)$ by Theorem 6.17. Therefore for by Rule **N0** we have that $\vdash_{\mathcal{N}}\ [\eta]P\ (\rho)$.

$$\textbf{A0:}\ \frac{(\eta)[\![c]\!]\ (id) \sqsubseteq \rho}{(\eta)c\ (\rho)} \qquad \textbf{A1:}\ (\eta)c\ (\top) \qquad \textbf{A2:}\ (\eta)\textbf{nil}\ (\rho)$$

$$\textbf{A3:}\ \frac{x \models\ (\eta)e\ (\rho),\ x : \mathtt{L}}{(\eta)x := e\ (\rho)} \qquad \textbf{A4:}\ \frac{x : \mathtt{H}}{(\eta)x := e\ (\rho)} \qquad \textbf{A5:}\ \frac{(\eta)c_1\ (\curlyvee\ (\rho)),\ \ [\rho]c_2\ (\curlyvee\ (\beta))}{(\eta)c_1; c_2\ (\curlyvee\ (\beta))}$$

$$\textbf{A6:}\ \frac{\{\rho\}_{\mathtt{L}}\ c\ \{\rho\}_{\mathtt{L}},\ x : \mathtt{H}}{(\rho)\textbf{while}\ x\ \textbf{do}\ c\ \textbf{endw}\ (\rho)} \qquad \textbf{A7:}\ \frac{(\eta)c\ (\rho),\ x : \mathtt{L}}{(\eta)\textbf{while}\ x\ \textbf{do}\ c\ \textbf{endw}\ (\rho)}$$

$$\textbf{A8:}\ \frac{(\eta)c\ (\rho'),\ \rho' \sqsubseteq \rho}{(\eta)c\ (\rho)} \qquad \textbf{A9:}\ \frac{\forall i \in I\ .\ (\eta)c\ (\rho_i)}{(\eta)c\ (\bigsqcup_{i\in I}\rho_i)} \qquad \textbf{A10:}\ \frac{\forall i \in I\ .\ (\eta)c\ (\rho_i)}{(\eta)c\ (\bigsqcap_{i\in I}\rho_i)}$$

Table 7.3. Axiomatic abstract non-interference

7.1.3 Proof system for Abstract non-interference

We now introduce in Table 7.3 a proof system for abstract non-interference, i.e., modeling how $(\eta)P\ (\rho)$ assertions compose inductively on program's syntax. As before, we are looking for sufficient conditions, checkable on the syntax, that allow us to derive non-interference properties about programs. Also in this case, we have to introduce a semantic rule, based on the derivation of secret kernels, described in the previous chapter, in order to be able to derive significant non-interference properties. Therefore, as in the narrow case, we prove that the proof system without the first semantic rule is at least sound. Also in this case, the task is to apply the first rule to single statements, and then to combine the resulting non-interference properties by using the other rules of the proof system. Let us explain the rules of this proof system:

[**A0**,**A1**] The rules **A0** and **A1** in Table 7.3 are similar to the ones in Table 7.2 and hold for the same reasons.

[**A2**] The rule **A2** differs from **N2** since abstract non-interference avoids deceptive flows. Indeed, checking non-interference, in this case, consists in checking if $\eta(l_1) = \eta(l_2)$ implies $\rho(\eta(l_1)) = \rho(\eta(l_2))$, for all the possible public values l_1 and l_2. It is immediate to note that this implication always holds.

[**A3**] In rule **A3** we consider the generalization of the notion of abstract non-interference to expressions as we made for the narrow one:

$$x \models\ (\eta)e\ (\rho) \text{ iff } \forall l \in \mathbb{V}^{\mathtt{L}}\ .\forall h_1, h_2 \in \mathbb{V}^{\mathtt{H}}\ .\rho_x([\![e]\!](h_1, \eta(l))) = \rho_x([\![e]\!](h_2, \eta(l)))$$

Moreover, as in **N3**, for applying this rule it is necessary to consider only abstractions of tuples that are tuples of abstractions. At this point we say that, if the expression that we have to assign to the public variable x is secret as regards η and ρ, then it is secret for the assignment of e to x.

[**A4**] Rule **A4** is straightforward, since the assignments to private variables leave unchanged public data, and therefore cannot generate insecure information

flows. In order to understand the difference between **A4** and **N4**, consider the example used for explaining **N4**, i.e., $h := h + 1$ then in abstract non-interference we compute the following sets: $\rho([\![h := h+1]\!](h, \eta(l_1))^{\mathtt{L}}) = \rho(\eta(l_1))$ and $\rho([\![h := h + 1]\!](h, \eta(l_2))^{\mathtt{L}}) = \rho(\eta(l_2))$, which are always equal whenever $\eta(l_1) = \eta(l_2)$.

[**A5**] The major difference between narrow and abstract non-interference is in rule **A5**. In this case we need to consider a narrow assertion for c_2 involving disjunctive domains. This is due to the fact that by definition abstract non-interference checks input properties on singletons while the output of the abstract non-interference assertion for c_1 deals with properties of sets of values. In order to cope with this 'type mismatch", we have to require narrow non-interference for the second statement in the composition. Next example shows that considering abstract non-interference for c_2 is not sufficient to achieve soundness.

Example 7.9. Consider *Par* and the program P in Example 7.7. We can prove that

$$(\top)l := 4h^2 + 4\ (Par) \text{ and } (Par)\mathbf{while}\ h\ \mathbf{do}\ l := l\ mod\ 4;\ h := 0\ \mathbf{endw}\ (\rho)$$

where $\rho \stackrel{\text{def}}{=} Par \cup \{0\}$. Indeed the first statement returns always an even number, while the second one, returns always even numbers if the low input is even, odd numbers if the low input is odd. On the other hand, $\not\vdash\ (\top)l := 4h^2 + 4$; **while** h **do** $l := l\ mod\ 4$; $h := 0$ **endw** (ρ) since

$$\rho([\![l := 4h^2 + 4;\ c]\!](0, \mathbb{Z})^{\mathtt{L}}) = \rho(4) = 2\mathbb{Z} \text{ while}$$
$$\rho([\![l := 4h^2 + 4;\ c]\!](1, \mathbb{Z})^{\mathtt{L}}) = \rho(0) = \{0\}$$

namely they are different.

Moreover, note that **A5** requires that for both c_1 and c_2 the output closures are additive maps, i.e., disjunctive abstract domains, which is a necessary condition as shown in the following example.

Example 7.10. Consider the program:

$$P \stackrel{\text{def}}{=} c_1; c_2 = l := (h\ mod\ 2)(2l\ mod\ 4) + (1 - (h\ mod\ 2))(l\ mod\ 2 + 1);$$
$$l := (l\ mod\ 2) * 4h + (1 - (l\ mod\ 2)) * (4h + 1)$$

with security typing: $t = \langle h : \mathtt{H}, l : \mathtt{L} \rangle$ and $\mathbb{V}^{\mathtt{H}} = \mathbb{V}^{\mathtt{L}} = \mathbb{Z}$. Consider the property $\rho = \{\mathbb{Z}, 4\mathbb{Z}, 4\mathbb{Z} + 1, 4\mathbb{Z} + 2, 4\mathbb{Z} + 3, \varnothing\}$ (not additive), then $(\top)c_1\ (\rho)$ since

$$\forall h \in 2\mathbb{Z}.\ \rho([\![c_1]\!](h, \mathbb{Z})^{\mathtt{L}}) = \rho(\{1, 2\}) = \mathbb{Z} \text{ and}$$
$$\forall h \in 2\mathbb{Z} + 1.\ \rho([\![c_1]\!](h, \mathbb{Z})^{\mathtt{L}}) = \rho(\{0, 2\}) = \mathbb{Z}$$

On the other hand, it is simple to show that $[\rho]c_2\ (\rho)$ since this statement leaves unchanged the abstraction of l, namely $\vdash\ (\top)P\ (\rho)$. But if we consider

the definition of abstract non-interference, then we have that $\not\models \ (\top)P\ (\rho)$ because if $h \in 2\mathbb{Z}$ then $\rho([\![P]\!](h,\mathbb{Z})^{\mathtt{L}}) = \rho(\{4h, 4h+1\}) = \mathbb{Z}$ while if $h \in 2\mathbb{Z}+1$ then $\rho([\![P]\!](h,\mathbb{Z})^{\mathtt{L}}) = \rho(\{4h+1\}) = 4\mathbb{Z}+1$. Note that the first statement is not secret if we consider the disjunctive completion of ρ in output.

[**A6**] Rule **A6** is equal to **N6**, since **A5** requires narrow non-interference.

[**A7**] Rule **A7** is straightforward from the definition of abstract non-interference and was absent in narrow non-interference for the presence of deceptive flows. Indeed, when we have a low guard no implicit flows are possible, and therefore if the body of the **while** is secret, then the whole **while** is surely secret.

[**A8**,**A9**,**A10**] The last three rules (**A8**, **A9** and **A10**) changes since in abstract non-interference we cannot concretize the input observation, as proved in Prop. 6.9.

The proof system for abstract non-interference in Table 7.3 is denoted $\mathcal{A} = \mathcal{N} \cup \{\mathbf{A0}, \ldots, \mathbf{A10}\}$ and the proof system without the semantic rules is denoted as $\mathcal{A}_0 = \mathcal{N}_0 \cup \{\mathbf{A1}, \ldots, \mathbf{A10}\}$. The following theorem proves the soundness of the proof system $\mathcal{A}_0$ with respect to the standard semantics of Imp.

Lemma 7.11. *$[\eta]P\ (\rho)$ with ρ additive implies that*

$$\forall H \subseteq \mathbb{V}^{\mathtt{H}}, \forall h \in \mathbb{V}^{\mathtt{H}}\ .\ \rho([\![P]\!](h,l)^{\mathtt{L}}) = \rho([\![P]\!](H,l)^{\mathtt{L}})$$

Proof. By hypothesis we have that $\forall l \in \mathbb{V}^{\mathtt{L}},\ \forall h' \in H$ we have $\rho([\![P]\!](h,l)^{\mathtt{L}}) = \rho([\![P]\!](h',l)^{\mathtt{L}})$. Being ρ additive we obtain the following equalities $\rho([\![P]\!](H,l)^{\mathtt{L}}) = \bigcup_{h' \in H} \rho([\![P]\!](h',l)^{\mathtt{L}}) = \rho([\![P]\!](h,l)^{\mathtt{L}})$.

Theorem 7.12. *Let $P \in$ Imp be a program and $\eta, \rho \in \mathrm{uco}(\mathbb{V}^{\mathtt{L}})$, such that $\rho = \langle \rho_1, \ldots, \rho_n \rangle$, where $n = \left|\{\ x \in \mathit{Var} \,|\, x : \mathtt{L}\ \}\right|$. If $\vdash_{\mathcal{A}_0}\ (\eta)P\ (\rho)$ then $\models\ (\eta)P\ (\rho)$.*

Proof. We prove the soundness of the system inductively on the rules in Table 7.3. Consider $l_1, l_2 \in \mathbb{V}^{\mathtt{L}}$ and $h_1, h_2 \in \mathbb{V}^{\mathtt{H}}$. The first three rules hold from Prop. 6.9. Consider **A3**, the hypothesis says that $\eta(l_1) = \eta(l_2)$ implies $\rho_x([\![e]\!](h_1, \eta(l_1))) = \rho_x([\![e]\!](h_2, \eta(l_2)))$, we have to prove that $\eta(l_1) = \eta(l_2)$ implies that $\rho([\![x := e]\!](h_1, \eta(l_1))^{\mathtt{L}}) = \rho([\![x := e]\!](h_2, \eta(l_2))^{\mathtt{L}})$. Let $l_1 = \langle x_1, \ldots, x, \ldots, x_n \rangle$ and $l_2 = \langle y_1, \ldots, y_n \rangle$, with $n \in \mathbb{N}$. By the hypothesis on η, we have that $\eta(l_1) = \eta(l_2)$ means $\forall i \leq n\ .\ \eta(x_i) = \eta(y_i)$. Therefore, the following equalities hold:

$$\begin{aligned}
\rho([\![x := e]\!](h_1, \eta(l_1))^{\mathtt{L}}) &= \rho(\langle \eta(x_1), \ldots, [\![e]\!](h_1, \eta(l_1)), \ldots, \eta(x_n) \rangle) \\
&= \langle \rho_1(\eta(x_1)), \ldots, \rho_x([\![e]\!](h_1, \eta(l_1))), \ldots, \rho_n(\eta(x_n)) \rangle \\
&= \langle \rho_1(\eta(y_1)), \ldots, \rho_x([\![e]\!](h_2, \eta(l_2))), \ldots, \rho_n(\eta(y_n)) \rangle \\
&= \rho(\langle \eta(y_1), \ldots, [\![e]\!](h_2, \eta(l_2)), \ldots, \eta(y_n) \rangle) \\
&= \rho([\![x := e]\!](h_2, \eta(l_2))^{\mathtt{L}})
\end{aligned}$$

Consider **A4**, suppose $\eta(l_1) = \eta(l_2)$. We can note that $\rho([\![x := e]\!](h_1, \eta(l_1))^{\mathtt{L}}) = \rho(\eta(l_1))$, being $x : \mathtt{H}$. Analogously, we have that $\rho([\![x := e]\!](h_2, \eta(l_2))^{\mathtt{L}}) = \rho(\eta(l_2))$,

and since $\eta(l_1) = \eta(l_2)$ we have the thesis. Consider **A5**, suppose $(\eta)c_1$ (ρ) and $[\rho]c_2$ (β). Consider $h_1, h_2 \in \mathbb{V}^{\mathtt{H}}$ and $l_1, l_2 \in \mathbb{V}^{\mathtt{L}}$ with $\eta(l_1) = \eta(l_2)$. The following implications hold:

$$\begin{aligned}\beta([\![c_1; c_2]\!](h_1, \eta(l_1))^{\mathtt{L}}) &= \beta([\![c_2]\!]([\![c_1]\!](h_1, \eta(l_1)))^{\mathtt{L}}) \\ &= \beta([\![c_2]\!](H_1, [\![c_1]\!](h_1, \eta(l_1))^{\mathtt{L}})^{\mathtt{L}}) = \textstyle\bigcup_{h\in H_1, l\in\eta(l_1)} \beta([\![c_2]\!](h, [\![c_1]\!](h_1, l)^{\mathtt{L}})^{\mathtt{L}})\end{aligned}$$

where $H_1 = [\![c_1]\!](h_1, \eta(l_1))^{\mathtt{H}}$. Note that $\rho([\![c_1]\!](h_1, \eta(l_1))^{\mathtt{L}}) = \rho([\![c_2]\!](h_2, \eta(l_2))^{\mathtt{L}})$, then for each $l \in \eta(l_1)$ there exists $l' \in \eta(l_2) = \eta(l_1)$ such that $\rho([\![c_1]\!](h_1, l)^{\mathtt{L}}) = \rho([\![c_2]\!](h_2, l')^{\mathtt{L}})$ and viceversa. This implies the following equalities.

$$\begin{aligned}&\textstyle\bigcup_{h\in H_1, l\in\eta(l_1)} \beta([\![c_2]\!](h, [\![c_1]\!](h_1, l)^{\mathtt{L}})^{\mathtt{L}}) = \bigcup_{h\in H_1, l'\in\eta(l_2)} \beta([\![c_2]\!](h, [\![c_1]\!](h_2, l')^{\mathtt{L}})^{\mathtt{L}}) \\ &\quad = \textstyle\bigcup_{l'\in\eta(l_2)} \beta([\![c_2]\!](H_2, [\![c_1]\!](h_2, l')^{\mathtt{L}})^{\mathtt{L}}) \quad \text{(by Lemma 7.11)} \\ &\quad = \beta([\![c_2]\!](H_2, [\![c_1]\!](h_2, \eta(l_2))^{\mathtt{L}})^{\mathtt{L}}) = \beta([\![c_2]\!]([\![c_1]\!](h_2, \eta(l_2)))^{\mathtt{L}}) \\ &\quad = \beta([\![c_1; c_2]\!](h_2, \eta(l_2))^{\mathtt{L}})\end{aligned}$$

where $H_2 = [\![c_1]\!](h_2, \eta(l_2))^{\mathtt{H}}$. Rule **A6** is the equal to **N6** and therefore holds by Th. 7.5 and from the fact that narrow non-interference implies abstract one. **A7** is straightforward by the definition of abstract non-interference. The last three rules hold by the properties of abstract non-interference (Prop. 6.9).

Next examples shows how we can apply the rule **A5**. In particular, the first one, shows that, by considering narrow non-interference in rule **A5**, then we guarantee the soundness of the rule.

Example 7.13. Consider *Par* and the program P in Example 7.7. We can prove that

$$(\top)l := 4h^2 + 4 \ (Par) \text{ and } [Par]\textbf{while } h \textbf{ do } l := l \ mod\ 4;\ h := 0 \textbf{ endw } (Par)$$

so, by rule **A5**, $(\top)l := 4h^2 + 4;$ **while** h **do** $l := l \ mod\ 4;\ h := 0$ **endw** (Par). Indeed, for instance, if we consider $l_1 = 4$ and $l_2 = 8$ then clearly $\top(4) = \top(8) = \top$ and

$$\begin{aligned}&Par([\![l := 4h^2 + 4;\ c]\!](0, \top)^{\mathtt{L}}) = Par([\![c]\!](0, 4h^2 + 4)^{\mathtt{L}}) = Par(4h^2 + 4) = 2\mathbb{Z} \text{ and} \\ &Par([\![l := 4h^2 + 4;\ c]\!](1, \top)^{\mathtt{L}}) = Par([\![c]\!](1, 4h^2 + 4)^{\mathtt{L}}) = Par(0) = 2\mathbb{Z}\end{aligned}$$

where $c \stackrel{\text{def}}{=}$ **while** h **do** $l := l \ mod\ 4;\ h := 0$ **endw**.

Example 7.14. Consider the program fragment

$$P \stackrel{\text{def}}{=} l := 2^h;\ \textbf{while } h \textbf{ do } l := 2 * l;\ h := h - 1\textbf{endw}$$

with security typing: $t = \langle h : \mathtt{H}, l : \mathtt{L} \rangle$ and $\mathbb{V}^{\mathtt{H}} = \mathbb{V}^{\mathtt{L}} = \mathbb{N}$. First of all we note that

$$l \models (\top)2^h \ (\rho_1) \text{ where } \rho_1 \stackrel{\text{def}}{=} \bigvee \left(\{\{2\}^{\mathbb{N}}\} \cup \left\{ n \,\middle|\, n \notin \{2\}^{\mathbb{N}} \right\}\right)$$

since the result is always an even number, independently from the initial value of h. This means that we can apply **A3**:

$$\frac{l \models\ (\top)2^h\ (\rho_1), l : \mathtt{L}}{(\top) l := 2^h\ (\rho_1)}$$

Consider the **while** statement, denoted by c, and the upper closure operator $\rho_2 \stackrel{\text{def}}{=} \curlyvee\left(\{\ n\{2\}^{\mathbb{N}} \,|\, n \in \mathbb{N} \text{ odd}\ \}\right)$. We note that $\{\rho_2\}\ \langle 2 * l, l\rangle\ \{\rho_2\}$, since the operation $2 * l$ does not change the property $n\{2\}^{\mathbb{N}}$ of the initial value of l, namely it does not change the odd factor of l. Therefore, we can apply **I4** to the low assignment and **I3** for the high assignment:

$$\frac{\{\rho_2\}\ \langle 2 * l, l\rangle\ \{\rho_2\}, l : \mathtt{L}}{\{\rho_2\}_{\mathtt{L}}\ l := 2 * l\ \{\rho_2\}_{\mathtt{L}}} \qquad \frac{h : \mathtt{H}}{\{\rho_2\}_{\mathtt{L}}\ h := h - 1\ \{\rho_2\}_{\mathtt{L}}}$$

and therefore by applying **I5** we obtain

$$\frac{\{\rho_2\}_{\mathtt{L}}\ l := 2 * l\ \{\rho_2\}_{\mathtt{L}}\,,\ \{\rho_2\}_{\mathtt{L}}\ h := h - 1\ \{\rho_2\}_{\mathtt{L}}}{\{\rho_2\}_{\mathtt{L}}\ l := 2 * l;\ h := h - 1\ \{\rho_2\}_{\mathtt{L}}}$$

Now we can apply **A6**

$$\frac{\{\rho_2\}_{\mathtt{L}}\ l := 2 * l;\ h := h - 1\ \{\rho_2\}_{\mathtt{L}}}{[\rho_2]\textbf{while}\ h\ \textbf{do}\ l := 2 * l;\ h := h - 1\ \textbf{endw}\ (\rho_2)}$$

and therefore we use **A5**:

$$\frac{(\top) l := 2^h\ (\rho_1),\ [\rho_2]\textbf{while}\ h\ \textbf{do}\ l := 2 * l;\ h := h - 1\ \textbf{endw}\ (\rho_2)}{(\top) P\ (\rho_2)}$$

The following example shows that the proof system $\mathcal{A}_0$ for abstract non interference in Table 7.3 is not complete.

Example 7.15. Consider the closure $\rho \stackrel{\text{def}}{=} \{\mathbb{Z}, 2\mathbb{Z}, 4\mathbb{Z}, \varnothing\}$ and consider the program

$$P \stackrel{\text{def}}{=} \textbf{while}\ h\ \textbf{do}\ l := (l\ \mathit{mod}\ 4) * (l \div 4);\ h := 0\ \textbf{endw}$$

with security typing: $t = \langle h : \mathtt{H}, l : \mathtt{L}\rangle$ and $\mathbb{V}^{\mathtt{H}} = \mathbb{V}^{\mathtt{L}} = \mathbb{Z}$. Note that $(\rho) P\ (\rho)$ since, for example, $\rho([\![P]\!](1, 2\mathbb{Z})^{\mathtt{L}}) = 2\mathbb{Z} = \rho([\![P]\!](0, 2\mathbb{Z})^{\mathtt{L}})$. But $\not\models\ \{\rho\}_{\mathtt{L}}\ P\ \{\rho\}_{\mathtt{L}}$ since $\rho([\![P]\!](1, 2)^{\mathtt{L}}) = \rho(0) = 4\mathbb{Z} \neq \rho(2) = 2\mathbb{Z}$.

The example above shows that **A6** is not complete, but it is not the only incomplete rule. In particular, by the same argument used in Example 7.7 for **N5**, **A5** is also incomplete. Even **A7** is incomplete, since the guard of the while can avoid interferences that may happen in the body, as shown in the following example. All the other rules are complete.

Example 7.16. Consider $\rho \stackrel{\text{def}}{=} \{\mathbb{Z}, \{0\}, 2\mathbb{Z}_0, 2\mathbb{Z}+1, \varnothing\}$, where $2\mathbb{Z}_0 \stackrel{\text{def}}{=} 2\mathbb{Z} \smallsetminus \{0\}$, and

$$P \stackrel{\text{def}}{=} \textbf{while } l_1 \textbf{ do } l_2 := \mathit{iszero}(l_1) * h^2;\ l_1 := 0 \textbf{ endw}$$

with security typing: $t = \langle h : \mathtt{H}, l_1, l_2 : \mathtt{L} \rangle$ and

$$\mathit{iszero}(x) = \begin{cases} 1 \text{ if } x = 0 \\ 0 \text{ otherwise} \end{cases}$$

Then we can prove that

$$\not\models\ (\rho) l_2 := \mathit{iszero}(l_1) * h^2;\ l_1 := 0\ (\rho)$$

since, if we take the low input $\langle 0, 2\mathbb{Z}_0 \rangle$ then we have

$$\rho([\![l_2 := \mathit{iszero}(l_1) * h^2;\ l_1 := 0]\!](1, \langle 0, 2\mathbb{Z}_0 \rangle)^{\mathtt{L}}) = \rho(\langle 0, 1 \rangle) = \langle 0, 2\mathbb{Z}+1 \rangle \neq$$
$$\rho([\![l_2 := \mathit{iszero}(l_1) * h^2;\ l_1 := 0]\!](2, \langle 0, 2\mathbb{Z}_0 \rangle)^{\mathtt{L}}) = \langle 0, 2\mathbb{Z}_0 \rangle$$

But it is straightforward that $(\rho)P\ (\rho)$. Indeed, for instance, $\rho([\![P]\!](h, \langle 0, 2\mathbb{Z}_0 \rangle)^{\mathtt{L}}) = \langle 0, 0 \rangle$ and $\rho([\![P]\!](h, \langle 2\mathbb{Z}_0, 2\mathbb{Z}_0 \rangle)^{\mathtt{L}}) = \langle 0, 0 \rangle$, since, inside the **while**, we always have $\mathit{iszero}(l_1) = 0$.

As above, for the proof system for narrow abstract non-interference $\mathcal{N}$, also for abstract non-interference, the semantic rule **A0** makes $\mathcal{A}$ complete. This is a straight consequence of Th. 6.17.

Corollary 7.17. *The proof system $\mathcal{A}$ is complete.*

Next result specifies a relation between derivations in the narrow and abstract non-interference proof systems. This result is in accordance with the expected relation between narrow and abstract non-interference, the first being stronger.

Lemma 7.18. *Consider $\eta, \rho \in \mathit{uco}(\mathbb{V}^{\mathtt{L}})$ and a program P in* Imp*, then*

$$[\eta]P\ (\rho)\ \Rightarrow\ [\eta]P\ (\curlyvee(\rho)).$$

Proof. Suppose $[\eta]P\ (\rho)$, namely $\forall h_1, h_2 \in \mathbb{V}^{\mathtt{H}}$ and $\forall l_1, l_2 \in \mathbb{V}^{\mathtt{L}}$ we have that $\eta(l_1) = \eta(l_2)$ implies $\rho([\![P]\!](h_1, l_1)^{\mathtt{L}}) = \rho([\![P]\!](h_2, l_2)^{\mathtt{L}})$. But since P is deterministic, then $[\![P]\!](h, l)^{\mathtt{L}}$ is a singleton in $\mathbb{V}^{\mathtt{L}}$. Therefore we can conclude, from the properties of disjunctive completion, that $\curlyvee(\rho)([\![P]\!](h_1, l_1)^{\mathtt{L}}) = \rho([\![P]\!](h_1, l_1)^{\mathtt{L}}) = \rho([\![P]\!](h_2, l_2)^{\mathtt{L}}) = \curlyvee(\rho)([\![P]\!](h_1, l_1)^{\mathtt{L}})$, namely we have non-interference.

Lemma 7.19. $x \models [\eta]e\ (\rho)\ \Rightarrow\ x \models (\eta)e\ (\rho)$

Proof. Suppose $x \models [\eta]e\ (\rho)$. Hence, by definition, for any $h_1, h_2 \in \mathbb{V}^{\mathtt{H}}$ and for any $l_1, l_2 \in \mathbb{V}^{\mathtt{L}}$ such that $\eta(l_1) = \eta(l_2)$, we have $\rho_x([\![e]\!](h_1, l_1)) = \rho_x(h_2, l_2)$. We want prove that for any $h_1, h_2 \in \mathbb{V}^{\mathtt{H}}$ and for any $l \in \mathbb{V}^{\mathtt{L}}$, we have $\rho_x([\![e]\!](h_1, \eta(l))) = \rho_x([\![e]\!](h_2, \eta(l)))$. Note that

$$\begin{aligned}
\rho_x([\![e]\!](h_1, \eta(l))) &= \rho_x\left(\textstyle\bigcup_{l'} \in \eta(l)[\![e]\!](h_1, l')\right) \\
&= \rho_x\left(\textstyle\bigcup_{l'} \in \eta(l)\rho_x([\![e]\!](h_1, l'))\right) \text{ (by Prop. 2.57)} \\
&= \rho_x\left(\textstyle\bigcup_{l'} \in \eta(l)\rho_x([\![e]\!](h_2, l'))\right) \text{ (by hypothesis)} \\
&= \rho_x\left(\textstyle\bigcup_{l'} \in \eta(l)[\![e]\!](h_2, l')\right) \text{ (by Prop. 2.57)} = \rho_x([\![e]\!](h_2, \eta(l)))
\end{aligned}$$

Theorem 7.20. *Let $P \in \textsc{Imp}$ be a program and $\eta, \rho \in uco(\mathbb{V}^{\mathtt{L}})$. If $\vdash_{\mathcal{N}_0} \ [\eta]P\ (\rho)$ then we have $\vdash_{\mathcal{A}_0} \ (\eta)P\ (\rho)$.*

Proof. We prove the implication by induction on the rules. For the axioms the proof is immediate. By Proposition 6.7 we have that in narrow non-interference implies abstract one. Consider now rule **N3**, consider $x := e$ with $x : \mathtt{L}$, by Lemma 7.19 we have that $x \models \ [\eta]e\ (\rho)$ implies that $x \models \ (\eta)e\ (\rho)$, then we can apply rule **A3**, and therefore we obtain that when $\vdash_{\mathcal{N}_0} \ [\eta]x := e\ (\rho)$ we have also $\vdash_{\mathcal{A}_0} \ (\eta)x := e\ (\rho)$. Rule **A4** is straightforward from **N4** when $x : \mathtt{H}$ since it requires less hypotheses. Consider rule **N5** and suppose that $\vdash_{\mathcal{N}_0} \ [\eta]c_1; c_2\ (\beta)$. This means that $\vdash_{\mathcal{N}_0} \ [\eta]c_1\ (\rho)$ and $\vdash_{\mathcal{N}_0} \ [\rho]c_2\ (\beta)$. By Lemma 7.18 this implies that $\vdash_{\mathcal{N}_0} \ [\eta]c_1\ (\curlyvee\ (\rho))$ and $\vdash_{\mathcal{N}_0} \ [\rho]c_2\ (\curlyvee\ (\beta))$. Now, by inductive hypothesis we have that $\vdash_{\mathcal{A}_0} \ (\eta)c_1\ (\curlyvee\ (\rho))$ and therefore, by rule **A5**, we obtain $\vdash_{\mathcal{A}_0} \ (\eta)c_1; c_2\ (\curlyvee\ (\beta))$. At this point, since $\beta \sqsupseteq \curlyvee\ (\beta)$, we can apply rule **A8** in order to derive $\vdash_{\mathcal{A}_0} \ (\eta)c_1; c_2\ (\beta)$. Rule **N6** and **A6** have the same hypothesis, therefore when $x : \mathtt{H}$ it is trivial to show that when $\vdash_{\mathcal{N}_0} \ [\rho]\textbf{while}\ x\ \textbf{do}\ c\ \textbf{endw}\ (\rho)$ then $\vdash_{\mathcal{A}_0} \ (\rho)\textbf{while}\ x\ \textbf{do}\ c\ \textbf{endw}\ (\rho)$. On the other hand it is immediate to prove that $\{\rho\}_{\mathtt{L}}\ c\ \{\rho\}_{\mathtt{L}}$ implies $[\rho]c\ (\rho)$ and therefore $(\rho)c\ (\rho)$ by Proposition 6.7. This means that if **N6** is applicable than also **A7** is applicable. Consider **N7** and suppose that $[\eta]P\ (\rho)$ since $[\eta']P\ (\rho')$ with $\eta' \sqsupseteq \eta$ and $\rho \sqsupseteq \rho'$. This means also that $[\eta]P\ (\rho)$ can be derived from $[\eta]P\ (\rho')$ (by using **N7**). At this point by inductive hypothesis we have that $\vdash_{\mathcal{A}_0} \ (\eta)P\ (\rho')$ and therefore we derive by **A8** $\vdash_{\mathcal{A}_0} \ (\eta)P\ (\rho)$ since $\rho \sqsupseteq \rho'$. It is immediate to prove that, by using the inductive hypothesis, the rules **A9** and **A10** can be applied when the hypotheses of **N8** and **N9** respectively, hold.

Next example shows that $\mathcal{A}$ is strictly weaker than $\mathcal{N}$. We show that if $\models \ [\eta]P\ (\rho)$ and $\vdash_{\mathcal{A}_0} \ (\eta)P\ (\rho)$, then $[\eta]P\ (\rho) \Rightarrow \ (\eta)P\ (\rho)$ does not imply that $\vdash_{\mathcal{N}_0} \ [\eta]P\ (\rho)$.

Example 7.21. Consider the property *Par* and the program:

$$P \stackrel{\text{def}}{=} h := h + 1; l := 2 * h$$

with security typing: $t = \langle h : \mathtt{H}, l : \mathtt{L} \rangle$ and $\mathbb{V}^{\mathtt{H}} = \mathbb{V}^{\mathtt{L}} = \mathbb{Z}$. Note that $[\mathit{Sign}]P\ (\mathit{Par})$ since

$$\forall l \in \mathbb{V}^{\mathtt{L}},\ h \in \mathbb{V}^{\mathtt{H}} \text{ we have } \mathit{Par}([\![P]\!](h,l)^{\mathtt{L}}) = \mathit{Par}(2 * h) = 2\mathbb{Z}$$

This means also that $\models \ (\mathit{Sign})P\ (\mathit{Par})$. Moreover, $\not\models \ [\mathit{Sign}]h := h + 1\ (\mathit{Par})$ since

$$\mathit{Sign}(2) = \mathit{Sign}(3) = \mathbb{Z}^+ \text{ and } \begin{array}{l} \mathit{Par}([\![h := h + 1]\!](h,2)^{\mathtt{L}}) = \mathit{Par}(2) = 2\mathbb{Z} \neq \\ \mathit{Par}([\![h := h + 1]\!](h,3)^{\mathtt{L}}) = \mathit{Par}(3) = 2\mathbb{Z} + 1 \end{array}$$

This means that $\not\vdash_{\mathcal{N}_0} \ [\mathit{Sign}]P\ (\mathit{Par})$. On the other hand, $\vdash_{\mathcal{A}_0} \ (\mathit{Sign})h := h+1\ (\mathit{Par})$ and $\vdash_{\mathcal{N}_0} \ [\mathit{Par}]l := 2 * h\ (\mathit{Par})$, therefore we can use **A5** since *Par* is disjunctive, and therefore we infer $\vdash_{\mathcal{A}_0} \ (\mathit{Sign})P\ (\mathit{Par})$.

7.2 Non-deterministic case

In this section, we extend the given proof system in order to derive abstract non-interference also for non-deterministic programs. The first step is the construction of the rule for the non-deterministic choice in the proof system $\mathcal{I}$:

$$\textbf{I8:}\ \frac{\forall i \in I\ .\ \{\rho_i\}_{\mathtt{L}}\ c_i\ \{\rho_i\}_{\mathtt{L}}}{\{\bigsqcup_{i\in I}\rho_i\}_{\mathtt{L}}\ \Box_i c_i\ \{\bigsqcup_{i\in I}\rho_i\}_{\mathtt{L}}}$$

Rule **I8** controls the non-deterministic choice in a rather standard way. Indeed, it says that an invariant property for a non-deterministic choice is the most abstract invariant among the ones for all the programs involved in the non-deterministic choice.
At this point, we have to modify the proof system $\mathcal{N}$. The problem is that it is not sufficient to add the rule for non-deterministic choice since the fact that the denotational semantics returns a set of values instead of a singleton, creates for **N5** the same problems that we found in **A5**. So we build a new rule **N′5** for composition and we introduce the rule for the non deterministic choice. In the following example we show that if we consider not additive output observations, as in **N5**, then we are not sound.

Example 7.22. Consider the program:

$$P = c_1;\ c_2 = l := 1 - (h\ mod\ 2) \Box\ l := 2 * (h\ mod\ 2) + 2 * (1 - (h\ mod\ 2));\\ l := (l\ mod\ 2) * 4h + (1 - (l\ mod\ 2)) * (4h + 1)$$

with security typing: $t = \langle h : \mathtt{H}, l : \mathtt{L} \rangle$ and $\mathbb{V}^{\mathtt{H}} = \mathbb{V}^{\mathtt{L}} = \mathbb{Z}$. Consider the property observing the modulus in the division by 4: $\rho = \{\mathbb{Z}, 4\mathbb{Z}, 4\mathbb{Z}+1, 4\mathbb{Z}+2, 4\mathbb{Z}+3, \varnothing\}$ (not additive), then we can show that $[\top]c_1\ (\rho)$ since

$$\forall h \in 2\mathbb{Z}.\, \forall l \in \mathbb{Z}\ \rho([\![c_1]\!](h,l)^{\mathtt{L}}) = \rho(\{1,2\}) = \mathbb{Z} \text{ and}\\ \forall h \in 2\mathbb{Z}+1.\, \forall l \in \mathbb{Z}\ \rho([\![c_1]\!](h,l)^{\mathtt{L}}) = \rho(\{0,2\}) = \mathbb{Z}$$

On the other hand, it is simple to show that $[\rho]c_2\ (\rho)$ since the abstraction of l does not depend on h. But if we consider the composition then we have that $\not\models\ [\top]P\ (\rho)$ because

$$\forall h \in 2\mathbb{Z}.\, \forall l \in \mathbb{Z}\ \rho([\![P]\!](h,l)^{\mathtt{L}}) = \rho(\{4h, 4h+1\}) = \mathbb{Z}\\ \forall h \in 2\mathbb{Z}+1.\, \forall l \in \mathbb{Z}\ \rho([\![P]\!](h,l)^{\mathtt{L}}) = \rho(\{4h+1\}) = 4\mathbb{Z}+1$$

while rule **N5** would infer that $\vdash\ [\top]P\ (\rho)$

Therefore, we have to add the following rules:

$$\textbf{N′5:}\ \frac{[\eta]c_1\ (\curlyvee\,(\rho)),\quad [\rho]c_2\ (\curlyvee\,(\beta))}{[\eta]c_1; c_2\ (\curlyvee\,(\beta))} \qquad \textbf{N10:}\ \frac{\forall i \in I\ .\ [\eta_i]c_i\ (\rho_i)}{[\bigsqcap_{i\in I}\eta_i]\Box_i c_i\ (\bigsqcup_{i\in I}\rho_i)}$$

N10 says that, if we have a non-deterministic choice among the elements of a set of programs, then this non-deterministic choice is secret for the attacker characterized, in input by the greatest lower bound of input observations for which the elements of the set are secret, and in output by the least upper bound of output observations of the same elements. For instance, note that if we have $c \stackrel{\text{def}}{=} l := 2 * h \,\square\, l := 2h + l$, and we consider $\rho = \curlyvee\left(\{2\mathbb{Z}\} \cup \left\{ \{n\} \,\middle|\, n \text{ odd} \right\}\right)$, then we obtain $[\rho]l := 2 * h$ (ρ) and $[Par]l := 2h + l$ (Par). Clearly the execution of c has to guarantee secrecy independently from the statement that is executed, so we have $[\rho]c$ (Par).

Lemma 7.23. *Let ρ and η additive and $[\eta]P$ (ρ), then we have also that for each $L_1, L_2 \in \wp(\mathbb{V}^{\mathtt{L}})$ and $H_1, H_2 \in \wp(\mathbb{V}^{\mathtt{H}})$ if $\eta(L_1) = \eta(L_2)$ then $\rho([\![P]\!](H_1, L_1)^{\mathtt{L}}) = \rho([\![P]\!](H_2, L_2)^{\mathtt{L}})$*

Proof. Being ρ additive we have

$$\rho([\![P]\!](H_1, L_1)^{\mathtt{L}}) = \bigcup_{h_1 \in H_1, l_1 \in L_1} \rho([\![P]\!](h_1, l_1)^{\mathtt{L}}).$$

Since also η is additive, we have that $\eta(L_1) = \eta(L_2)$ implies that for each $l_1 \in L_1$ there exists $l_2 \in L_2$ such that $\eta(l_1) = \eta(l_2)$. Namely we have that

$$\bigcup_{h_1 \in H_1, l_1 \in L_1} \rho([\![P]\!](h_1, l_1)^{\mathtt{L}}) \subseteq \bigcup_{h_2 \in H_2, l_2 \in L_2} \rho([\![P]\!](h_2, l_2)^{\mathtt{L}})$$

since $[\eta]P$ (ρ). Viceversa we can prove the other inclusion in a similar way, therefore we have that

$$\rho([\![P]\!](H_1, L_1)^{\mathtt{L}}) = \rho([\![P]\!](H_2, L_2)^{\mathtt{L}}).$$

Theorem 7.24. *The proof system $\mathcal{N}^{\mathrm{ND}} \stackrel{def}{=} \mathcal{N} \setminus \{\mathbf{N5}\} \cup \{\mathbf{N'5}, \mathbf{N10}\}$ is complete and the proof system $\mathcal{N}_0^{\mathrm{ND}} \stackrel{def}{=} \mathcal{N}_0 \setminus \{\mathbf{N5}\} \cup \{\mathbf{N'5}, \mathbf{N10}\}$ is sound.*

Proof. The completeness is straightforward for the presence of the rule **N0** (see Corollary 7.8).

In order to prove correctness of $\mathbf{N'5}$ we have to show that whenever the premises of the rule hold then the consequence holds as well. Consider ρ and β additive, namely $\rho = \curlyvee(\rho)$ and $\beta = \curlyvee(\beta)$. Then we suppose that $[\eta]c_1$ (ρ) and that $[\rho]c_2$ (β), namely if $\eta(l_1) = \eta(l_2)$ then $\rho([\![c_1]\!](h_1, l_1)^{\mathtt{L}}) = \rho([\![c_1]\!](l_2, h_2)^{\mathtt{L}})$ and if $\rho(l_1) = \rho(l_2)$ then $\beta([\![c_2]\!](h_1, l_1)^{\mathtt{L}}) = \beta([\![c_2]\!](l_2, h_2)^{\mathtt{L}})$. We have to prove that if $\eta(l_1) = \eta(l_2)$ then $\beta([\![c_1; c_2]\!](h_1, l_1)^{\mathtt{L}}) = \beta([\![c_1; c_2]\!](l_2, h_2)^{\mathtt{L}})$. Hence suppose $\eta(l_1) = \eta(l_2)$, the following equalities hold:

$$\begin{aligned}
\beta([\![c_1; c_2]\!](h_1, l_1)^{\mathtt{L}}) &= \beta([\![c_2]\!]([\![c_1]\!](h_1, l_1))^{\mathtt{L}}) = \beta([\![c_2]\!]([\![c_1]\!](h_1, l_1)^{\mathtt{H}}, [\![c_1]\!](h_1, l_1)^{\mathtt{L}})^{\mathtt{L}}) \\
&= \beta([\![c_2]\!]([\![c_1]\!](h_2, l_2)^{\mathtt{H}}, [\![c_1]\!](h_2, l_2)^{\mathtt{L}})^{\mathtt{L}}) \qquad \text{(By Lemma 7.23)} \\
&= \beta([\![c_2]\!]([\![c_1]\!](h_2, l_2))^{\mathtt{L}}) = \beta([\![c_1; c_2]\!](h_2, l_2)^{\mathtt{L}})
\end{aligned}$$

and so we have non-interference.

N10 is sound since $[\![\square_i c_{i \in I}]\!] = [\![c_k]\!]$ for some $k \in I$, and $[\eta_k]c_k$ (ρ_k), that holds by

hypothesis, implies by **N7**, that $[\sqcap_i \eta_i]c_k\ (\sqcup_i \rho_i)$.

Finally, we modify the proof system for abstract non-interference. In this case we have only to add the new rule for the non-deterministic choice:

$$\textbf{A11:}\ \frac{\forall i \in I\ .\ (\eta)c_i\ (\rho_i)}{(\eta)\Box_i c_i\ (\bigsqcup_{i\in I} \rho_i)}$$

The completeness of $\mathcal{A} \cup \{\textbf{A11}\}$ and the soundness of $\mathcal{A}_0 \cup \{\textbf{A11}\}$ is immediate.

Theorem 7.25. *The proof system $\mathcal{A}^{\mathrm{ND}}$ is complete, while the proof system $\mathcal{A}_0^{\mathrm{ND}}$ is only sound.*

7.3 Discussion

In this chapter, we have introduced a sound proof-system for both narrow and abstract non-interference. The advantage of a proof-system for abstract non-interference is that checking abstract non-interference can be easily mechanized. Both $\mathcal{N}$ and $\mathcal{A}$ can benefit of standard abstract interpretation methods for generating basic certificates for simple program fragments (rules **N0** and **A0**). The other rules allow us to combine certificates from program fragments in a proof-theoretic derivation of harmless models of attackers, certifying program secrecy. The interest in this technology is mostly related with its use in *a la* proof carrying code (PCC) verification of abstract non-interference, when mobile code is allowed. In this case in a PCC architecture, the code producer may create an abstract non-interference certificate that attests to the fact that the code secrecy cannot be violated by the corresponding model of the attacker. Then the code consumer may validate the certificate to check that the foreign code is secure for the corresponding model of attacker. The implementation of this technology requires an appropriate choice of a logic for specifying abstractions and an adequate logical framework where the logic can be manipulated. We believe that predicate abstraction [46, 70] is a fairly simple and easily mechanizable way for reasoning about abstract domains. More appropriate logics can be designed following the ideas in [8], even though a mechanizable logic for reasoning about abstractions is currently a major challenge in this field and deserves further investigations.

8

Abstract Non-Interference: A completeness problem

The hidden harmony is more beautiful than the visible one.
IPPOLITO

In the previous chapters, we introduced the notion of abstract non-interference whose aim is that of modeling the secrecy degree of programs by using abstract interpretation. In particular, we are able to characterize which is the observational capability of the most concrete *harmless* attacker, namely of the most concrete attacker that cannot disclose any confidential information. Moreover, we noticed that our model allows also to characterize what aspect of private information can flow during the execution of a given program, when non-interference fails. This latter fact puts our work in strong relation with the notion of robust declassification introduced in [118], even if we consider here only passive attackers, i.e., attackers that can only observe the computation. It is clear that the stronger is the attacker, the more information can be released by the program. Namely, the more concrete is the model of the harmless attacker, the more abstract is the confidential information that can be kept private. This observation gives an intuitive explanation of the adjoint relation existing between the actions of weakening attackers and of declassifying private information. In particular, we can note that when we derive the most concrete attacker model, then we are looking for the most concrete *public observer*, while when we derive the most abstract property the has to be declassified in order to guarantee non-interference, we are looking for the most abstract *private observable*. Indeed, the most concrete public observer is the model of the most powerful attacker that can observe only public data. While, the most abstract private observable is the maximal amount of information that a program releases during computation. This means that, whenever we declassify a property which contains this private observable, we surely guarantee secrecy.

In this chapter, we prove that this duality corresponds precisely to an adjunction in the lattice of abstract interpretations. This is achieved by considering

abstract non-interference as a generalization of both declassification for passive attackers and attack models. In this setting we prove that, under non restrictive hypothesis, abstract non-interference corresponds precisely to making abstract interpretation complete [65] relatively to the denotational semantics of programs. This derives directly from an abstract interpretation-based generalization of Joshi and Leino's approach to secure information flows [78], which makes this approach equivalent to a completeness problem. Abstract interpretation plays a key role here, providing the adequate framework where program properties can be compared by considering their relative precision. In particular, we prove that declassification and attack models are adjoint notions and they correspond respectively to the minimal complete refinement, providing the most concrete *public observer* property of a program, and the minimal complete simplification, providing the most abstract *private observable* property of the program. This chapter is based on the paper [57].

The chapter is structured as follows. In Sect. 8.1 we prove that abstract non-interference is a problem of completeness in the abstract interpretation framework, while in Sect. 8.2 and in Sect. 8.3 we prove that the most concrete public observer, modeling harmless attackers, and the most abstract private observable, characterizing declassification, are respectively the minimal simplification and refinement of a given abstract domain, in order to achieve completeness. The chapter ends in Sect. 8.4 by proving an adjunction between public observers and private observables.

8.1 Abstract Non-Interference as Completeness

Joshi and Leino's semantic-based approach to information flows [78] provides a way to interpret abstract non-interference as the problem of making an abstract domain complete [65] (see Sect. 5.1.3). We remind the reader that, in the Joshi and Leino's semantic-based approach to information flows, non-interference is modeled by the Eq. 5.1, which is:

$$\mathtt{HH}\ ;\ P;\ \mathtt{HH}\ \doteq\ P\ ;\ \mathtt{HH}$$

where $\mathtt{HH}$ is a program that "assigns to h an arbitrary value". Let us consider now, the denotational semantics of a program P, denoted by $[\![P]\!]$. Then we can show that the equation above can be rewritten as a *backward* completeness problem by describing the semantics of $\mathtt{HH}$ as an abstraction. Indeed, the program that associates with any private variable an arbitrary value can be interpreted as the closure that abstracts the private value to the *"don't know"* abstract value, i.e., to the set of all the possible values for private variables. Therefore, we define the function $\mathcal{H} : \wp(\mathbb{V}) \longrightarrow \wp(\mathbb{V})$ in the following way (recall that $\wp(\mathbb{V}) = \wp(\mathbb{V}^{\mathtt{H}} \times \mathbb{V}^{\mathtt{L}})$):

$$\mathcal{H} = \lambda X.\, \langle \mathbb{V}^{\mathtt{H}}, X^{\mathtt{L}} \rangle \text{ where } X^{\mathtt{L}} \stackrel{\text{def}}{=} \left\{\ l \,\middle|\, \langle h, l\rangle \in X\ \right\}$$

It is straightforward to prove its monotonicity, idempotence and extensivity, namely it is an upper closure operator. Hence, we can conclude that

$$[\![\mathtt{H\,H}\ ;\ P;\ \mathtt{H\,H}]\!] = \mathcal{H} \circ [\![P]\!] \circ \mathcal{H} \qquad [\![P;\ \mathtt{H\,H}]\!] = \mathcal{H} \circ [\![P]\!]$$

This means that, non-interference can be equivalently formalized in the equation

$$\mathcal{H} \circ [\![P]\!] \circ \mathcal{H} = \mathcal{H} \circ [\![P]\!]$$

which is a *backward completeness problem* (see Sect. 2.2.4). At this point, we remind the reader that in [65] the problem of making abstract domains complete as regards a given function, is solved and allows us to modify $\mathcal{H}$ in order to make the equation above to hold for the given function $[\![P]\!]$, which corresponds to characterizing a new abstract domain $\mathcal{H}^{\#}$ which makes the program P satisfy non-interference. In particular, the idea is to transform $\mathcal{H}$, either refining or simplifying it, in order to get completeness in the equation above, and, therefore, abstract non-interference. Note that

$$\mathcal{H} = \lambda X.\langle \top(X^{\mathtt{H}}), id(X^{\mathtt{L}})\rangle = \lambda X.\ \langle \mathbb{V}^{\mathtt{H}}, X^{\mathtt{L}}\rangle$$

where $X^{\mathtt{H}} \stackrel{\text{def}}{=} \left\{\ h \,\middle|\, \langle l, h\rangle \in X\ \right\}$, i.e., $\mathcal{H}$ is the product of respectively the top abstraction on private information, and the bottom abstraction on public one, in the lattice of abstract interpretations. This means that the private component of $\mathcal{H}$ can only be refined as well as its public component can only be abstracted. Recall that, given an abstract domain A, that we want to make complete for f, the backward completeness core looks for the most concrete abstract domain contained in A and which is complete for f, while the complete shell looks for the most abstract domain that contains A and which is complete for f. Therefore, we can interpret completeness cores and shells in the following way:

- The core, which can only abstract the public component, characterizes the most concrete attacker that cannot disclose private properties, i.e., the most concrete *public observer*;
- The shell, which can only refine the private component, characterizes the most abstract property that flows, i.e., the most abstract *private observable*.

Formally, completeness cores and shells of an abstract domain ρ, relatively to a monotone function f, are defined in the following way (see Sect. 2.2.4):

$$\mathcal{R}^{\mathcal{B}}_f(\rho) = gfp^{\sqsubseteq}_{\rho}\lambda\varphi.\rho \sqcap R^{\mathcal{B}}_f(\varphi) \qquad \mathcal{C}^{\mathcal{B}}_f(\rho) = lfp^{\sqsubseteq}_{\rho}\lambda\varphi.\rho \sqcup C^{\mathcal{B}}_f(\varphi)$$

where

$$R^{\mathcal{B}}_f \stackrel{\text{def}}{=} \lambda\rho.\mathcal{M}(\textstyle\bigcup_{y\in\rho}\max(f^{-1}(\downarrow y))) \qquad C^{\mathcal{B}}_f \stackrel{\text{def}}{=} \lambda\rho.\left\{\ y \in C \,\middle|\, \max(f^{-1}(\downarrow y)) \subseteq \rho\ \right\}$$

The following examples show how we can apply these transformers in the context of abstract non-interference, and how we can interpret the results obtained.

Example 8.1. Consider the program fragment:

$$P \stackrel{\text{def}}{=} l := 2 * h$$

where $l : \mathtt{L}$ and $h : \mathtt{H}$. It is worth noting that P violates non-interference, since for example

$$\mathcal{H} \circ [\![P]\!] \circ \mathcal{H}(\langle 2, 3\rangle) = \mathcal{H} \circ [\![P]\!](\langle \mathbb{Z}, 3\rangle) = \mathcal{H}(\langle \mathbb{Z}, 2\mathbb{Z}\rangle) = \langle \mathbb{Z}, 2\mathbb{Z}\rangle \text{ while}$$
$$\mathcal{H} \circ [\![P]\!](\langle 2, 3\rangle) = \mathcal{H}(\langle 2, 4\rangle) = \langle \mathbb{Z}, 4\rangle$$

where $2\mathbb{Z} \neq 4$. We can derive the complete core of $\mathcal{H}$ for $[\![P]\!]$, which makes the program secure. From [65] we have to keep only those elements whose inverse image is a fixpoint of $\mathcal{H}$:

$$\begin{aligned}
\mathcal{C}^{\mathcal{B}}_{[\![P]\!]}(\mathcal{H}) &= \mathcal{H} \sqcup \left\{ X' \,\middle|\, \max\left\{ X \,\middle|\, [\![P]\!](X) \subseteq X' \right\} \subseteq \mathcal{H} \right\} \\
&= \left\{ \langle \mathbb{Z}, L\rangle \,\middle|\, \left\{ \langle h, l\rangle \,\middle|\, [\![P]\!](\langle h, l\rangle) \subseteq \langle \mathbb{Z}, L\rangle \right\} \subseteq \mathcal{H} \right\} \\
&= \left\{ \langle \mathbb{Z}, L\rangle \,\middle|\, \left\{ \langle h, l\rangle \,\middle|\, \langle h, 2h\rangle \subseteq \langle \mathbb{Z}, L\rangle \right\} \subseteq \mathcal{H} \right\}
\end{aligned}$$

Note that $\langle H, L\rangle \in \mathcal{H}$ iff $H = \mathbb{Z}$ and $\langle \mathbb{Z}, L'\rangle \supseteq \langle \mathbb{Z}, 2\mathbb{Z}\rangle$ iff $L' \supseteq 2\mathbb{Z}$. In general, if we have that $L' \subseteq 2\mathbb{Z} + 1$ then $\langle h, 2h\rangle \in \langle \mathbb{Z}, L'\rangle$ is false for each possible L', namely

$$\left\{ \langle h, l\rangle \,\middle|\, [\![P]\!](\langle h, l\rangle) \subseteq \langle H', L'\rangle \right\} = \varnothing \subseteq \mathcal{H}$$

which means that in this case $\langle \mathbb{Z}, L'\rangle$ is kept. Therefore, we have that the complete core is the transformer

$$\mathcal{C}^{\mathcal{B}}_{[\![P]\!]}(\mathcal{H}) = \left\{ \langle \mathbb{Z}, L\rangle \,\middle|\, L \cap 2\mathbb{Z} \in \{2\mathbb{Z}, \varnothing\} \right\}$$

which corresponds to abstracting the public output in the domain that is not able to distinguish even numbers, but only the odd ones. Let $\overline{\mathcal{H}} \stackrel{\text{def}}{=} \mathcal{C}^{\mathcal{B}}_{[\![P]\!]}(\mathcal{H})$, then we have for example

$$\overline{\mathcal{H}} \circ [\![P]\!] \circ \overline{\mathcal{H}}(\langle 2, 3\rangle) = \overline{\mathcal{H}} \circ [\![P]\!](\langle \mathbb{Z}, 3\rangle) = \overline{\mathcal{H}}(\langle \mathbb{Z}, 2\mathbb{Z}\rangle) = \langle \mathbb{Z}, 2\mathbb{Z}\rangle \text{ and}$$
$$\overline{\mathcal{H}} \circ [\![P]\!](\langle 2, 3\rangle) = \overline{\mathcal{H}}(\langle 2, 4\rangle) = \langle \mathbb{Z}, 2\mathbb{Z}\rangle$$

Example 8.2. Consider the program fragment

$$P \stackrel{\text{def}}{=} l := (2h + 1) \ mod \ 2$$

where $l : \mathtt{L}$ and $h : \mathtt{H}$. The program violates non-interference, since, for instance,

$$\mathcal{H} \circ [\![P]\!] \circ \mathcal{H}(\langle 2, 3\rangle) = \mathcal{H} \circ [\![P]\!](\langle \mathbb{Z}, 3\rangle) = \mathcal{H}(\langle \mathbb{Z}, \{-1, 1\}\rangle) = \langle \mathbb{Z}, \{-1, 1\}\rangle \text{ while}$$
$$\mathcal{H} \circ [\![P]\!](\langle 2, 3\rangle) = \mathcal{H}(\langle 2, 1\rangle) = \langle \mathbb{Z}, 1\rangle$$

and $\{-1, 1\} \neq 1$. We compute the complete shell of $\mathcal{H}$, characterizing the flowing property of private information, namely we add all the inverse images of the elements in $\mathcal{H}$.

$$\begin{aligned}
\mathcal{R}^{\mathcal{B}}_{[\![P]\!]}(\mathcal{H}) &= \mathcal{H} \sqcap \mathcal{M}(\textstyle\bigcup_{L' \in \wp(\mathbb{V}^{\mathtt{L}})} \max\left\{ X \,\middle|\, [\![P]\!](X) \subseteq \langle \mathbb{Z}, L'\rangle \right\}) \\
&= \mathcal{H} \sqcap \mathcal{M}(\textstyle\bigcup_{L' \in \wp(\mathbb{V}^{\mathtt{L}})} \left\{ \langle h, l\rangle \,\middle|\, \langle h, 2h + 1 \ mod \ 2\rangle \in \langle \mathbb{Z}, L'\rangle \right\})
\end{aligned}$$

At this point note that, if $-1 \notin L'$, then

$$\left\{ \langle h, l\rangle \,\middle|\, \langle h, 2h+1 \ mod\ 2\rangle \in \langle \mathbb{Z}, L'\rangle \right\} = \langle \mathbb{Z}_0^+, \mathbb{Z}\rangle$$

where $\mathbb{Z}_0^+ \stackrel{\text{def}}{=} \mathbb{Z}^+ \cup \{0\}$. While, if $1 \notin L'$, then we have

$$\left\{ \langle h, l\rangle \,\middle|\, \langle h, 2h+1 \ mod\ 2\rangle \in \langle \mathbb{Z}, L'\rangle \right\} = \langle \mathbb{Z}^-, \mathbb{Z}\rangle$$

Finally, if $1, -1 \notin L'$, then we obtain

$$\left\{ \langle h, l\rangle \,\middle|\, \langle h, 2h+1 \ mod\ 2\rangle \in \langle \mathbb{Z}, L'\rangle \right\} = \varnothing$$

Hence $\bigcup_{L' \in \wp(\mathbb{V}^{\mathtt{L}})} \left\{ \langle h, l\rangle \,\middle|\, \langle h, 2h+1 \ mod\ 2\rangle \in \langle \mathbb{Z}, L'\rangle \right\} = \{\langle \mathbb{Z}_0^+, \mathbb{Z}\rangle, \langle \mathbb{Z}^-, \mathbb{Z}\rangle, \varnothing\}$, and

$$\mathcal{R}^{\mathcal{B}}_{[\![P]\!]}(\mathcal{H}) = \mathcal{H} \cup \left\{ \langle H, L\rangle \,\middle|\, H \in \{\mathbb{Z}_0^+, \mathbb{Z}^-\},\ L \in \wp(\mathbb{V}^{\mathtt{L}}) \right\}$$

At this point, let $\overline{\mathcal{H}} \stackrel{\text{def}}{=} \mathcal{R}^{\mathcal{B}}_{[\![P]\!]}(\mathcal{H})$, then for example, we have

$$\overline{\mathcal{H}} \circ [\![P]\!] \circ \overline{\mathcal{H}}(\langle 2, 3\rangle) = \overline{\mathcal{H}} \circ [\![P]\!](\langle \mathbb{Z}_0^+, 3\rangle) = \overline{\mathcal{H}}(\langle \mathbb{Z}_0^+, \{1\}\rangle) = \langle \mathbb{Z}_0^+, \{1\}\rangle \text{ and}$$
$$\overline{\mathcal{H}} \circ [\![P]\!](\langle 2, 3\rangle) = \overline{\mathcal{H}}(\langle 2, 1\rangle) = \langle \mathbb{Z}_0^+, \{1\}\rangle$$

At this point, we would like to model abstract non-interference by considering this completeness characterization of non-interference. The idea is to embed the model of an attacker, described, in abstract non-interference, by a pair of input/output abstractions, inside the closure operator $\mathcal{H}$. Consider the domain

$$\langle \wp(\mathbb{V}^{\mathtt{H}}) \times \wp(\mathbb{V}^{\mathtt{L}}), \varnothing, \langle \mathbb{V}^{\mathtt{H}}, \mathbb{V}^{\mathtt{L}}\rangle, \Cup, \cap, \subseteq\rangle$$

where $\langle H_1, L_1\rangle \Cup \langle H_2, L_2\rangle \stackrel{\text{def}}{=} \langle H_1 \cup H_2, L_1 \cup L_2\rangle$. It is well known that there exists an obvious Galois insertion from $\wp(\mathbb{V}^{\mathtt{H}}) \times \wp(\mathbb{V}^{\mathtt{L}})$ in $\wp(\mathbb{V}^{\mathtt{H}} \times \mathbb{V}^{\mathtt{L}})$, corresponding to the closure:

$$\mathit{Split} \stackrel{\text{def}}{=} \lambda X.\ \left\{ \langle x_1, x_2\rangle \,\middle|\, \exists y.\, \langle x_1, y\rangle \in X,\ \exists z.\, \langle z, x_2\rangle \in X \right\}$$

Given a closure $\rho \in \mathit{uco}(\wp(\mathbb{V}^{\mathtt{L}}))$, let us define $\mathcal{H}_\rho \in \mathit{uco}(\wp(\mathbb{V}^{\mathtt{H}}) \times \wp(\mathbb{V}^{\mathtt{L}}))$

$$\mathcal{H}_\rho \stackrel{\text{def}}{=} \lambda X \in \wp(\mathbb{V}^{\mathtt{H}} \times \mathbb{V}^{\mathtt{L}}).\, \langle \mathbb{V}^{\mathtt{H}}, \rho(X^{\mathtt{L}})\rangle$$

Then we have that $\mathcal{H}_\rho \in \mathit{uco}(\wp(\mathbb{V}^{\mathtt{H}} \times \mathbb{V}^{\mathtt{L}}))$, and that $\mathcal{H} = \mathcal{H}_{\mathit{id}}$. At this point, we can define abstract non-interference by using the completeness equation, in the following way:

$$\mathcal{H}_\rho \circ [\![P]\!] \circ \mathcal{H}_\eta = \mathcal{H}_\rho \circ [\![P]\!]$$

where $\eta, \rho \in \mathit{uco}(\wp(\mathbb{V}^{\mathtt{L}}))$ and $\langle h, l\rangle \in \mathbb{V}$. The following results prove that this characterization provide a notion of narrow abstract non-interference which is, in general, stronger than the notion introduced in the previous chapters. Anyway, under some restrictions on the attacker model, we have the equivalence of these two characterizations of narrow non-interference.

Theorem 8.3. *Let $\rho, \eta \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$.*

1. $[\eta]P\ (\rho) \ \Leftarrow\ \mathcal{H}_\rho \circ [\![P]\!] \circ \mathcal{H}_\eta = \mathcal{H}_\rho \circ [\![P]\!]$;
2. *If ρ is disjunctive and η is partitioning:* $[\eta]P\ (\rho) \ \Rightarrow\ \mathcal{H}_\rho \circ [\![P]\!] \circ \mathcal{H}_\eta = \mathcal{H}_\rho \circ [\![P]\!]$.

Proof. 1. Let us see what means completeness in the context of non-interference. Indeed,

$$\mathcal{H}_\rho \circ [\![P]\!] \circ \mathcal{H}_\eta(\langle h, l\rangle) = \langle \mathbb{V}^{\mathtt{H}}, \rho([\![P]\!](\mathbb{V}^{\mathtt{H}}, \eta(l))^{\mathtt{L}})\rangle$$

while, on the other hand,

$$\mathcal{H}_\rho \circ [\![P]\!](\langle h, l\rangle) = \langle \mathbb{V}^{\mathtt{H}}, \rho([\![P]\!](h, l)^{\mathtt{L}})\rangle$$

Hence, the equality becomes $\rho([\![P]\!](\langle \mathbb{V}^{\mathtt{H}}, \eta(l)\rangle)^{\mathtt{L}}) = \rho([\![P]\!](\langle h, l\rangle)^{\mathtt{L}})$, which has to hold for each $l \in \mathbb{V}^{\mathtt{L}}$ and $h \in \mathbb{V}^{\mathtt{H}}$. Consider $l, l' \in \mathbb{V}^{\mathtt{L}}$ and $h, h' \in \mathbb{V}^{\mathtt{H}}$, then we have the following implications, which corresponds to narrow abstract non-interference:

$$\begin{aligned}\eta(l) = \eta(l') \ \Rightarrow\ & \rho([\![P]\!](\langle h, l\rangle)^{\mathtt{L}}) = \rho([\![P]\!](\mathbb{V}^{\mathtt{H}}, \eta(l))^{\mathtt{L}}) \\ & = \rho([\![P]\!](\mathbb{V}^{\mathtt{H}}, \eta(l'))^{\mathtt{L}}) = \rho([\![P]\!](\langle h', l'\rangle)^{\mathtt{L}})\end{aligned}$$

2. By definition of $[\![P]\!]$, $[\![P]\!](\langle \mathbb{V}^{\mathtt{H}}, \eta(l)\rangle) = \bigcup_{h \in \mathbb{V}^{\mathtt{H}}, l' \in \eta(l)} [\![P]\!](\langle h, l'\rangle)$. Since η is partitioning, for each $l' \in \eta(l)$, we have that $\eta(l') = \eta(l)$, therefore by the hypothesis of non-interference, $\forall h' \in \mathbb{V}^{\mathtt{H}}, \forall l' \in \eta(l)$ we have that $\rho([\![P]\!](\langle h', l'\rangle)^{\mathtt{L}}) = \rho([\![P]\!](\langle h, l\rangle)^{\mathtt{L}})$. Hence, by additivity of ρ, for each $l' \in \eta(l)$, and for each $h \in \mathbb{V}^{\mathtt{H}}$

$$\rho([\![P]\!](\langle \mathbb{V}^{\mathtt{H}}, \eta(l)\rangle)^{\mathtt{L}}) = \rho([\![P]\!](\langle h, l'\rangle)^{\mathtt{L}})$$

At this point, consider $X \in \wp(\mathbb{V}^{\mathtt{H}} \times \mathbb{V}^{\mathtt{L}})$, then for what we have just proved, we have that for each $\langle h, l\rangle \in X$, $\rho([\![P]\!](\langle \mathbb{V}^{\mathtt{H}}, \eta(l)\rangle)^{\mathtt{L}}) = \rho([\![P]\!](\langle h, l\rangle)^{\mathtt{L}})$, therefore

$$\bigcup_{\langle h,l\rangle \in X} \rho([\![P]\!](\langle \mathbb{V}^{\mathtt{H}}, \eta(l)\rangle)^{\mathtt{L}}) = \bigcup_{\langle h,l\rangle \in X} \rho([\![P]\!](\langle h, l\rangle)^{\mathtt{L}})$$

By additivity of ρ (and of $[\![P]\!]$) this corresponds to $\rho([\![P]\!](\langle \mathbb{V}^{\mathtt{H}}, \eta(X^{\mathtt{L}})\rangle)^{\mathtt{L}}) = \rho([\![P]\!](X)^{\mathtt{L}})$. On the other hand, note that $\mathcal{H}_\rho \circ [\![P]\!](X) = \langle \mathbb{V}^{\mathtt{H}}, \rho([\![P]\!](X)^{\mathtt{L}})\rangle$, and

$$\mathcal{H}_\rho \circ [\![P]\!] \circ \mathcal{H}_\eta(X) = \mathcal{H}_\rho \circ [\![P]\!](\langle \mathbb{V}^{\mathtt{H}}, \eta(X^{\mathtt{L}})\rangle) = \langle \mathbb{V}^{\mathtt{H}}, \rho([\![P]\!](\langle \mathbb{V}^{\mathtt{H}}, \eta(X^{\mathtt{L}})\rangle)^{\mathtt{L}})\rangle$$

For what we proved above, these tuples are the same, and therefore we have completeness.

In order to extend Theorem 8.3 to model abstract non-interference we have to modify the program semantics. The idea is to consider an abstract semantics that is applied to abstract (public and private) data. Consider $\eta \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$ and the private property $\phi \in uco(\wp(\mathbb{V}^{\mathtt{H}}))$. We define the abstract semantics as $[\![P]\!]^{\eta,\phi} \stackrel{\text{def}}{=} \lambda\langle h, l\rangle.\ [\![P]\!](\phi(h), \eta(l))$, and therefore, we can define abstract non interference as follows

$$\mathcal{H}_\rho \circ [\![P]\!]^{\eta,\phi} \circ \mathcal{H}_\eta = \mathcal{H}_\rho \circ [\![P]\!]^{\eta,\phi}$$

where $\eta, \rho \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$ and $\langle h, l\rangle \in \mathbb{V}$. At this point, we can prove that this characterization of abstract non-interference is, in general, stronger than the notion introduced in the previous chapters. Anyway, also in this case, under some restrictions on the attacker model, we can prove the equivalence of the two abstract non-interference characterizations.

Theorem 8.4. *Consider $\eta, \rho \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$ and $\phi \in uco(\wp(\mathbb{V}^{\mathtt{H}}))$:*

1. $(\rho)P\ (\phi \rightsquigarrow \eta) \Leftarrow \mathcal{H}_\rho \circ [\![P]\!]^{\eta,\phi} \circ \mathcal{H}_\eta = \mathcal{H}_\rho \circ [\![P]\!]^{\eta,\phi}$;
2. *Consider ρ and η disjunctive:* $(\eta)P\ (\phi \rightsquigarrow \rho) \Rightarrow \mathcal{H}_\rho \circ [\![P]\!]^{\eta,\phi} \circ \mathcal{H}_\eta = \mathcal{H}_\rho \circ [\![P]\!]^{\eta,\phi}$.

Proof. 1. Note that $\mathcal{H}_\rho \circ [\![P]\!]^{\eta,\phi} \circ \mathcal{H}_\eta(\langle h, l\rangle) = \mathcal{H}_\rho \circ [\![P]\!]^{\eta,\phi}(\langle \mathbb{V}^{\mathtt{H}}, \eta(l)\rangle)$, since we have $\phi(\mathbb{V}^{\mathtt{H}}) = \mathbb{V}^{\mathtt{H}}$ and η is idempotent. Therefore,

$$\mathcal{H}_\rho \circ [\![P]\!](\mathbb{V}^{\mathtt{H}}, \eta(l)) = \langle \mathbb{V}^{\mathtt{H}}, \rho([\![P]\!](\mathbb{V}^{\mathtt{H}}, \eta(l))^{\mathtt{L}})\rangle$$

On the other hand,

$$\mathcal{H}_\rho \circ [\![P]\!]^{\eta,\phi}(\langle h, l\rangle) = \mathcal{H}_\rho \circ [\![P]\!](\langle \phi(h), \eta(l)\rangle) = \langle \mathbb{V}^{\mathtt{H}}, \rho([\![P]\!](\langle \phi(h), \eta(l)\rangle)^{\mathtt{L}})\rangle$$

Hence, we have $\rho([\![P]\!](\mathbb{V}^{\mathtt{H}}, \eta(l))^{\mathtt{L}}) = \rho([\![P]\!](\langle \phi(h), \eta(l)\rangle)^{\mathtt{L}})$, for each possible input and therefore for each possible input we have the same result.

2. The implication can be proved similarly to Theorem 8.3. In particular, by additivity of $[\![P]\!]$ on $\wp(\mathbb{V}^{\mathtt{H}} \times \mathbb{V}^{\mathtt{L}})$, $[\![P]\!](\langle \mathbb{V}^{\mathtt{H}}, \eta(l)\rangle) = \bigcup_{h \in \mathbb{V}^{\mathtt{H}}} [\![P]\!](\langle \phi(h), \eta(l)\rangle)$. By hypothesis of non-interference we have that $\forall h' \in \mathbb{V}^{\mathtt{H}}\,.\, \rho([\![P]\!](\langle h', \eta(l)\rangle)^{\mathtt{L}}) = \rho([\![P]\!](\langle h, \eta(l)\rangle)^{\mathtt{L}})$, hence the additivity of ρ implies that, for each $h \in \mathbb{V}^{\mathtt{H}}$, and for each $l \in \mathbb{V}^{\mathtt{L}}$

$$\rho([\![P]\!](\langle \mathbb{V}^{\mathtt{H}}, \eta(l)\rangle)^{\mathtt{L}}) = \rho([\![P]\!](\langle \phi(h), \eta(l)\rangle)^{\mathtt{L}})$$

Let $X \in \wp(\mathbb{V}^{\mathtt{H}} \times \mathbb{V}^{\mathtt{L}})$, what we have just proved implies that for each $\langle h, l\rangle \in X$ we have $\rho([\![P]\!](\langle \mathbb{V}^{\mathtt{H}}, \eta(l)\rangle)^{\mathtt{L}}) = \rho([\![P]\!](\langle \phi(h), \eta(l)\rangle)^{\mathtt{L}})$. Therefore we obtain that

$$\bigcup_{\langle h,l\rangle \in X} \rho([\![P]\!](\langle \mathbb{V}^{\mathtt{H}}, \eta(l)\rangle)^{\mathtt{L}}) = \bigcup_{\langle h,l\rangle \in X} \rho([\![P]\!](\langle \phi(h), \eta(l)\rangle)^{\mathtt{L}})$$

which implies $\rho([\![P]\!]^{\eta,\phi}(\langle \mathbb{V}^{\mathtt{H}}, \eta(X^{\mathtt{L}})\rangle)^{\mathtt{L}}) = \rho([\![P]\!]^{\eta,\phi}(X)^{\mathtt{L}})$ by additivity of ρ and of η. On the other hand, note that $\mathcal{H}_\rho \circ [\![P]\!]^{\eta,\phi}(X) = \langle \mathbb{V}^{\mathtt{H}}, \rho([\![P]\!]^{\eta,\phi}(X)^{\mathtt{L}})\rangle$. While we also have

$$\begin{aligned} \mathcal{H}_\rho \circ [\![P]\!]^{\eta,\phi} \circ \mathcal{H}_\eta(X) &= \mathcal{H}_\rho \circ [\![P]\!]^{\eta,\phi}(\langle \mathbb{V}^{\mathtt{H}}, \eta(X^{\mathtt{L}})\rangle) \\ &= \langle \mathbb{V}^{\mathtt{H}}, \rho([\![P]\!]^{\eta,\phi}(\langle \mathbb{V}^{\mathtt{H}}, \eta(X^{\mathtt{L}})\rangle)^{\mathtt{L}})\rangle. \end{aligned}$$

And these two tuples are equal for what we proved above.

In the following, without loss of generality we consider the abstract non-interference case being more general.

8.2 The most concrete *observer* as completeness core

In Sect 6.3, we gave a method for systematically deriving the most concrete harmless attacker associated with a given program, namely the most concrete attacker that cannot disclose any confidential property. By Theorem 8.4, the most concrete public observer can be derived as the most concrete abstraction satisfying the following completeness problem:

$$\mathcal{H} \circ [\![P]\!]^{\eta,\phi} \circ \mathcal{H}_\eta = \mathcal{H} \circ [\![P]\!]^{\eta,\phi} \tag{8.1}$$

Then, we have the following result which allows us to specify the most concrete harmless attacker as the fixpoint of an abstract domain simplification. In the following, we omit the apex $\mathcal{B}$ from shells and cores, since we will consider always backward completeness.

Theorem 8.5. *Let $\eta \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$ be disjunctive and $\phi \in uco(\mathbb{V}^{\mathtt{H}})$. Then we have* $\mathcal{C}^{\mathcal{H}_\eta}_{[\![P]\!]^{\eta,\phi}}(\mathcal{H}) = \left\{ \langle \mathbb{V}^{\mathtt{H}}, L\rangle \,\middle|\, \left\{ \langle h, l\rangle \,\middle|\, [\![P]\!](\langle \phi(h), \eta(l)\rangle) \subseteq \langle \mathbb{V}^{\mathtt{H}}, L\rangle \right\} \in \mathcal{H}_\eta \right\}$ *and*

$$\left\{ L \in \wp(\mathbb{V}^{\mathtt{L}}) \,\middle|\, \langle \mathbb{V}^{\mathtt{H}}, L\rangle \in \mathcal{C}^{\mathcal{H}_\eta}_{[\![P]\!]^{\eta,\phi}}(\mathcal{H}) \right\} = (\eta)[\![P]\!]\ (\phi \rightsquigarrow id).$$

Proof. We remind the reader that, in Sect. 6.3 the domain $(\eta)[\![P]\!]\ (\phi \rightsquigarrow id)$ is characterized as the set $\left\{ X \in \wp(\mathbb{V}^{\mathtt{L}}) \,\middle|\, Secr^{\eta}_{[\![P]\!]}(X) \right\}$, where $Secr^{\eta}_{[\![P]\!]}(X)$ if and only if, for each $l \in \mathbb{V}^{\mathtt{L}}$ we have $(\exists Z \in \Upsilon^{\eta,\phi}_{[\![P]\!]}(l).\, Z \subseteq X \ \Rightarrow\ \forall W \in \Upsilon^{\eta,\phi}_{[\![P]\!]}(l).\, W \subseteq X)$, and the set of indistinguishable elements is $\Upsilon^{\eta,\phi}_{[\![P]\!]}(l) \stackrel{\text{def}}{=} \left\{ [\![P]\!](\langle \phi(h), \eta(l)\rangle) \,\middle|\, h \in \mathbb{V}^{\mathtt{H}} \right\}$ (Th. 6.12). Therefore, we have to prove that all the elements $\langle \mathbb{V}^{\mathtt{H}}, L\rangle$ in the core are such that L is secret, and that all the secret elements L are such that $\langle \mathbb{V}^{\mathtt{H}}, L\rangle$ is in the core. Consider $X \in \wp(\mathbb{V}^{\mathtt{H}} \times \mathbb{V}^{\mathtt{L}})$.

$$\begin{aligned}
\mathcal{C}^{\mathcal{H}_\eta}_{[\![P]\!]^{\eta,\phi}}(\mathcal{H}) &= \mathcal{H}\sqcup \\
&\quad \left\{ X \,\middle|\, \max\left\{ X' \in \wp(\mathbb{V}^{\mathtt{H}} \times \mathbb{V}^{\mathtt{L}}) \,\middle|\, [\![P]\!]^{\phi,\eta}(X') \subseteq X \right\} \subseteq \mathcal{H}_\eta \right\} \\
&= \left\{ \langle \mathbb{V}^{\mathtt{H}}, L\rangle \,\middle|\, \left\{ \langle h, l\rangle \,\middle|\, [\![P]\!]^{\phi,\eta}(\langle h, l\rangle) \subseteq \langle \mathbb{V}^{\mathtt{H}}, L\rangle \right\} \in \mathcal{H}_\eta \right\} \qquad (*) \\
&= \left\{ \langle \mathbb{V}^{\mathtt{H}}, L\rangle \,\middle|\, \left\{ \langle h, l\rangle \,\middle|\, [\![P]\!](\langle \phi(h), \eta(l)\rangle) \subseteq \langle \mathbb{V}^{\mathtt{H}}, L\rangle \right\} \in \mathcal{H}_\eta \right\}
\end{aligned}$$

where the equality $(*)$ holds since $[\![P]\!]^{\phi,\eta}$ is additive (being an additive lift) on $\wp(\mathbb{V}^{\mathtt{H}} \times \mathbb{V}^{\mathtt{L}})$. Now we have to prove that a set $L \in \wp(\mathbb{V}^{\mathtt{L}})$ is secret, i.e., $Secr^{\eta}_{[\![P]\!]}(L)$, if and only if $\langle \mathbb{V}^{\mathtt{H}}, L\rangle \in \mathcal{C}^{\mathcal{H}_\eta}_{[\![P]\!]^{\eta,\phi}}(\mathcal{H})$. Consider $\langle \mathbb{V}^{\mathtt{H}}, L\rangle \in \mathcal{C}^{\mathcal{H}_\eta}_{[\![P]\!]^{\eta,\phi}}(\mathcal{H})$, i.e., L is such that there exists $L' \in \wp(\mathbb{V}^{\mathtt{L}})$ such that $\left\{ \langle h, l\rangle \,\middle|\, [\![P]\!](\phi(h), \eta(l))^{\mathtt{L}} \subseteq L \right\} = \langle \mathbb{V}^{\mathtt{H}}, \eta(L')\rangle$. We have to prove that $\forall l \in \mathbb{V}^{\mathtt{L}} \colon \forall h \in \mathbb{V}^{\mathtt{H}}\ .\, [\![P]\!](\phi(h), \eta(l))^{\mathtt{L}} \subseteq L$ or we prove that $\forall h \in \mathbb{V}^{\mathtt{H}}\ .\, [\![P]\!](\phi(h), \eta(l))^{\mathtt{L}} \not\subseteq L$. Let $l \in \eta(L')$, then $\forall h \in \mathbb{V}^{\mathtt{H}}\ .\, [\![P]\!](\phi(h), \eta(l))^{\mathtt{L}} \subseteq L$, since otherwise there exists $h' \in \mathbb{V}^{\mathtt{H}}$ such that $[\![P]\!](\phi(h'), \eta(l))^{\mathtt{L}} \not\subseteq L$, which means that $\langle h', l\rangle \in \langle \mathbb{V}^{\mathtt{H}}, \eta(L')\rangle$, but also that $\langle h', l\rangle \notin \left\{ \langle h, l\rangle \,\middle|\, [\![P]\!](\phi(h), \eta(l))^{\mathtt{L}} \subseteq L \right\}$, which is a contradiction. Consider now $l \notin \eta(L')$. Then $\forall h \in \mathbb{V}^{\mathtt{H}}\ .\, [\![P]\!](\phi(h), \eta(l))^{\mathtt{L}} \not\subseteq L$. Indeed, if $\exists h' \in \mathbb{V}^{\mathtt{H}}.\ [\![P]\!](\phi(h'), \eta(l))^{\mathtt{L}} \subseteq L$, then $\langle h', l\rangle \notin \langle \mathbb{V}^{\mathtt{H}}, \eta(L')\rangle$, but

$\langle h', l\rangle \in \{\ \langle h, l\rangle \mid [\![P]\!](\phi(h), \eta(l))^{\mathtt{L}} \subseteq L\ \}$, which is again a contradiction.
Consider now L secret, then we have $\forall l \in \mathbb{V}^{\mathtt{L}} : \forall h \in \mathbb{V}^{\mathtt{H}} \,.\, [\![P]\!](\phi(h), \eta(l))^{\mathtt{L}} \subseteq L$ or $\forall h \in \mathbb{V}^{\mathtt{H}} \,.\, [\![P]\!](\phi(h), \eta(l))^{\mathtt{L}} \not\subseteq L$. We have to prove that there exists $L' \in \wp(\mathbb{V}^{\mathtt{L}})$ such that

$$\{\ \langle h, l\rangle \mid [\![P]\!](\phi(h), \eta(l))^{\mathtt{L}} \subseteq L\ \} = \langle \mathbb{V}^{\mathtt{H}}, \eta(L')\rangle$$

Let us define the set $L' \stackrel{\text{def}}{=} \{\ l \in \mathbb{V}^{\mathtt{L}} \mid \forall h \in \mathbb{V}^{\mathtt{H}} \,.\, [\![P]\!](\phi(h), \eta(l))^{\mathtt{L}} \subseteq L\ \}$, we prove that this L' is the needed set. Let $\langle h, l\rangle$ such that $[\![P]\!](\phi(h), \eta(l))^{\mathtt{L}} \subseteq L$, then by secrecy of L we have that $\forall h' \in \mathbb{V}^{\mathtt{H}} \,.\, [\![P]\!](\phi(h'), \eta(l))^{\mathtt{L}} \subseteq L$, therefore, by definition of L' and by extensivity of η, we have $l \in L' \subseteq \eta(L')$. So $\langle h, l\rangle \in \langle \mathbb{V}^{\mathtt{H}}, \eta(L')\rangle$. Consider, now, $\langle h, l\rangle \in \langle \mathbb{V}^{\mathtt{H}}, \eta(L')\rangle$, namely $l \in \eta(L')$. If $l \in L'$, then by definition we have $\forall h \in \mathbb{V}^{\mathtt{H}} \,.\, [\![P]\!](\phi(h), \eta(l))^{\mathtt{L}} \subseteq L$.
At this point note that $\eta(L') = \eta(\{\ l \in \mathbb{V}^{\mathtt{L}} \mid \forall h \in \mathbb{V}^{\mathtt{H}} \,.\, [\![P]\!](\phi(h), \eta(l))^{\mathtt{L}} \subseteq L\ \})$, let us prove that $\eta(L') = L'$. By extensivity we have $\eta(L') \supseteq L'$, suppose, towards a contradiction, that there exists $l' \in \eta(L')$ such that $l' \notin L'$. Then by definition of L', $\exists h' \in \mathbb{V}^{\mathtt{H}}$ such that $[\![P]\!](\phi(h'), \eta(l'))^{\mathtt{L}} \not\subseteq L$. Since η is an additive closure, $l' \in \eta(L')$ implies that $\exists l'' \in L'$ such that $l' \in \eta(l'')$, which means that $\eta(l') \subseteq \eta(l'')$. So, if $[\![P]\!](\phi(h'), \eta(l''))^{\mathtt{L}} \subseteq L$, then also $[\![P]\!](\phi(h'), \eta(l'))^{\mathtt{L}} \subseteq L$, being $[\![P]\!](\phi(h'), \eta(l'))^{\mathtt{L}} \subseteq [\![P]\!](\phi(h'), \eta(l''))^{\mathtt{L}}$. But this fact contradicts the hypothesis made, therefore $[\![P]\!](\phi(h'), \eta(l'))^{\mathtt{L}} \not\subseteq L$, which is again a contradiction since $l'' \in L'$. This means that $l' \in \eta(L')$ iff $l' \in L'$. Therefore, if $\langle h, l\rangle \in \langle \mathbb{V}^{\mathtt{H}}, \eta(L')\rangle$ then $[\![P]\!](\phi(h), \eta(l))^{\mathtt{L}} \subseteq L$. Hence, $\{\ \langle h, l\rangle \mid [\![P]\!](\phi(h), \eta(l))^{\mathtt{L}} \subseteq L\ \} = \langle \mathbb{V}^{\mathtt{H}}, \eta(L')\rangle$, namely $\langle \mathbb{V}^{\mathtt{H}}, L\rangle \in \mathcal{C}^{\mathcal{H}_\eta}_{[\![P]\!]^{\eta,\phi}}(\mathcal{H})$.

The following example shows how we can use this completeness characterization of abstract non-interference, in order to derive the most concrete public observer, which corresponds to the most concrete harmless attacker.

Example 8.6. Consider the following program fragment, with $l : \mathtt{L}$ and $h : \mathtt{H}$.

$$P \stackrel{\text{def}}{=} \textbf{while } h \textbf{ do } l := 2l;\ h := 0 \textbf{ endw} \qquad [\![P]\!](\langle h, l\rangle) = \begin{cases} \langle h, l\rangle & \text{if } h = 0 \\ \langle h, 2l\rangle & \text{otherwise} \end{cases}$$

We look for the core in order to make $\langle \mathcal{H}, \mathcal{H}\rangle$ complete for the map $[\![P]\!]^{id,id} = [\![P]\!]$.

$$\begin{aligned} \mathcal{C}^{\mathcal{H}}_{[\![P]\!]}(\mathcal{H}) &= \mathcal{H} \sqcup \left\{\ \langle H, L\rangle \,\middle|\, \max\{\ \langle H', L'\rangle \mid [\![P]\!](\langle H', L'\rangle) \subseteq \langle H, L\rangle\ \} \subseteq \mathcal{H}\ \right\} \\ &= \left\{\ \langle \mathbb{Z}, L\rangle \,\middle|\, \max\{\ \langle H', L'\rangle \mid [\![P]\!](\langle H', L'\rangle) \subseteq \langle \mathbb{Z}, L\rangle\ \} \subseteq \mathcal{H}\ \right\} \\ &= \left\{\ \langle \mathbb{Z}, L\rangle \,\middle|\, \{\ \langle h, l\rangle \mid \forall h \in \mathbb{V}^{\mathtt{H}} \,.\, [\![P]\!](\langle h, l\rangle)^{\mathtt{L}} \subseteq L\ \}\ \right\} \\ &= \{\ \langle \mathbb{Z}, L\rangle \mid \forall l \in \mathbb{V}^{\mathtt{L}} \,.\, l \in L \ \Leftrightarrow\ 2l \in L\ \} \end{aligned}$$

It is straightforward to show that $\mathcal{C}^{\mathcal{H}}_{[\![P]\!]}(\mathcal{H})$ is the domain that abstracts the public data in the domain $\curlyvee(\{\ n\{2\}^{\mathbb{N}} \mid n \in 2\mathbb{Z}+1\ \})$, where $\{2\}^{\mathbb{N}} \stackrel{\text{def}}{=} \{\ 2^k \mid k \in \mathbb{N}\ \}$. Let $\overline{\mathcal{H}} \stackrel{\text{def}}{=} \mathcal{C}^{\mathcal{H}}_{[\![P]\!]}(\mathcal{H})$, then, for instance, we have that

$$\begin{aligned} &\overline{\mathcal{H}} \circ [\![P]\!] \circ \mathcal{H}(\langle 3, 5\rangle) = \overline{\mathcal{H}} \circ [\![P]\!](\mathbb{Z}, 5) = \overline{\mathcal{H}}(\langle \mathbb{Z}, \{5, 10\}\rangle) = \langle \mathbb{Z}, 5\{2\}^{\mathbb{N}}\rangle \text{ and} \\ &\overline{\mathcal{H}} \circ [\![P]\!](\langle 3, 5\rangle) = \overline{\mathcal{H}}(\langle \mathbb{Z}, \{10\}\rangle) = \langle \mathbb{Z}, 5\{2\}^{\mathbb{N}}\rangle \end{aligned}$$

while we have that

$$\mathcal{H} \circ [\![P]\!] \circ \mathcal{H}(\langle 3,5\rangle) = \mathcal{H} \circ [\![P]\!](\mathbb{Z},5) = \mathcal{H}(\langle \mathbb{Z}, \{5,10\}\rangle) = \langle \mathbb{Z}, \{5,10\}\rangle \text{ and}$$
$$\mathcal{H} \circ [\![P]\!](\langle 3,5\rangle) = \mathcal{H}(\langle \mathbb{Z}, \{10\}\rangle) = \langle \mathbb{Z}, \{10\}\rangle$$

8.3 The most abstract *observable* as completeness shell

We are now interested in applying the same construction for characterizing the most abstract private observable, used for characterizing abstract declassification, as a solution of a completeness problem in abstract interpretation. Namely, we are interested in the most abstract property that has to be declassified in order to guarantee abstract non-interference. By Theorem 8.4, this information can be obtained by solving the following completeness problem:

$$\mathcal{H}_\rho \circ [\![P]\!]^{\eta,id} \circ \mathcal{H}_\eta = \mathcal{H}_\rho \circ [\![P]\!]^{\eta,id} \tag{8.2}$$

Lemma 8.7. *Let $\rho \in uco(D)$, and $f : C \longrightarrow D$. Let $x \in C$ and $A \in \rho \subseteq D$, then $f(x) \subseteq A$ iff $\rho f(x) \subseteq A$.*

Proof. If $f(x) \subseteq A$, then by monotonicity $\rho f(x) \subseteq \rho(A) = A$, since $A \in \rho$. If $\rho f(x) \subseteq A$, then by extensivity of ρ, $f(x) \subseteq \rho f(x) \subseteq A$.

Lemma 8.8. *Let $\rho, \eta \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$. Then we have*

$$\mathcal{R}^{\mathcal{H}_\rho}_{[\![P]\!]^{\eta,id}}(\mathcal{H}_\eta) = \mathcal{H}_\eta \sqcap \mathcal{M}(\{\ \{\ \langle h,l\rangle \mid \rho([\![P]\!](\langle h,\eta(l)\rangle)^{\mathtt{L}}) \subseteq L\ \} \mid L \in \rho\ \}).$$

Moreover, let $\mathcal{R} \stackrel{def}{=} \mathcal{R}^{\mathcal{H}_\rho}_{[\![P]\!]^{\eta,id}}(\mathcal{H}_\eta)$, then for all $l, l' \in \mathbb{V}^{\mathtt{L}}, h, h' \in \mathbb{V}^{\mathtt{H}}$ we have

$$\mathcal{R}(\langle h,l\rangle) = \mathcal{R}(\langle h',l'\rangle) \qquad \textit{iff} \qquad \rho[\![P]\!](\langle h,\eta(l)\rangle)^{\mathtt{L}} = \rho[\![P]\!](\langle h',\eta(l')\rangle)^{\mathtt{L}}$$

Proof. First of all, we want to characterize the complete shell of $\mathcal{H}_\eta$.

$$\begin{aligned}
\mathcal{R}^{\mathcal{H}_\rho}_{[\![P]\!]^{\eta,id}}(\mathcal{H}_\eta) &= \mathcal{H}_\eta \sqcap \mathcal{M}(\textstyle\bigcup_{Y \in \mathcal{H}_\rho} \max\{\ X' \mid [\![P]\!]^{id,\eta}(X') \subseteq Y\ \}) \\
&= \mathcal{H}_\eta \sqcap \mathcal{M}(\textstyle\bigcup_{\langle \mathbb{V}^{\mathtt{H}}, \rho(L)\rangle} \{\ \langle h,l\rangle \mid [\![P]\!](\langle h,\eta(l)\rangle) \subseteq \langle \mathbb{V}^{\mathtt{H}}, \rho(L)\rangle\ \}) \\
&= \mathcal{H}_\eta \sqcap \mathcal{M}(\textstyle\bigcup_{\rho(L)} \{\ \langle h,l\rangle \mid [\![P]\!](\langle h,\eta(l)\rangle)^{\mathtt{L}} \subseteq \rho(L)\ \}) \\
&= \mathcal{H}_\eta \sqcap \mathcal{M}(\{\ \{\ \langle h,l\rangle \mid [\![P]\!](\langle h,\eta(l)\rangle)^{\mathtt{L}} \subseteq L\ \} \mid L \in \rho\ \}) \\
&= \mathcal{H}_\eta \sqcap \mathcal{M}(\{\ \{\ \langle h,l\rangle \mid \rho[\![P]\!](\langle h,\eta(l)\rangle)^{\mathtt{L}} \subseteq L\ \} \mid L \in \rho\ \}) \\
&\qquad \text{(By Lemma 8.7)}
\end{aligned}$$

Now, if $\langle h',l'\rangle \in \{\ \langle h,l\rangle \mid \rho[\![P]\!](\langle h,\eta(l)\rangle)^{\mathtt{L}} \subseteq L\ \}$, then we have $\forall l'' \in \eta(l').\ \eta(l'') \subseteq \eta(l')$, which implies, by monotonicity, $[\![P]\!](\langle h',\eta(l'')\rangle)^{\mathtt{L}} \subseteq [\![P]\!](\langle h',\eta(l')\rangle)^{\mathtt{L}} \subseteq L$, therefore

$$\langle h',l''\rangle \in \{\ \langle h,l\rangle \mid \rho[\![P]\!](\langle h,\eta(l)\rangle)^{\mathtt{L}} \subseteq L\ \}$$

namely $\langle h', \eta(l')\rangle \subseteq \left\{ \langle h, l\rangle \,\middle|\, \rho[\![P]\!](\langle h, \eta(l)\rangle)^{\mathtt{L}} \subseteq L \right\}$. This simple fact implies that the reduced product cannot refine the property η on the public data.
Consider now $\langle h, l\rangle \in \mathbb{V}^{\mathtt{H}} \times \mathbb{V}^{\mathtt{L}}$, let $L \stackrel{\text{def}}{=} \rho[\![P]\!](\langle h, \eta(l)\rangle)$. Then we have that

$$\langle h, l\rangle \in \left\{ \langle h_1, l_1\rangle \,\middle|\, \rho[\![P]\!](\langle h_1, \eta(l_1)\rangle)^{\mathtt{L}} \subseteq L \right\}$$

At this point, if $\exists L_1 \in \rho.\ \langle h, l\rangle \in \left\{ \langle h_1, l_1\rangle \,\middle|\, \rho[\![P]\!](\langle h_1, \eta(l_1)\rangle)^{\mathtt{L}} \subseteq L_1 \right\}$, then we have $\rho[\![P]\!](\langle h_1, \eta(l_1)\rangle)^{\mathtt{L}} = L \subseteq L_1$ and we obtain

$$\left\{ \langle h_1, l_1\rangle \,\middle|\, \rho[\![P]\!](\langle h_1, \eta(l_1)\rangle)^{\mathtt{L}} \subseteq L \right\} \subseteq \left\{ \langle h_1, l_1\rangle \,\middle|\, \rho[\![P]\!](\langle h_1, \eta(l_1)\rangle)^{\mathtt{L}} \subseteq L_1 \right\}$$

which implies that the Moore closure doesn't add new elements containing $\langle h, l\rangle$, namely

$$\mathcal{R}(\langle h, l\rangle) = \left\{ \langle h_1, l_1\rangle \,\middle|\, \rho[\![P]\!](\langle h_1, \eta(l_1)\rangle)^{\mathtt{L}} \subseteq \rho[\![P]\!](\langle h, \eta(l)\rangle) \right\}.$$

Now, if $\mathcal{R}(\langle h, l\rangle) = \mathcal{R}(\langle h', l'\rangle)$, for some $\langle h'l'\rangle \in \mathbb{V}^{\mathtt{H}} \times \mathbb{V}^{\mathtt{L}}$, then we have

$$\begin{array}{c} \left\{ \langle h_1, l_1\rangle \,\middle|\, \rho[\![P]\!](\langle h_1, \eta(l_1)\rangle)^{\mathtt{L}} \subseteq \rho[\![P]\!](\langle h, \eta(l)\rangle) \right\} = \\ \left\{ \langle h_1, l_1\rangle \,\middle|\, \rho[\![P]\!](\langle h_1, \eta(l_1)\rangle)^{\mathtt{L}} \subseteq \rho[\![P]\!](\langle h', \eta(l')\rangle) \right\} \end{array}$$

This implies that $\langle h, l\rangle \in \left\{ \langle h_1, l_1\rangle \,\middle|\, \rho[\![P]\!](\langle h_1, \eta(l_1)\rangle)^{\mathtt{L}} \subseteq \rho[\![P]\!](\langle h', \eta(l')\rangle)^{\mathtt{L}} \right\}$, i.e., it implies $\rho[\![P]\!](\langle h, \eta(l)\rangle)^{\mathtt{L}} \subseteq \rho[\![P]\!](\langle h', \eta(l')\rangle)^{\mathtt{L}}$.
On the other hand, $\langle h', l'\rangle \in \left\{ \langle h_1, l_1\rangle \,\middle|\, \rho[\![P]\!](\langle h_1, \eta(l_1)\rangle)^{\mathtt{L}} \subseteq \rho[\![P]\!](\langle h, \eta(l)\rangle)^{\mathtt{L}} \right\}$, i.e., $\rho[\![P]\!](\langle h', \eta(l')\rangle)^{\mathtt{L}} \subseteq \rho[\![P]\!](\langle h, \eta(l)\rangle)^{\mathtt{L}}$. So $\rho[\![P]\!](\langle h', \eta(l')\rangle)^{\mathtt{L}} = \rho[\![P]\!](\langle h, \eta(l)\rangle)^{\mathtt{L}}$. If $\rho[\![P]\!](\langle h', \eta(l')\rangle)^{\mathtt{L}} = \rho[\![P]\!](\langle h, \eta(l)\rangle)^{\mathtt{L}}$, then clearly we have $\mathcal{R}(\langle h, l\rangle) = \mathcal{R}(\langle h', l'\rangle)$, by definition of $\mathcal{R}$.

It is worth noting that, by Lemma 8.8, the partition induced by the complete shell of $\mathcal{H}_\eta$ on $\wp(\mathbb{V}^{\mathtt{H}} \times \mathbb{V}^{\mathtt{L}})$ for Eq. 8.2 does not affect the closure η. This means that the only component which is actually refined is the abstraction on private data, and this corresponds to the most abstract partitioning of private data which can generate insecure information flows. This means that any change between equivalent elements does not produce insecure flows, as stated in the following theorem.

Theorem 8.9. *Let $\rho, \eta \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$ then for each $l, l' \in \mathbb{V}^{\mathtt{L}}, h, h' \in \mathbb{V}^{\mathtt{H}}$ we have $\eta(l) = \eta(l') = Y \ \Rightarrow\ (\mathcal{R}(\langle h, l\rangle) = \mathcal{R}(\langle h', l'\rangle)$ iff $h' \in [h]_{\Pi_{\mathtt{P}}(\eta,\rho)_{|Y}})$.*

Proof. The abstract declassification introduced in Sect. 6.4 considers

$$\Pi_{\mathtt{P}}(\eta, \rho) \stackrel{\text{def}}{=} \left\{ \langle \left\{ h \in \mathbb{V}^{\mathtt{H}} \,\middle|\, \rho[\![P]\!](\langle h, \eta(l)\rangle)^{\mathtt{L}} = A \right\}, \eta(l)\rangle \,\middle|\, l \in \mathbb{V}^{\mathtt{L}},\ A \in \rho \right\}$$

We define the sets $\Pi_{\mathtt{P}}(\eta, \rho)_{|Y} = \left\{ X \subseteq \mathbb{V}^{\mathtt{H}} \,\middle|\, \langle X, Y\rangle \in \Pi_{\mathtt{P}}(\eta, \rho) \right\}$, for each $Y \in \eta$, which are partitions of private data. Consider $l, l' \in \mathbb{V}^{\mathtt{L}}$ such that $Y \stackrel{\text{def}}{=} \eta(l) = \eta(l')$. By Lemma 8.8, $\mathcal{R}(\langle h, l\rangle) = \mathcal{R}(\langle h', l'\rangle)$ iff $\rho[\![P]\!](\langle h', \eta(l')\rangle)^{\mathtt{L}} = \rho[\![P]\!](\langle h, \eta(l)\rangle)^{\mathtt{L}}$, for $h, h' \in \mathbb{V}^{\mathtt{H}}$. This can be rewritten as $\rho[\![P]\!](\langle h', Y\rangle)^{\mathtt{L}} = \rho[\![P]\!](\langle h, Y\rangle)^{\mathtt{L}}$. But, by definition of $\Pi_{\mathtt{P}}(\eta, \rho)_{|Y}$, this holds iff $h' \in [h]_{\Pi_{\mathtt{P}}(\eta,\rho)_{|Y}}$.

Next examples show how the most abstract private observable, characterizing abstract declassification, can be obtained as solution of a completeness problem.

Example 8.10. Consider the program fragment:

$$P \stackrel{\text{def}}{=} \; l := l * h^2$$

with $l : \mathtt{L}$ and $h : \mathtt{H}$. We want to find the shell in order to make $\langle \mathcal{H}, \mathcal{H}_{Par} \rangle$ complete for the map $[\![P]\!]^{id,id} = [\![P]\!]$.

$$\begin{aligned}
\mathcal{R}^{\mathcal{H}_{Par}}_{[\![P]\!]}(\mathcal{H}) &= \mathcal{H} \sqcap \mathcal{M} \bigcup_{\langle \mathbb{Z}, L\rangle \in \mathcal{H}_{Par}} \max \left\{ \; X' \;\middle|\; [\![P]\!](X') \subseteq \langle \mathbb{Z}, L\rangle \; \right\} \\
&= \mathcal{H} \sqcap \mathcal{M} \bigcup_{L \in Par} \left\{ \; \langle h, l\rangle \;\middle|\; [\![P]\!](\langle h, l\rangle)^{\mathtt{L}} \subseteq L \; \right\} \\
&= \mathcal{H} \sqcap \mathcal{M} \bigcup_{L \in Par} \left\{ \; \langle h, l\rangle \;\middle|\; h^2 * l \in L \; \right\} \\
&= \mathcal{H} \sqcap \mathcal{M}(\left\{ \; \langle h, l\rangle \;\middle|\; h^2 * l \in 2\mathbb{Z} \; \right\} \cup \left\{ \; \langle h, l\rangle \;\middle|\; h^2 * l \in 2\mathbb{Z} + 1 \; \right\}) \\
&= \mathcal{H} \sqcap \mathcal{M}\{\langle \mathbb{Z}, 2\rangle \cup \langle \mathbb{Z}, 4\rangle \cup \ldots \cup \langle 2\mathbb{Z}, 1\rangle \cup \langle 2\mathbb{Z}, 3\rangle \cup \ldots, \\
&\qquad\qquad \langle 2\mathbb{Z}+1, 1\rangle \cup \langle 2\mathbb{Z}+1, 3\rangle \cup \ldots\}) \\
&= \mathcal{H} \sqcap \mathcal{M}\{\langle \mathbb{Z}, 2\mathbb{Z}\rangle \cup \langle 2\mathbb{Z}, 2\mathbb{Z}+1\rangle, \langle 2\mathbb{Z}+1, 2\mathbb{Z}+1\rangle\}) \\
&= \mathcal{H} \sqcap \left(\left\{ \begin{array}{l} \langle \mathbb{Z}, \mathbb{Z}\rangle, \langle \mathbb{Z}, 2\mathbb{Z}\rangle \cup \langle 2\mathbb{Z}, 2\mathbb{Z}+1\rangle, \langle 2\mathbb{Z}+1, 2\mathbb{Z}+1\rangle, \\ \langle 2\mathbb{Z}+1, 2\mathbb{Z}\rangle, \varnothing \end{array} \right\} \right)
\end{aligned}$$

This means that the reduced product generates also $\langle 2\mathbb{Z}, 2\mathbb{Z}+1\rangle$ and therefore generates $\langle 2\mathbb{Z}, l\rangle$ for each $l \in 2\mathbb{Z}+1$. Let $\overline{\mathcal{H}} \stackrel{\text{def}}{=} \mathcal{R}^{\mathcal{H}_{Par}}_{[\![P]\!]}(\mathcal{H})$, then for instance, we have

$$\begin{array}{l}
\mathcal{H}_{Par} \circ [\![P]\!] \circ \overline{\mathcal{H}}(\langle 2, 3\rangle) = \mathcal{H}_{Par} \circ [\![P]\!](\langle 2\mathbb{Z}, 3\rangle) = \langle \mathbb{Z}, 2\mathbb{Z}\rangle \text{ and} \\
\mathcal{H}_{Par} \circ [\![P]\!](\langle 2, 3\rangle) = \langle \mathbb{Z}, 2\mathbb{Z}\rangle
\end{array}$$

while $\mathcal{H}_{Par} \circ [\![P]\!] \circ \mathcal{H}(\langle 2, 3\rangle) = \mathcal{H}_{Par} \circ [\![P]\!](\mathbb{Z}, 3) = \langle \mathbb{Z}, \mathbb{Z}\rangle$. As in abstract declassification, this means that it is the variation of parity of the private input that generates the flow.

Example 8.11. Consider $\rho \stackrel{\text{def}}{=} \{\mathbb{Z}, 2\mathbb{Z}, 4\mathbb{Z}, 2\mathbb{Z}+1, \varnothing\}$ and $\eta \stackrel{\text{def}}{=} \{\mathbb{Z}, 2\mathbb{Z}, 5\mathbb{Z}, 10\mathbb{Z}, \varnothing\}$, and consider the program fragment

$$P \stackrel{\text{def}}{=} \; \textbf{if } (h \; mod \; 4) = 0 \textbf{ then } l := l * h \textbf{ else } l := l * (h+1)$$

Compute, first, the abstract declassification seen in Sect. 6.4:

$$\Pi_{\mathtt{P}}(\eta, \rho) = \left\{ \begin{array}{l} \langle 4\mathbb{Z} \cup 4\mathbb{Z}+3, 10\mathbb{Z}\rangle, \langle 4\mathbb{Z}+1 \cup 4\mathbb{Z}+2, 10\mathbb{Z}\rangle, \\ \langle 4\mathbb{Z} \cup 4\mathbb{Z}+3, 5\mathbb{Z}\rangle, \langle 4\mathbb{Z}+1, 5\mathbb{Z}\rangle, \langle 4\mathbb{Z}+2, 5\mathbb{Z}\rangle, \\ \langle 4\mathbb{Z} \cup 4\mathbb{Z}+3, 2\mathbb{Z}\rangle, \langle 4\mathbb{Z}+1 \cup 4\mathbb{Z}+2, 2\mathbb{Z}\rangle, \\ \langle 4\mathbb{Z} \cup 4\mathbb{Z}+3, \mathbb{Z}\rangle, \langle 4\mathbb{Z}+1, \mathbb{Z}\rangle, \langle 4\mathbb{Z}+2, \mathbb{Z}\rangle \end{array} \right\}$$

Therefore we obtain $\Pi_{\mathtt{P}}(\eta, \rho)_{|10\mathbb{Z}} = \Pi_{\mathtt{P}}(\eta, \rho)_{|2\mathbb{Z}} = \{4\mathbb{Z} \cup 4\mathbb{Z}+3, 4\mathbb{Z}+1 \cup 4\mathbb{Z}+2\}$ and $\Pi_{\mathtt{P}}(\eta, \rho)_{|5\mathbb{Z}} = \Pi_{\mathtt{P}}(\eta, \rho)_{|\mathbb{Z}} = \{4\mathbb{Z} \cup 4\mathbb{Z}+3, 4\mathbb{Z}+1, 4\mathbb{Z}+2\}$. Consider now the completeness shell:

$$\begin{array}{l}
\left\{ \; \langle h, l\rangle \;\middle|\; [\![P]\!](\langle h, \eta(l)\rangle)^{\mathtt{L}} \subseteq 4\mathbb{Z} \; \right\} = \langle 4\mathbb{Z} \cup 4\mathbb{Z}+3, \mathbb{Z}\rangle \\
\left\{ \; \langle h, l\rangle \;\middle|\; [\![P]\!](\langle h, \eta(l)\rangle)^{\mathtt{L}} \subseteq 2\mathbb{Z} \; \right\} = \langle \mathbb{Z} \smallsetminus 4\mathbb{Z}+2, \mathbb{Z}\rangle \cup \langle 4\mathbb{Z}+2, 2\mathbb{Z}\rangle \\
\left\{ \; \langle h, l\rangle \;\middle|\; [\![P]\!](\langle h, \eta(l)\rangle)^{\mathtt{L}} \subseteq 2\mathbb{Z}+1 \; \right\} = \varnothing
\end{array}$$

Then we have:

$$\mathcal{R}^{\mathcal{H}_\rho}_{[\![P]\!]}(\mathcal{H}_\eta) = \mathcal{H}_\eta \sqcap \mathcal{M}\left(\{\langle\langle \mathbb{Z} \smallsetminus 4\mathbb{Z}+2, \mathbb{Z}\rangle \cup \langle 4\mathbb{Z}+2, 2\mathbb{Z}\rangle, 4\mathbb{Z} \cup 4\mathbb{Z}+3, \mathbb{Z}\rangle\}\right)$$
$$= \mathcal{H}_\eta \cup \left\{ \begin{array}{l} \langle\mathbb{Z} \smallsetminus 4\mathbb{Z}+2, \mathbb{Z}\rangle \cup \langle 4\mathbb{Z}+2, 2\mathbb{Z}\rangle, \langle 4\mathbb{Z} \cup 4\mathbb{Z}+3, \mathbb{Z}\rangle, \\ \langle\mathbb{Z} \smallsetminus 4\mathbb{Z}+2, 5\mathbb{Z}\rangle \cup \langle 4\mathbb{Z}+2, 10\mathbb{Z}\rangle, \langle 4\mathbb{Z} \cup 4\mathbb{Z}+3, 5\mathbb{Z}\rangle, \\ \langle 4\mathbb{Z} \cup 4\mathbb{Z}+3, 2\mathbb{Z}\rangle, \langle 4\mathbb{Z} \cup 4\mathbb{Z}+3, 10\mathbb{Z}\rangle \end{array} \right\}$$

For instance, consider $5, 9 \in 4\mathbb{Z}+1$, $6, 10 \in 4\mathbb{Z}+2$, and note that $\eta(10) = \eta(30) = 10\mathbb{Z}$, and $\eta(5) = \eta(15) = 5\mathbb{Z}$. Note that, 5 and 6 are in the same equivalence class in the partition induced by $\Pi_{\mathtt{P}}\,(\eta, \rho)_{|10\mathbb{Z}}$, written $5 \in [6]_{10\mathbb{Z}}$, and indeed $\mathcal{R}(\langle 5, 10\rangle) = \mathcal{R}(\langle 6, 30\rangle) = \langle\mathbb{Z}, 10\mathbb{Z}\rangle \in \mathcal{H}_\eta$. While $5 \in [9]_{5\mathbb{Z}} \neq [6]_{5\mathbb{Z}}$, namely the partition induced by $\Pi_{\mathtt{P}}\,(\eta, \rho)_{|5\mathbb{Z}}$ distinguishes 5 and 6, while 5 is together with 9 and 6 is together with 10, i.e., $10 \in [6]_{5\mathbb{Z}}$. On the other hand, we have

$$\mathcal{R}(\langle 5, 5\rangle) = \mathcal{R}(\langle 9, 15\rangle) = \langle\mathbb{Z} \smallsetminus 4\mathbb{Z}+2, 5\mathbb{Z}\rangle \cup \langle 4\mathbb{Z}+2, 10\mathbb{Z}\rangle$$

and

$$\mathcal{R}(\langle 6, 5\rangle) = \mathcal{R}(\langle 10, 15\rangle) = \langle\mathbb{Z}, 5\mathbb{Z}\rangle \in \mathcal{H}_\eta$$

8.4 Adjoining observer and observable properties

Modeling attackers means characterizing which is the maximal power of an harmless attacker, i.e., an attacker which is not able to disclose confidential information. Declassification, instead, means characterizing which is the information revealed to a fixed attacker. As we have seen in the previous sections, the model of the most concrete harmless attacker corresponds to the most concrete public observer, while abstract declassification is characterized by the most abstract private observable. Clearly there is a strong relation between these two notions, since the more powerful is attacker and the less confidential information can be kept private. Therefore, it is intuitively clear that there is a *balance* between public observers and private observables, i.e., between attack models and declassification. In other words, the index of the partition of private data for declassification is proportional to the cardinality of the abstract domain which models the precision of the property that the attacker can observe. This phenomenon can be precisely characterized in the lattice of abstract interpretations as an adjunction. In Fig. 8.1 we provide a graphical representation of the relation existing between the most concrete property modeling the public observer and the most abstract property modeling the private observable. In particular, this picture represents the fact that the more powerful is the attacker, i.e., the more concrete is the observer property, the less confidential information can be kept private, i.e., the more concrete is the private observable. In particular, in Fig. 8.1 we also show that, if the arrow represents the most abstract private observable, then when we declassify a confidential property which lays in the white area we cannot guarantee the secrecy of the program, since we are declassifying less than what is released by the semantics. While when

we declassify a property in the filled area, then we guarantee that no confidential information leakage may happen. Moreover, note that, even if the attacker is able to observe the value of public variables, then the observable property can be more abstract then the identity since the program itself can behave as a firewall for certain confidential properties, such as the square operation hides the sign.

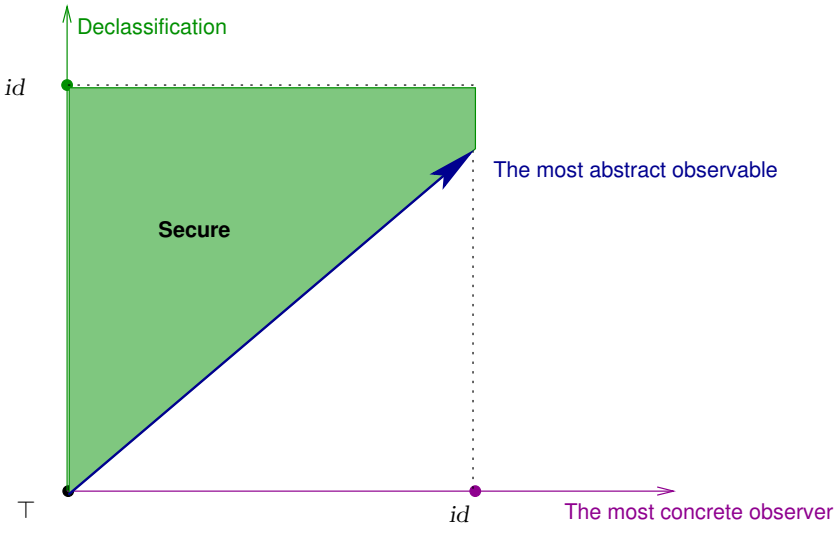


Fig. 8.1. Public observer vs confidential observable

In Section 8.2 and 8.3 we proved that both problems can be viewed as instances of the problem of making abstract interpretations complete. While the private observable for declassification is obtained by computing the completeness shell, the public observer, modeling the attacker, is obtained by computing the completeness core in the same completeness problem. These abstract domain transformers have been proved in [65] to be adjoint functions (see Sect. 2.2.4) on the lattice of abstract interpretations. The following result is therefore a consequence of Theorem 8.5 and 8.9.

Theorem 8.12. *Let $\eta \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$ be a disjunctive property, and P a program. Then we have that $id \sqsubset (\eta)[\![P]\!]\ (id \rightsquigarrow id) \ \Leftrightarrow\ \mathcal{P}(\sqcap_{L\in\eta}\mathcal{M}(\Pi_{\mathtt{P}}(\eta, id)_{|L})) \sqsubset \top$.*

Proof. First of all, note that $id \sqsubset (\eta)[\![P]\!]\ (id \rightsquigarrow id)$ is equivalent to saying $\mathcal{H}_{id} \sqsubset \mathcal{C}^{\mathcal{H}_\eta}_{[\![P]\!]_{\eta,id}}(\mathcal{H}_{id})$, by Th. 8.5. In Sect. 2.2.4 we showed that $\mathcal{C}^{\ell,\eta}_f(\rho) \sqsupset \rho \ \Leftrightarrow\ \eta \sqsupset \mathcal{R}^{\ell,\rho}_f(\eta)$, therefore we have $\mathcal{H}_{id} \sqsubset \mathcal{C}^{\mathcal{H}_\eta}_{[\![P]\!]_{\eta,id}}(\mathcal{H}_{id})$ if and only if $\mathcal{H}_\eta \sqsupset \mathcal{R}^{\mathcal{H}_{id}}_{[\![P]\!]_{\eta,id}}(\mathcal{H}_\eta)$. But this last inequality holds if and only if the partition induced by $\mathcal{R}^{\mathcal{H}_{id}}_{[\![P]\!]_{\eta,id}}(\mathcal{H}_\eta)$ (see Th. 8.9) is strictly more concrete than the $\top$.

This theorem tells us that the more we abstract the public observer, in order to guarantee non-interference, and the more we can concretize the information kept secret.

8.5 Discussion

In this chapter, we formalize the duality existing between the derivation of the most concrete public observer, and of the most abstract private observable, as adjunction in the algebra of abstract domain transformers. This result provides a precise mathematical framework where declassification and attack models can be systematically derived and compared with each other in the lattice of abstract interpretations by applying well known methods for abstract domain design. This framework can be the basis for applying quantitative methods and metrics [16] for measuring the amount of information leaked relatively to a given attack model, or by adjunction, the precision of an attacker under the hypothesis that some information can be declassified. This framework can also be particularly interesting when $[\![\varphi]\!]$ is the semantics of a temporal formula φ in the μ-calculus on a transition system $\mathcal{T} = \langle \Sigma, \rightarrow \rangle$. In this case, abstract non-interference corresponds to prove that the property φ on some *public* states is not affected by varying the *private* ones. Because also bisimulation is a problem of (forward) completeness, as proved in [102], abstract non-interference on temporal logic boils down to bisimulation in a (adjoint) system, such that $\langle \Sigma, \rightsquigarrow \rangle$, where $s \rightsquigarrow s'$ iff $s \rightarrow s_1$ and for all $s' \neq s_1$: $s \not\rightarrow s'$. This result proves a strong link between proving abstract non-interference and proving strong preservation in abstract model checking. Moreover, abstract non-interference can be used in regression testing by typing program variables according to code update: when new code is added to existing code, regression testing verifies that the existing code continues to work correctly. The impact of this observation in security protocols and in regression testing deserves further investigations.

9

Timed Abstract Non-Interference

The true knowledge is daughter of experience,
and experience is mother of any knowledge.
LEONARDO DA VINCI

The notion of abstract non-interference introduced so far, based on denotational semantics compares computations observing simply the results of the computations, without considering any other possible external observation, i.e., without taking into account any possible covert channel. Unfortunately, there are many other aspects of the computation that an attacker may observe in order to get some confidential information. One of these aspects is the time elapsed during the computation. Consider for example the following program fragment:

$$P \stackrel{\text{def}}{=} \textbf{while } h \textbf{ do } l := l; h := h - 1 \textbf{ endw}$$

In this case, the public output is always the same, independently from the initial value of h. Anyway, if the attacker can measure the time elapsed, then it could understand whether the while is executed or not, disclosing some information about the initial value of h. This means that, in this case, the program is not secure, even if our notion of abstract non-interference, such as the standard non-interference, would say that the program guarantees secrecy. The problem is that abstract non-interference is time insensitive, since the semantics used doesn't consider time. We would like to extend the notion of abstract non-interference, in order to model also the case when the attacker is able to observe *timing channels*, namely when it is able to measure the time elapsed during the computation of programs. Let us consider standard imperative languages, where time cannot be modified or used by the system during the computation, as it happens, for example, in real-time systems. We obtain this generalization of abstract non-interference including time by concretizing the semantics used. A first approach consists in considering the maximal trace semantics, instead of the denotational one, since the trace semantics compare

the partial results at each step of computation. Indeed, the trace semantics implicitly embeds the *time* elapsed during computations in the *length* of traces. A more general approach consists in considering an operational semantics that *stores* also the time elapsed during the execution of each statement. In this case we consider this time information as a public datum, i.e., that can be observed by the attacker. Once, that we have a timed notion of abstract non-interference, we are interested in characterizing the relation between proving abstract non-interference with and without time. This would provide the conditions characterizing the attackers whose capability of observing time does not increase their power in disclosing confidential information about data. In other words, we characterize in which conditions timed abstract non-interference, where attackers observe time, implies abstract non-interference, where the same attackers cannot observe time.
In this chapter, we introduce a notion of abstract non-interference which capture timing channels, called *timed abstract non-interference*. This notion provides the appropriate setting for studying how properties, of private data, interfere during the execution of the program, with properties of the elapsed time. On the other hand, in this context, we do not consider how properties of the time elapsed during program execution interfere with properties of data, since we do not consider here real-time system, where time interferes in the execution of a system. We define a timed notion of abstract non-interference by concretizing the semantics, in particular we first introduce abstract non-interference on traces, which can capture timing channels, since the length of traces corresponds to a discretization of the elapsed time. A more precise definition of timed abstract non-interference can be obtained by enriching the operational semantics of Imp in order to measure the elapsed time. We prove that it is always possible to transform the model of an attacker in order to characterize the closest one, with respect to the standard ordering between abstractions [31], that guarantees independence between data and time. This gives a model of the optimal harmless attacker which, by observing time elapsed, is unable to enrich its observational capabilities on data. Finally, since manipulation of data is independent from time, we observe that there cannot be timing channels whenever also time is independent from manipulation of data. In both the approaches to timed abstract non-interference we characterize the relations between the timed and the untimed notion, and moreover we provide the conditions on the attacker model which make these two notions comparable. This section is based on the unpublished paper [56].

9.1 The timed semantics for a deterministic language

In the following, we consider a simple imperative language, Imp introduced in Sect. 4.2.1, with the only difference that, here, we want to analyze also the elapsed time. Therefore, we enhance the operational semantics given in Table 4.3, by considering states that are pairs: one component is the standard representation of

$$\langle \mathbf{nil}, \langle s,t\rangle\rangle \longrightarrow \langle s,t\rangle \qquad \frac{\langle e,\langle s,t\rangle\rangle \longrightarrow n \in \mathbb{V}_x}{\langle x := e,\langle s,t\rangle\rangle \longrightarrow \langle s[n/x], t + t_{\mathtt{A}}\rangle}$$

$$\frac{\langle c_0,\langle s,t\rangle\rangle \longrightarrow \langle s_0,t_0\rangle,\ \langle c_1,\langle s_0,t_0\rangle\rangle \longrightarrow \langle s_1,t_1\rangle}{\langle c_0; c_1,\langle s,t\rangle\rangle \longrightarrow \langle s_1,t_1\rangle}$$

$$\frac{\langle x,\langle s,t\rangle\rangle \longrightarrow 0}{\langle \mathbf{while}\ x\ \mathbf{do}\ c\ \mathbf{endw},\langle s,t\rangle\rangle \longrightarrow \langle s, t + t_{\mathtt{T}}\rangle}$$

$$\frac{\begin{array}{c}\langle x,\langle s,t\rangle\rangle \longrightarrow n \geq 1,\ \langle c,\langle s,t\rangle\rangle \longrightarrow \langle s_0,t_0\rangle,\\ \langle \mathbf{while}\ x\ \mathbf{do}\ c\ \mathbf{endw},\langle s_0,t_0\rangle\rangle \longrightarrow \langle s_1,t_1\rangle\end{array}}{\langle \mathbf{while}\ x\ \mathbf{do}\ c\ \mathbf{endw},\langle s,t\rangle\rangle \longrightarrow \langle s_1, t_1 + t_{\mathtt{T}}\rangle}$$

Table 9.1. Operational timed semantics of Imp

the memory (the standard notion of state), while the other component is a non-negative number denoting the elapsed time. This new semantics is given in Table 9.1, where $s \in \Sigma$, $t \in \mathbb{N}$ and $t_{\mathtt{A}}, t_{\mathtt{T}} \in \mathbb{N}$ are constant values denoting respectively the time spent for an assignment and for a test. As before, if $|Var(P)| = n$ and $\Sigma = \mathbb{V}^n$ is the set of values for the variables, then the states in the concrete semantics are $\widehat{\Sigma} \stackrel{\text{def}}{=} \Sigma \times \mathbb{N}$, namely a state is a pair composed by a tuple of values for the variable and by a natural value representing the time passed from the beginning of the execution. The operational semantics naturally induces a transition relation on a set of states Σ, denoted $\longrightarrow$, specifying the relation between a state and its possible successors. In order to define the maximal trace semantics in this context where states store also information about the elapsed time, we start from the tree model of a program P, i.e., $\{\!|P|\!\}$, and we derive its trace semantics as an abstraction of this model (see Sect. 4.1.2). Namely, the *maximal trace semantics* [33] of a transition system associated with a program P is $\langle\!|P|\!\rangle \stackrel{\text{def}}{=} \alpha_{\mathtt{T}}(\{\!|P|\!\})$. We use the same notation used in Sect. 4.1.2 extended to states with time, therefore if $\widehat{\sigma} \in \langle\!|P|\!\rangle^+ \stackrel{\text{def}}{=} \alpha_{\mathtt{T}}(\{\!|P|\!\}^+)$, then $\widehat{\sigma}_{\dashv}$ and $\widehat{\sigma}_{\vdash}$ denote respectively the final and initial state of $\widehat{\sigma}$. The *denotational semantics* associates input/output functions with programs, by modeling non-termination by $\perp$, and it is derived from the maximal trace semantics with abstraction $\alpha^{\mathcal{D}}$ (see Tab. 4.1). Note that, since our programs are deterministic, $\alpha^{\mathcal{D}}(X)(\widehat{s})$ is always a singleton. It is well known that we can associate, inductively on its syntax, with each program $P \in$ Imp a function $[\![P]\!]$ denoting its input/output relation, such that $[\![P]\!] \stackrel{\text{def}}{=} \alpha^{\mathcal{D}}(\langle\!|P|\!\rangle)$.

9.2 Timed abstract non-interference on traces

One of the most important features of abstract non-interference is that it is parametric on the chosen semantics. This means that we can enrich/change the checked

notion of abstract non-interference for imperative languages simply by enriching/changing the considered semantics. For this reason, the idea for making abstract non-interference time sensitive, namely able to detect timing channels, is to consider a more concrete semantics observing time. The first approach consists in considering the maximal trace semantics, instead of the denotational one, since the trace semantics compare the partial results at each step of computation. Indeed, the trace semantics implicitly embeds the *time* elapsed during computations in the *length* of traces since we can suppose that the time is measured as the discrete number of computational steps executed by the system. This observation suggests us that the trace semantics can be used for defining a stronger notion of non-interference that capture also timing channels. Therefore, we assume that we can have a timing channel in presence of an attacker that can count the number of execution steps, which means that the attacker observes time by looking at the program counter.

In Sect. 6.5.1 we introduced a notion of abstract non-interference based on the trace semantics. The semantics that we obtain so far distinguishes traces that differ only for the repetition of states, namely it models also the timing channels due to the capability of the attacker of observing the clock of the program. In order to check this further kind of non-interference we consider another abstraction of the semantics. First of all we have to consider the safety abstraction of the trace semantics, which add all the prefixes of the traces [72]: $\langle\!|P|\!\rangle^{safe} = \alpha^{safe}(\langle\!|P|\!\rangle^{\rho})$ where $\alpha^{safe}(X) = \left\{ \delta \,\middle|\, \delta \preccurlyeq \sigma,\ \sigma \in X \right\}$, where $\preccurlyeq$ is the prefix relation between traces. This is clearly an abstraction of the power domain of traces. At this point, in order to check narrow non-interference we can consider the following steps of abstractions.

$$\alpha^{\mathtt{D}}_i(X) = \lambda s.\ \left\{ \sigma_{\dashv} \,\middle|\, \sigma \in X,\ |\sigma| = i+1, \sigma_{\vdash} = s \right\}$$
$$\alpha^{\mathtt{D}}(X) = \left\{ \alpha^{\mathtt{D}}_i(X) \,\middle|\, 0 < i < \max\left\{ |\sigma| \,\middle|\, \sigma \in X \right\} \right\}$$

then we denote by $[\![P]\!]^{safe}_i \stackrel{\text{def}}{=} \alpha^{\mathtt{D}}_i(\langle\!|P|\!\rangle^{safe})$ and therefore, we denote by $[\![P]\!]^{safe}$ the abstract semantics $\alpha^{\mathtt{D}}(\langle\!|P|\!\rangle^{safe}) = \{[\![P]\!]^{safe}_i | 0 < i < \max\{|\sigma|\ |\sigma \in \langle\!|P|\!\rangle^{safe}\}\}$. Moreover, if $[\![P]\!]^{safe}_i(s) = s'$ then we write $s \mapsto_i s'$. Now we can formalize the abstract non-interference for timing channels in the following way:

P is *secure* for timed abstract non-interference if $\forall h_1, h_2 \in \mathbb{V}^{\mathtt{H}}, \forall l \in \mathbb{V}^{\mathtt{L}}$.
$\forall i$. $\rho([\![P]\!]^{safe}_i(\phi(h_1), \eta(l))^{\mathtt{L}}) = \rho([\![P]\!]^{safe}_i(\phi(h_2), \eta(l))^{\mathtt{L}})$.

The interesting aspect of this extension is that we can apply the transformers defined on abstract non-interference simply by considering the approximation based on *bounded iteration* (see Sect. 6.3.3). Bounded iteration, in fact, proves I/O non-interference by requiring a stronger condition, i.e., it requires that all the partial computations provide the same public output.

At this point, since we observed that the simple use of traces makes abstract non-interference time sensitive, we could wonder how we can obtain a notion

of abstract non-interference on traces which is time-insensitive. Therefore, we want to derive a trace semantics which is not able to observe the clock, namely we have to make indistinguishable traces that differ only for the repetition of states. This is a known notion in literature called *stuttering* [3], indeed a semantics is said to be without stuttering if it is insensitive to the repetition of states. Namely we can think of transforming the set of traces that gives semantics to the program by eliminating the stuttering and then we can check non-interference exactly as we have done before. Let X be a property on traces σ. Then X is safety without stuttering (also called stuttering) if it is safety and if $\sigma \in X \,.\, \sigma = \sigma_0\sigma_1 \ldots \sigma_n \ldots$ then $\forall i \geq 0 \,.\, \sigma_0 \ldots \sigma_i\sigma_i \ldots \in X$. It is straightforward to show that the following abstraction, which characterizes the stuttering properties, is an abstraction of sets of traces [55]. Then $\langle\!|P|\!\rangle^{stu} = \alpha^{stu}(\langle\!|P|\!\rangle^{safe})$ where

$$\alpha^{stu}(X) = \left\{ \langle \sigma_0, \ldots, \sigma_n \rangle \,\middle|\, \begin{array}{l} \exists \delta \in X \,.\, \delta = \langle \sigma_0^{k_0} \ldots, \sigma_n^{k_n} \rangle, \\ \forall i \,.\, k_i \in \mathbb{N} \smallsetminus \{0\},\ \sigma_i \neq \sigma_{i+1} \end{array} \right\}$$

Therefore, we can check abstract non-interference on traces for attacker that cannot observe time elapsed, by considering the abstraction $\alpha^{\mathtt{D}}$ as above. We denote by $[\![P]\!]_i^{stu} \stackrel{\text{def}}{=} \alpha_i^{\mathtt{D}}(\langle\!|P|\!\rangle^{stu})$ and therefore

$$[\![P]\!]^{stu} \stackrel{\text{def}}{=} \alpha^{\mathtt{D}}(\langle\!|P|\!\rangle^{stu}) = \left\{ [\![P]\!]_i^{stu} \,\middle|\, 0 < i < \max\left\{ |\sigma| \,\middle|\, \sigma \in \langle\!|P|\!\rangle^{stu} \right\} \right\}.$$

Moreover, if $[\![P]\!]_i^{stu}(s) = s'$ then we also write $s \mapsto_i s'$. Now we can formalize abstract non-interference in the following way:

$$\begin{array}{|c|}\hline P \text{ is } \textit{secure} \text{ for abstract non-interference if } \forall h_1, h_2 \in \mathbb{V}^{\mathtt{H}}, \forall l \in \mathbb{V}^{\mathtt{L}} \,. \\ \forall i \,.\, \rho([\![P]\!]_i^{stu}(\phi(h_1), \eta(l))^{\mathtt{L}}) = \rho([\![P]\!]_i^{stu}(\phi(h_2), \eta(l))^{\mathtt{L}}). \\ \hline \end{array}$$

Example 9.1. We show now the differences between the semantics described above. Consider the following trace semantics of a program P, with states $\langle h, l \rangle$, consider $n \in \mathbb{N}$ and $m \in 2\mathbb{N} + 1$:

$$\langle\!|P|\!\rangle = \{\langle 0,2\rangle \longrightarrow \langle 0,3\rangle \longrightarrow \langle 0,5\rangle, \langle n,2\rangle \longrightarrow \langle 0,2\rangle \longrightarrow \langle 0,3\rangle, \langle n,m\rangle \longrightarrow \langle n,m{+}1\rangle\}$$

Consider the property $\rho = Par$. We want to determine the abstract trace semantics relatively to Par:

$$\begin{array}{l} \langle\!|P|\!\rangle^{Par} = \alpha^{Par}(\langle\!|P|\!\rangle) = \{\langle 0, 2\mathbb{N}\rangle \longrightarrow \langle 0, 2\mathbb{N}+1\rangle \longrightarrow \langle 0, 2\mathbb{N}+1\rangle, \\ \qquad\qquad \langle n, 2\mathbb{N}\rangle \longrightarrow \langle 0, 2\mathbb{N}\rangle \longrightarrow \langle 0, 2\mathbb{N}+1\rangle, \langle n, 2\mathbb{N}+1\rangle \longrightarrow \langle n, 2\mathbb{N}\rangle\} \end{array}$$

At this point, we can compute the abstract denotational semantics by using the known abstraction:

$$[\![P]\!]_{Par} = \alpha^{\mathcal{D}}(\langle\!|P|\!\rangle^{Par}) = \{2\mathbb{N} \mapsto 2\mathbb{N}+1, 2\mathbb{N}+1 \mapsto 2\mathbb{N}\}$$

such that the standard non-interference on it is narrow non-interference on the concrete semantics. Let us introduce, now, the time. We have first to compute the safety abstraction:

$$\begin{aligned}\langle\!|P|\!\rangle^{safe} = \alpha^{safe}(\langle\!|P|\!\rangle^{Par}) = \{&\langle 0, 2\mathbb{N}\rangle, \langle 0, 2\mathbb{N}\rangle \longrightarrow \langle 0, 2\mathbb{N}+1\rangle,\\ &\langle 0, 2\mathbb{N}\rangle \longrightarrow \langle 0, 2\mathbb{N}+1\rangle \longrightarrow \langle 0, 2\mathbb{N}+1\rangle, \langle n, 2\mathbb{N}\rangle, \langle n, 2\mathbb{N}\rangle \longrightarrow \langle 0, 2\mathbb{N}\rangle,\\ &\langle n, 2\mathbb{N}\rangle \longrightarrow \langle 0, 2\mathbb{N}\rangle \longrightarrow \langle 0, 2\mathbb{N}+1\rangle, \langle n, 2\mathbb{N}+1\rangle, \langle n, 2\mathbb{N}+1\rangle \longrightarrow \langle n, 2\mathbb{N}\rangle\}\end{aligned}$$

namely the semantics for checking non-interference is:

$$\begin{aligned}[\![P]\!]^{safe} = \alpha^{\mathtt{D}}(\langle\!|P|\!\rangle^{safe}) = \{&\langle 0, 2\mathbb{N}\rangle \mapsto_1 \langle 0, 2\mathbb{N}+1\rangle,\ \langle 0, 2\mathbb{N}\rangle \mapsto_2 \langle 0, 2\mathbb{N}+1\rangle,\\ &\langle n, 2\mathbb{N}\rangle \mapsto_1 \langle 0, 2\mathbb{N}\rangle, \langle n, 2\mathbb{N}\rangle \mapsto_2 \langle 0, 2\mathbb{N}+1\rangle,\ \langle n, 2\mathbb{N}+1\rangle \mapsto_1 \langle n, 2\mathbb{N}\rangle\}\end{aligned}$$

From this semantics we note that when we start with an even low variable, then we have interference (in general it can be both security and deceptive interference). Now, if we want to guarantee non-interference knowing that the attacker cannot observe the time elapsed, then we use the stuttering abstraction, obtaining:

$$\begin{aligned}\langle\!|P|\!\rangle^{stu} = \alpha^{stu}(\langle\!|P|\!\rangle^{safe}) = \{&\langle 0, 2\mathbb{N}\rangle,\ \langle 0, 2\mathbb{N}\rangle \longrightarrow \langle 0, 2\mathbb{N}+1\rangle,\\ &\langle n, 2\mathbb{N}\rangle \longrightarrow \langle 0, 2\mathbb{N}+1\rangle, \langle n, 2\mathbb{N}+1\rangle, \langle n, 2\mathbb{N}+1\rangle \longrightarrow \langle n, 2\mathbb{N}\rangle\}\end{aligned}$$

Finally, we find the denotational abstraction, which is

$$\begin{aligned}[\![P]\!]^{stu} = \alpha^{\mathtt{D}}(\langle\!|P|\!\rangle^{stu}) = \{&\langle 0, 2\mathbb{N}\rangle \mapsto_1 \langle 0, 2\mathbb{N}+1\rangle,\ \langle n, 2\mathbb{N}\rangle \mapsto_1 \langle 0, 2\mathbb{N}+1\rangle,\\ &\langle n, 2\mathbb{N}+1\rangle \mapsto_1 \langle n, 2\mathbb{N}\rangle\}\end{aligned}$$

In this case, we note that there's not interference, since for each abstract property we have only one possible result.

9.3 Timed abstract non-interference in sequential systems

A more general way for concretizing the semantics in order to make abstract non-interference time sensitive, is to consider timed denotational semantics $[\![P]\!]^{+T}$ as introduced in Sect. 9.1.

Consider transition systems where the states contain the information of time. In this way, we explicitly treat time in abstract non-interference, which means that we could use languages where time can interfere in the flow of computation.

Let us consider $\widehat{\Sigma}$, where the low input is in general a pair where the first component is a tuple of possible values for low variables, and the second one is the time passed, i.e., $\widehat{l} = \langle l, t\rangle$. We denote by $\widehat{l}^{\mathtt{D}} = l \in \mathbb{V}^{\mathtt{L}}$ the projection on the data component, and $\widehat{l}^{\mathtt{T}} = t \in \mathbb{N}$ the projection on the time component. Therefore, in the following a state $\widehat{\sigma}$ will be the triple $\langle s^{\mathtt{H}}, s^{\mathtt{L}}, t\rangle$. The initial states are of the kind $\widehat{\sigma} = \langle s_i, 0\rangle$, since we suppose to start measuring time when the computation starts. With these premises we can formulate the standard notion of non-interference without timing channels. In the following we will denote by $[\![P]\!]^{+\mathtt{T}}$ the denotational semantics measuring time, obtained by using the rules in Table 9.1.

A program P is *secure* if $\forall v \in \mathbb{V}^{\mathtt{L}}, t \in \mathbb{N}, \forall v_1, v_2 \in \mathbb{V}^{\mathtt{H}}$.
$$([\![P]\!]^{+\mathtt{T}}(\langle v_1, v, 0\rangle))^{\mathtt{L}\,\mathtt{T}} = ([\![P]\!]^{+\mathtt{T}}(\langle v_2, v, 0\rangle))^{\mathtt{L}\,\mathtt{T}}$$

Consider the closure $\rho \in uco(\wp(\widehat{\Sigma}^{\mathtt{L}\,\mathtt{T}}))$ where $\widehat{\Sigma}^{\mathtt{L}\,\mathtt{T}} = \mathbb{V}^{\mathtt{L}} \times \mathbb{N}$. In the following, we assume $\eta, \rho \in uco(\wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N}))$ and $\phi \in uco(\wp(\mathbb{V}^{\mathtt{H}}))$. We can therefore define the notions of narrow and abstract timed non-interference. In order to distinguish the timed notions from the ones introduced before, when a program satisfies timed narrow or abstract non-interference we write respectively $[\eta]P^{+\mathtt{T}}\ (\rho)$ and $(\eta)P^{+\mathtt{T}}\ (\phi \rightsquigarrow\!\!\!\!\shortmid\!\!\rho)$.

Definition 9.2.

- *A program* $P \in$ Imp *is such that* $[\eta]P^{+\mathtt{T}}\ (\rho)$ *if* $\forall h_1, h_2 \in \mathbb{V}^{\mathtt{H}}, \forall \widehat{l_1}, \widehat{l_2} \in \mathbb{V}^{\mathtt{L}} \times \{0\}$ *such that* $\eta(\widehat{l_1})^{\mathtt{L}} = \eta(\widehat{l_2})^{\mathtt{L}} \ \Rightarrow\ \rho([\![P]\!]^{+\mathtt{T}}(\langle h_1, \widehat{l_1}\rangle)^{\mathtt{L}\,\mathtt{T}}) = \rho([\![P]\!]^{+\mathtt{T}}(\langle h_2, \widehat{l_2}\rangle)^{\mathtt{L}\,\mathtt{T}})$.
- *A program* $P \in$ Imp *is such that* $(\eta)P^{+\mathtt{T}}\ (\phi \rightsquigarrow\!\!\!\!\shortmid\!\!\rho)$ *if* $\forall h_1, h_2 \in \mathbb{V}^{\mathtt{H}}, \forall \widehat{l} \in \mathbb{V}^{\mathtt{L}} \times \{0\}$ *we have* $\rho([\![P]\!]^{+\mathtt{T}}(\langle \phi(h_1), \eta(\widehat{l})^{\mathtt{L}}\rangle)^{\mathtt{L}\,\mathtt{T}}) = \rho([\![P]\!]^{+\mathtt{T}}(\langle \phi(h_2), \eta(\widehat{l})^{\mathtt{L}}\rangle)^{\mathtt{L}\,\mathtt{T}})$.

It is clear that the only difference between these notions and the untimed ones is in the semantics, therefore we can inherit, in a straightforward way, the whole construction made in the previous sections, simply by considering the time as a further public datum. In particular this allows us to derive the most concrete property, about time, that an harmless attacker can observe, as we can see in the following example.

Example 9.3. Let us consider a really simple example:

$$P \stackrel{\text{def}}{=} h := h \ mod\ 4;\ \textbf{while}\ h\ \textbf{do}\ l := 2l - l; h := h - 1;\ \textbf{endw}$$

with security typing $t = \langle h : \mathtt{H}, l : \mathtt{L}\rangle$ and $\mathbb{V}^{\mathtt{L}} = \mathbb{N}$. Let us consider the trace semantics where each state is $\langle h, l, t\rangle$. Consider $l \in \mathbb{V}^{\mathtt{L}}$ and $h \in \mathbb{V}^{\mathtt{H}}$, $h \neq 0$:

$$\begin{array}{l}\langle 0, l, 0\rangle \longrightarrow \langle 0, l, t_{\mathtt{A}}\rangle \longrightarrow \langle 0, l, t_{\mathtt{A}} + t_{\mathtt{T}}\rangle \\ \langle h, l, 0\rangle \longrightarrow \langle h\ mod\ 4, l, t_{\mathtt{A}}\rangle \longrightarrow \langle (h\ mod\ 4) - 1, l, 3t_{\mathtt{A}} + t_{\mathtt{T}}\rangle\rangle \\ \qquad\qquad \longrightarrow \langle 0, l, 2(h\ mod\ 4)t_{\mathtt{A}} + (h\ mod\ 4 + 1)t_{\mathtt{T}}\rangle\rangle\end{array}$$

Therefore, if for example $h\ mod\ 4 = 2$ then the total time is $4t_{\mathtt{A}} + 3t_{\mathtt{T}}$. This means that the most concrete abstraction of the domain of time that avoids timing channels is the one that have the element $\{t_{\mathtt{A}} + t_{\mathtt{T}}, 2t_{\mathtt{A}} + 2t_{\mathtt{T}}, 4t_{\mathtt{A}} + 3t_{\mathtt{T}}, 6t_{\mathtt{A}} + 4t_{\mathtt{T}}\}$ and abstracts all the other natural numbers in themselves.

Now that we have defined abstract non-interference for the timed denotational semantics, we would like to understand the relation existing with abstract non-interference. Our aim is to compare abstract non-interference defined in terms of standard denotational semantics, with timed non-interference, in order to study the relation existing between these two models. Starting from closures $\eta, \rho \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$ without time, the extension to semantics with time is trivially obtained by taking the closure that is not able to observe anything about time, i.e., we interpret a generic closure $\eta \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$ as the closure that is not able to observe time, therefore that abstracts time to the top: $\eta^{+\mathtt{T}} = \langle \eta, \lambda t.\, \mathbb{N}\rangle$. It is worth noting that $[\eta]P\ (\rho) \ \Leftrightarrow\ [\eta^{+\mathtt{T}}]P^{+\mathtt{T}}\ (\rho^{+\mathtt{T}})$. In other words, since time can

be treated as an additional public variable, we could see abstract non-interference as timed abstract non-interference where the time variable is abstracted to the top. Unfortunately, if we start from a semantics with time and we want to derive abstract non-interference properties without time, then the relation is not so immediate. Indeed, if time interferes with data we have that the abstraction of time is not straight. Clearly, time cannot interfere with data in the concrete semantics for the simple imperative language we are considering, but it can interfere in the abstract semantics modeling the attacker. In other words, when we consider timed abstract non-interference, the attacker model could observe *relations* between data and time, avoiding the independent abstraction of time. Namely if we abstract time we may lose something about the attacker's observation of data. Therefore, if we start from closure operators on semantics with time, namely from the closures $\eta, \rho \in \mathit{uco}(\wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N}))$, we can obtain properties without time in two ways. We can think of erasing the observation of time only in the output, i.e., we abstract away the information about time, or we can think of collecting all the possible results for every possible time value. In this way, we are able to ignore the information about time also in the input.

In the following, we characterize some necessary and sufficient conditions on the attacker model, that indeed make timed abstract non-interference stronger than abstract non-interference, and therefore the two notions comparable.

Abstracting time in the output.

Consider the first case, namely we do not observe time in the output. Let us define the projection, of a pair $X \in \wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N})$, on data (first component) or on time (second component) in the following way:

$$\Pi_{\mathtt{T}}(X) \stackrel{\text{def}}{=} \left\{ \langle x, y\rangle \,\middle|\, \exists y' \in \mathbb{N} \,.\, \langle x, y'\rangle \in X,\ y \in \mathbb{N} \right\}$$

$$\Pi_{\mathtt{D}}(X) \stackrel{\text{def}}{=} \left\{ \langle x, y\rangle \,\middle|\, \exists x' \in \mathbb{N} \,.\, \langle x', y\rangle \in X,\ x \in \mathbb{V}^{\mathtt{L}} \right\}$$

In particular, given a closure $\rho \in \mathit{uco}(\wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N}))$, we can apply these abstractions to ρ obtaining

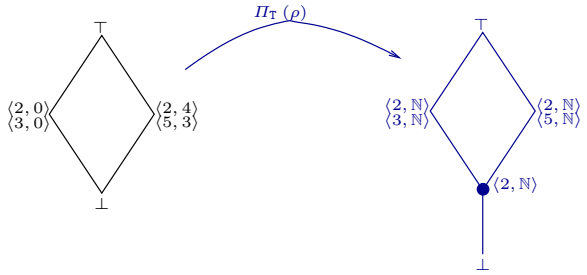
$$\Pi_{\mathtt{T}}(\rho) \stackrel{\text{def}}{=} \left\{ \Pi_{\mathtt{T}}(X) \,\middle|\, X \in \rho \right\} \text{ and } \Pi_{\mathtt{D}}(\rho) \stackrel{\text{def}}{=} \left\{ \Pi_{\mathtt{D}}(X) \,\middle|\, X \in \rho \right\}$$

Proposition 9.4.
$\Pi_{\mathtt{T}} \in \mathit{uco}(\wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N}))$ and $\Pi_{\mathtt{D}} \in \mathit{uco}(\wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N}))$.

Proof. We prove the fact only for $\Pi_{\mathtt{T}}$, since the other proof is analogous. In order to prove the thesis we have to show that $\Pi_{\mathtt{T}}$ is extensive, monotone and idempotent. By definition we have that $\Pi_{\mathtt{T}}(X) \stackrel{\text{def}}{=} \left\{ \langle x, y\rangle \,\middle|\, \exists y' \in \mathbb{N} \,.\, \langle x, y'\rangle \in X,\ y \in \mathbb{N} \right\}$, therefore if $\langle x, y\rangle \in X$, then we have $\forall z \in \mathbb{N} \,.\, \langle x, z\rangle \in \Pi_{\mathtt{T}}(X)$, which means that in particular $\langle x, y\rangle \in \Pi_{\mathtt{T}}(X)$, proving that $\Pi_{\mathtt{T}}$ is extensive. Let us prove monotonicity. Consider $X, Y \in \wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N})$ such that $X \subseteq Y$. Let $\langle x, y\rangle \in \Pi_{\mathtt{T}}(X)$, this means that

there exists $y' \in \mathbb{N}$ such that $\langle x, y' \rangle \in X$ and $y \in \mathbb{N}$. Since $X \subseteq Y$, we have also that $\langle x, y' \rangle \in Y$, which implies that $\langle x, y \rangle \in \left\{ \langle x, y \rangle \,\middle|\, \exists y' \in \mathbb{N} \,.\, \langle x, y' \rangle \in Y,\ y \in \mathbb{N} \right\} = \Pi_{\mathtt{T}}(Y)$. Namely $\Pi_{\mathtt{T}}(X) \subseteq \Pi_{\mathtt{T}}(Y)$. Finally let's prove idempotence. First of all, note that $\Pi_{\mathtt{T}}(\Pi_{\mathtt{T}}(X)) \supseteq \Pi_{\mathtt{T}}(X)$, by extensivity of $\Pi_{\mathtt{T}}(X)$. Let $\langle x, y \rangle \in \Pi_{\mathtt{T}}(\Pi_{\mathtt{T}}(X))$, by definition there exists $y' \in \mathbb{N}$ such that $\langle x, y' \rangle \in \Pi_{\mathtt{T}}(X)$ and $y \in \mathbb{N}$. By definition of $\Pi_{\mathtt{T}}(X)$, this implies that $\forall z \in \mathbb{N} \,.\, \langle x, z \rangle \in \Pi_{\mathtt{T}}(X)$, therefore in particular $\langle x, y \rangle \in \Pi_{\mathtt{T}}(X)$. We proved in this way that $\Pi_{\mathtt{T}}(\Pi_{\mathtt{T}}(X)) \subseteq \Pi_{\mathtt{T}}(X)$, and therefore the equality, which is idempotence.

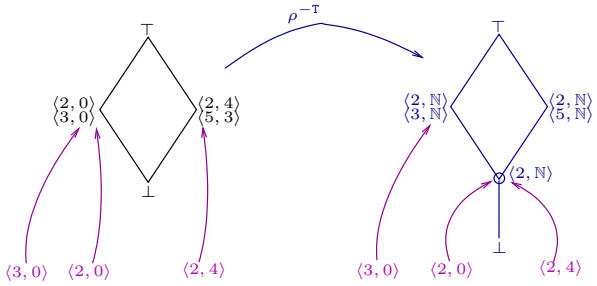
It is clear that the set $\Pi_{\mathtt{T}}(\rho)$ is the set of the images of the map $\Pi_{\mathtt{T}} \circ \rho$. Note that, even if both $\Pi_{\mathtt{T}}$ (or $\Pi_{\mathtt{D}}$) and ρ are closure operator, then their composition may not be a closure operator, as we can see in the picture below, where we have $\langle 2, \mathbb{N} \rangle \notin \Pi_{\mathtt{T}}(\rho)$.



Since in general $\Pi_{\mathtt{T}} \circ \rho$ is not an upper closure operator, we define the closures $\rho^{-\mathtt{T}} \stackrel{\text{def}}{=} \mathcal{M}(\Pi_{\mathtt{T}}(\rho))$ and $\rho^{-\mathtt{D}} \stackrel{\text{def}}{=} \mathcal{M}(\Pi_{\mathtt{D}}(\rho))$.

Proposition 9.5. *Let $\eta \in uco(\wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N}))$, then we have that both $\eta^{-\mathtt{T}}$ and $\eta^{-\mathtt{D}}$ are closures, i.e., $\eta^{-\mathtt{T}} \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$ and $\eta^{-\mathtt{D}} \in uco(\wp(\mathbb{N}))$.*

The fact that in general we have $\rho^{-\mathtt{T}} \neq \Pi_{\mathtt{T}}(\rho)$ and $\rho^{-\mathtt{D}} \neq \Pi_{\mathtt{D}}(\rho)$ is a problem when we want to compare timed non-interference with abstract non-interference since elements that have the same image in ρ may be different in $\rho^{-\mathtt{T}}$, and viceversa as we can see in the picture below.



So, we would like to characterize when the two notions are comparable. The picture above shows that problems arise when the Moore closure adds new points, namely when $\Pi_{\mathtt{T}}(\rho)$ is not a closure. Therefore, we first want to understand when

this is an upper closure operator, namely when $\Pi_{\tau} \circ \rho \in uco(\wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N}))$. It is well known in literature [93] that, given two closures $\rho, \pi \in uco(C)$, then we have $\pi \circ \rho \in uco(C)$ iff $\pi \circ \rho = \rho \circ \pi = \rho \sqcup \pi$. This means that we need a ρ such that $\Pi_{\tau} \circ \rho = \rho \circ \Pi_{\tau}$. At this point, we can note that if two closure commute then one is complete as regards the other and viceversa, for both forward and backward completeness.

Lemma 9.6. *Let $\rho, \pi \in uco(C)$, the following facts are equivalent:*

1. $\rho \circ \pi = \pi \circ \rho$;
2. $\pi \circ \rho = \pi \circ \rho \circ \pi$ *(backward completeness);*
3. $\rho \circ \pi = \pi \circ \rho \circ \pi$ *(forward completeness).*

Proof. (1) $\Leftrightarrow$ (2) Consider $\rho \circ \pi = \pi \circ \rho$, then $\pi \circ \rho \circ \pi = \pi \circ \pi \circ \rho = \pi \circ \rho$ by hypothesis and idempotence of π. Namely we proved that $\rho \circ \pi = \pi \circ \rho \Rightarrow \pi \circ \rho = \pi \circ \rho \circ \pi$. Consider now $\pi \circ \rho = \pi \circ \rho \circ \pi$. First of all, note that $\rho \circ \pi \leq \pi \circ \rho \circ \pi = \pi \circ \rho$ by extensivity of π and by the hypothesis, therefore $\rho \circ \pi \leq \pi \circ \rho$ $(*)$. Moreover note that $\pi \circ \rho \leq \rho \circ \pi \circ \rho$ by extensivity of ρ, and $\pi \circ \rho = \pi \circ \rho \circ \rho \geq \rho \circ \pi \circ \rho$ by idempotence of ρ and by $(*)$. Therefore we proved that $\pi \circ \rho = \rho \circ \pi \circ \rho$ $(**)$. At this point note that $\pi \circ \rho$ is monotone and extensive, since composition of monotone and extensive maps. Let us prove that it is also idempotent.

$$\begin{aligned} \pi \circ \rho &= \rho \circ \pi \circ \rho && \text{(by the property } (**)) \\ &\leq \pi \circ \rho \circ \pi \circ \rho && \text{(by extensivity of } \pi) \end{aligned}$$

On the other hand:

$$\begin{aligned} \pi \circ \rho \circ \pi \circ \rho &\leq \pi \circ \rho \circ \pi \circ \rho \circ \pi && \text{(by monotonicity)} \\ &= \pi \circ \pi \circ \rho \circ \pi && \text{(by the property } (**)) \\ &= \pi \circ \rho \circ \pi && \text{(by idempotence of } \pi) \\ &= \pi \circ \rho && \text{(by the hypothesis)} \end{aligned}$$

Therefore we proved that $\pi \circ \rho \circ \pi \circ \rho = \pi \circ \rho$, namely that $\pi \circ \rho$ is idempotent. This also proves that $\pi \circ \rho \in uco(C)$ that by [93] implies that $\pi \circ \rho = \rho \circ \pi$.

(1) $\Leftrightarrow$ (3) Consider $\rho \circ \pi = \pi \circ \rho$, then $\pi \circ \rho \circ \pi = \rho \circ \pi \circ \pi = \rho \circ \pi$ by hypothesis and idempotence of π. Namely we proved that $\rho \circ \pi = \pi \circ \rho \Rightarrow \rho \circ \pi = \pi \circ \rho \circ \pi$. Consider now $\rho \circ \pi = \pi \circ \rho \circ \pi$. First of all note that $\pi \circ \rho \leq \pi \circ \rho \circ \pi = \rho \circ \pi$ by monotonicity of $\pi \circ \rho$ and by the hypothesis, therefore $\rho \circ \pi \geq \pi \circ \rho$ $(*)$. Moreover note that $\rho \circ \pi \leq \rho \circ \pi \circ \rho$ by monotonicity of $\rho \circ \pi$, and $\rho \circ \pi = \rho \circ \rho \circ \pi \geq \rho \circ \pi \circ \rho$ by idempotence of ρ and by $(*)$. Therefore we proved that $\rho \circ \pi = \rho \circ \pi \circ \rho$ $(**)$. At this point note that $\rho \circ \pi$ is monotone and extensive, since composition of monotone and extensive maps. Let us prove that it is also idempotent.

$$\begin{aligned} \rho \circ \pi &= \rho \circ \pi \circ \rho && \text{(by the property } (**)) \\ &\leq \rho \circ \pi \circ \rho \circ \pi && \text{(by monotonicity of } \rho \circ \pi \circ \rho) \end{aligned}$$

On the other hand:

$$\begin{aligned}\rho\circ\pi\circ\rho\circ\pi &\leq \pi\circ\rho\circ\pi\circ\rho\circ\pi \qquad \text{(by extensivity of } \pi)\\ &= \pi\circ\rho\circ\pi\circ\pi \qquad \text{(by the property } (**))\\ &= \pi\circ\rho\circ\pi \qquad \text{(by idempotence of } \pi)\\ &= \rho\circ\pi \qquad \text{(by the hypothesis)}\end{aligned}$$

Therefore we proved that $\rho\circ\pi\circ\rho\circ\pi = \rho\circ\pi$, namely that $\rho\circ\pi$ is idempotent. This also proves that $\rho\circ\pi \in uco(C)$ that by [93] implies that $\pi\circ\rho = \rho\circ\pi$.

Therefore we have the following result.

Theorem 9.7. *Let* $\rho \in uco(\wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N}))$*, then* $\rho^{-\mathtt{T}} = \Pi_{\mathtt{T}} \circ\rho \in uco(\wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N}))$ *iff* $\Pi_{\mathtt{T}} \circ\rho\circ\Pi_{\mathtt{T}} = \rho\circ\Pi_{\mathtt{T}}$ *iff* $\Pi_{\mathtt{T}} \circ\rho\circ\Pi_{\mathtt{T}} = \Pi_{\mathtt{T}} \circ\rho$*. Analogously* $\rho^{-\mathtt{D}} = \Pi_{\mathtt{D}} \circ\rho \in uco(\wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N}))$ *iff* $\Pi_{\mathtt{D}} \circ\rho\circ\Pi_{\mathtt{D}} = \rho\circ\Pi_{\mathtt{D}}$ *iff* $\Pi_{\mathtt{D}} \circ\rho\circ\Pi_{\mathtt{D}} = \Pi_{\mathtt{D}} \circ\rho$*.*

In [65] it is provided a method for transforming $\Pi_{\mathtt{T}}$ in order to make it satisfy $\Pi_{\mathtt{T}} \circ\rho\circ\Pi_{\mathtt{T}} = \rho\circ\Pi_{\mathtt{T}}$ and $\Pi_{\mathtt{T}} \circ\rho\circ\Pi_{\mathtt{T}} = \Pi_{\mathtt{T}} \circ\rho$. However, since in this context the variable closure is ρ, we are more interested in modifying ρ in order to make it satisfy the given relations. In particular, this can be easily done for forward completeness.
Let $\mathtt{X} \in \{\mathtt{T}, \mathtt{D}\}$, consider the following transformations of ρ satisfying forward completeness, and, consider $\Pi^{+}_{\mathtt{x}}(X) \stackrel{\text{def}}{=} \bigcup \{\, Y \mid \Pi_{\mathtt{x}}(Y) \subseteq X \,\}$ which is well defined since $\Pi_{\mathtt{x}}$ is additive:

$$\rho^{\uparrow}_{\mathtt{X}}(Y) \stackrel{\text{def}}{=} \begin{cases} \Pi_{\mathtt{x}} \circ\rho(Y) & \text{if } Y \in \Pi_{\mathtt{x}} \\ \rho(Y) & \text{otherwise} \end{cases} \qquad \rho^{\downarrow}_{\mathtt{X}}(Y) \stackrel{\text{def}}{=} \begin{cases} \Pi^{+}_{\mathtt{x}} \circ\rho(Y) & \text{if } Y \in \Pi_{\mathtt{x}} \\ \rho(Y) & \text{otherwise} \end{cases}$$

Then we have that $\rho^{\downarrow}_{\mathtt{X}} \sqsubseteq \rho \sqsubseteq \rho^{\uparrow}_{\mathtt{X}}$. This transformation tells us that we can always transform the abstractions, used for modeling the attacker, is order to guarantee that the abstraction of time is a complete upper closure operator. Namely, given a generic ρ we always have that $(\rho^{\downarrow}_{\mathtt{X}})^{-\mathtt{T}}$ and $(\rho^{\uparrow}_{\mathtt{X}})^{-\mathtt{T}}$ are closure operators. For this reason we can suppose to consider closures η and ρ such that $\eta^{-\mathtt{T}}$ and $\rho^{-\mathtt{T}}$ are closures, and therefore we can, in general write:

$$[\eta]P^{+\mathtt{T}}(\rho) \;\Leftrightarrow\; [\eta^{-\mathtt{T}}]P^{+\mathtt{T}}(\rho^{-\mathtt{T}})$$

where $[\eta^{-\mathtt{T}}]P^{+\mathtt{T}}(\rho^{-\mathtt{T}})$ is narrow abstract non-interference (without time) for a semantics with time. The following results prove the relation existing between timed narrow abstract non-interference and narrow non-interference, as introduced in the previous sections.

Lemma 9.8. $\Pi_{\mathtt{T}} \circ\rho \neq \rho\circ\Pi_{\mathtt{T}}$ *implies* $\rho(X) = \rho(Y) \;\not\Rightarrow\; \rho^{-\mathtt{T}}(X) = \rho^{-\mathtt{T}}(Y)$ *and* $\rho^{-\mathtt{T}}(X) = \rho^{-\mathtt{T}}(Y) \;\not\Rightarrow\; \rho(X) = \rho(Y)$*.*

Proof. Suppose that $\Pi_{\mathtt{T}} \circ\rho \neq \rho\circ\Pi_{\mathtt{T}}$, this means that $\Pi_{\mathtt{T}} \circ\rho \neq \rho^{-\mathtt{T}}$. We prove that for each X $\Pi_{\mathtt{T}} \circ\rho(X) \supseteq \rho^{-\mathtt{T}}(X)$. Consider $\langle x, y\rangle \in \rho^{-\mathtt{T}}(X)$, this means that

$\exists z \in \mathbb{N} \,.\, \langle x, z\rangle \in \rho(X)$. But Π_{τ} is extensive, therefore $\langle x, z\rangle \in \Pi_{\tau} \circ\rho(X)$. By the properties of Π_{τ}, this implies that $\forall w \in \mathbb{N} \,.\, \langle x, w\rangle \in \Pi_{\tau} \circ\rho(X)$, and therefore in particular $\langle x, y\rangle \in \Pi_{\tau} \circ\rho(X)$. Namely for each X we have $\Pi_{\tau} \circ\rho(X) \subseteq \rho^{-\mathtt{T}}(X)$, namely $\Pi_{\tau} \circ\rho \sqsupseteq \rho^{-\mathtt{T}}$. Since the two closure are different, there exists an element X such that $\Pi_{\tau} \circ\rho(X) \subsetneq \rho^{-\mathtt{T}}(X)$. This means that $\exists\langle x, y\rangle \in \Pi_{\tau} \circ\rho(X)$ such that $\langle x, y\rangle \notin \rho^{-\mathtt{T}}(X)$. This implies that $\forall z \in \mathbb{N} \,.\, \langle x, z\rangle \notin X$. On the other hand $\langle x, y\rangle \in \Pi_{\tau} \circ\rho(X)$ implies that $\forall z \in \mathbb{N} \,.\, \langle x, z\rangle \in \Pi_{\tau} \circ\rho(X)$, which implies that $\exists\langle x, w\rangle \in \rho(X)$. By what we observed above we have also that $\langle x, w\rangle \notin X$, therefore $X \neq \rho(X)$. Let $Y = \rho(X)$, we have hence that $\rho(X) = \rho(Y)$ But $\rho^{-\mathtt{T}}(Y) = \Pi_{\tau} \circ\rho(X) \neq \rho^{-\mathtt{T}}(X)$.
We prove now that $\exists X, Y$ such that $\rho^{-\mathtt{T}}(X) = \rho^{-\mathtt{T}}(Y)$ but $\rho(X) \neq \rho(Y)$. If all the fix points of ρ would be of the kind $\bigcup\langle x, \mathbb{N}\rangle$, then $\Pi_{\tau} \circ\rho = \rho \in uco(\wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N}))$ which is absurd. Therefore there exists $X \in \rho$ such that $\exists x \in \mathbb{V}^{\mathtt{L}}$ such that $\langle x, y_1\rangle \in X$ and $\langle x, y_2\rangle \notin X$, for some $y_1, y_2 \in \mathbb{N}$, $y_1 \neq y_2$. But this means that $\rho(\langle x, y_1\rangle) \neq \rho(\langle x, y_2\rangle)$ since otherwise if $\rho(\langle x, y_1\rangle) = \rho(\langle x, y_2\rangle)$ then $\langle x, y_2\rangle \subseteq \rho(\langle x, y_2\rangle) = \rho(\langle x, y_1\rangle) \subseteq X$, which is absurd for the hypothesis made on X. On the other hand we have $\rho^{-\mathtt{T}}(\langle x, y_1\rangle) = \rho^{-\mathtt{T}}(\langle x, y_2\rangle) = \bigcap\{\, X \,|\, X \supseteq \langle x, \mathbb{N}\rangle \,\}$.

The following theorem says that the semantics for the timed notion of abstract non-interference is an abstract interpretation of the one for abstract non-interference with $\rho^{-\mathtt{T}}$ iff the output observation ρ commutes with Π_{τ}

Theorem 9.9. *Let $\eta, \rho \in uco(\wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N}))$, then ($[\eta]P^{+\mathtt{T}}(\rho) \;\Rightarrow\; [\eta^{-\mathtt{T}}]P^{+\mathtt{T}}(\rho^{-\mathtt{T}})$) if and only if we have ($\Pi_{\mathtt{T}} \circ\rho = \rho \circ\Pi_{\mathtt{T}}$).*

Proof. Consider $\eta, \rho \in uco(\wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N}))$ and a program P with time. In the following we will denote by π the closure $\Pi_{\mathtt{T}}$.

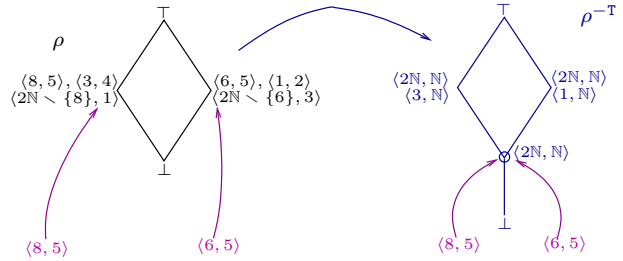
($\Leftarrow$) We have to prove that $[\eta]P^{+\mathtt{T}}(\rho)$ implies $[\eta^{-\mathtt{T}}]P^{+\mathtt{T}}(\rho^{-\mathtt{T}})$. Namely, we prove that $\eta(\widehat{l_1})^{\mathtt{L}} = \eta(\widehat{l_2})^{\mathtt{L}} \;\Rightarrow\; \rho([\![P]\!](\langle h_1, \widehat{l_1}\rangle)^{\mathtt{L}\,\mathtt{T}}) = \rho([\![P]\!](\langle h_2, \widehat{l_2}\rangle)^{\mathtt{L}\,\mathtt{T}})$ implies abstract non-interference with time, i.e., $\eta^{-\mathtt{T}}(\widehat{l_1}) = \eta^{-\mathtt{T}}(\widehat{l_2}) \;\Rightarrow\; \rho^{-\mathtt{T}}([\![P]\!](\langle h_1, \widehat{l_1}\rangle)^{\mathtt{L}\,\mathtt{T}}) = \rho^{-\mathtt{T}}([\![P]\!](\langle h_2, \widehat{l_2}\rangle)^{\mathtt{L}\,\mathtt{T}})$. It's worth noting that $\eta^{-\mathtt{T}}(\widehat{l_1}) = \eta^{-\mathtt{T}}(\widehat{l_2})$ implies $\eta(\widehat{l_1})^{\mathtt{L}} = \eta(\widehat{l_2})^{\mathtt{L}}$. Therefore, we prove that $\rho([\![P]\!](\langle h_1, \widehat{l_1}\rangle)^{\mathtt{L}\,\mathtt{T}}) = \rho([\![P]\!](\langle h_2, \widehat{l_2}\rangle)^{\mathtt{L}\,\mathtt{T}})$ implies $\rho^{-\mathtt{T}}([\![P]\!](\langle h_1, \widehat{l_1}\rangle)^{\mathtt{L}\,\mathtt{T}}) = \rho^{-\mathtt{T}}([\![P]\!](\langle h_2, \widehat{l_2}\rangle)^{\mathtt{L}\,\mathtt{T}})$. By the hypothesis we have $\Pi_{\tau} \circ\rho = \rho \circ\Pi_{\tau}$, namely $1\Pi_{\tau} \circ\rho = \rho^{-\mathtt{T}}$, consider now $X, Y \in \wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N})$, then $\rho(X) = \rho(Y)$ implies $\Pi_{\tau}(\rho(X)) = \Pi_{\tau}(\rho(X))$, namely $\rho^{-\mathtt{T}}(X) = \rho^{-\mathtt{T}}(Y)$. Therefore we have the wanted implication.

($\Rightarrow$) Let us prove that if $\Pi_{\tau} \circ\rho \neq \rho \circ\Pi_{\tau}$ then timed non-interference doesn't imply the abstract one. By Lemma 9.8 this implies that $\exists X, Y \in \wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N})$ such that $\rho(X) = \rho(Y) \;\not\Rightarrow\; \rho^{-\mathtt{T}}(X) = \rho^{-\mathtt{T}}(Y)$, and therefore, for what we said above, in general we have that timed non-interference does not imply abstract one.

Note that, adding time in abstract non-interference adds also new *deceptive flows*, indeed if the abstraction of the output is a property considering relations

between time and data, then timed abstract non-interference may fail even if the program is secure and avoid timing channels, as it happens in the following example.

Example 9.10. In this example, we show that in the timed abstract non-interference we add new deceptive flows due to the possible relation between data and time of the abstract property. Consider a property ρ such that $\rho(\langle 8,5\rangle) \neq \rho(\langle 6,5\rangle)$ (as depicted in the picture on the left, below) and $\eta = \langle Par, id\rangle$, which observes parity of data and the identity on time.



Consider the program fragment

$$P: \; h := 2; \; \textbf{while } h \textbf{ do } l := l + 2; \; h := h - 1; \textbf{endw}$$

Suppose $t_{\mathtt{T}} = 1$ and $t_{\mathtt{A}} = 0,5$. Consider the initial low values (data and time) $\langle 4,0\rangle$ and $\langle 2,0\rangle$, clearly we have $\eta(\langle 4,0\rangle) = \langle 2\mathbb{N}, 0\rangle = \eta(\langle 2,0\rangle)$. We have now to compute the semantics:

$$[\![P]\!](0,\langle 4,0\rangle)^{\mathtt{L}\,\mathtt{T}} = \langle 8, 3t_{\mathtt{T}} + 4t_{\mathtt{T}}\rangle = \langle 8,5\rangle$$

On the other hand

$$[\![P]\!](0,\langle 2,0\rangle)^{\mathtt{L}\,\mathtt{T}} = \langle 6,5\rangle$$

At this point, since $\rho(\langle 8,5\rangle) \neq \rho(\langle 6,5\rangle)$, non-interference is not satisfied, while there aren't timing information flows, namely $\rho^{-\mathtt{T}}(\langle 8,5\rangle) = \rho^{-\mathtt{T}}(\langle 6,5\rangle)$.

We conclude this section with a theorem that shows in which conditions timing channels are impossible in a given program.

Theorem 9.11. *If ρ commutes with $\Pi_{\mathtt{T}}$, i.e., $\Pi_{\mathtt{T}} \circ\rho = \rho \circ\Pi_{\mathtt{T}}$, and $[\eta^{-\mathtt{T}}]P^{+\mathtt{T}}(\rho^{-\mathtt{T}})$ iff $[\eta]P^{+\mathtt{T}}(\rho)$, then timing channels are impossible in P.*

Proof. By Theorem 9.9 we have that, if ρ commutes with $\Pi_{\mathtt{T}}$, i.e., $\Pi_{\mathtt{T}} \circ\rho = \rho \circ\Pi_{\mathtt{T}}$, then $[\eta]P^{+\mathtt{T}}(\rho)$ implies $[\eta^{-\mathtt{T}}]P^{+\mathtt{T}}(\rho^{-\mathtt{T}})$. We show that, if $[\eta^{-\mathtt{T}}]P^{+\mathtt{T}}(\rho^{-\mathtt{T}}) \Rightarrow [\eta]P^{+\mathtt{T}}(\rho)$, then timing channels are impossible. Indeed if this implication holds, it means that $\rho^{-\mathtt{T}}(X) = \rho^{-\mathtt{T}}(Y) \Rightarrow \rho(X) = \rho(Y)$. Namely $\Pi_{\mathtt{T}} \circ\rho(X) = \Pi_{\mathtt{T}} \circ\rho(Y)$, which means that $\forall x \in \mathbb{V}^{\mathtt{L}}\ \langle x, \mathbb{N}\rangle \in \Pi_{\mathtt{T}} \circ\rho(X)$ iff $\langle x, \mathbb{N}\rangle \in \Pi_{\mathtt{T}} \circ\rho(Y)$. This also means that $\forall x \in \mathbb{V}\ .\exists y_1, y_2 \in \mathbb{N}$ such that $\langle x, y_1\rangle \in \rho(X)$ iff $\langle x, y_2\rangle \in \rho(Y)$. But the fact that $\rho(X) = \rho(Y)$ implies also that $\langle x, y_1\rangle \in \rho(X)$ iff $\langle x, y_1\rangle \in \rho(Y)$ and therefore there are not timing channels.

Abstracting time in the input.

As we said above we can think of another way of erasing time, this is also suggested by the fact that we have two possible compositions of a closure with the projection $\Pi_{\mathtt{T}} : \Pi_{\mathtt{T}} \circ\rho$ and $\rho \circ\Pi_{\mathtt{T}}$, which are the same when they are closures. Anyway, the meaning of them is different, the first compute the property with time and abstract the observation, while the second abstracts time in the input value, namely compute the property on the abstracted value, without time. Let $L \subseteq \mathbb{V}^{\mathtt{L}}$, this second composition allows us to determine a closure on $\wp(\mathbb{V}^{\mathtt{L}})$ in the following way: $\rho_{-\mathtt{T}}(L) \stackrel{\text{def}}{=} \downarrow_{\mathtt{D}} \circ\rho \circ\Pi_{\mathtt{T}}(\langle L, \mathbb{N}\rangle)$, where $\langle L, \mathbb{N}\rangle \stackrel{\text{def}}{=} \{\langle x, y\rangle | x \in L,\ y \in \mathbb{N}\}$ and the abstraction $\downarrow_{\mathtt{D}}$ is the projection of the tuple on data, i.e., $\downarrow_{\mathtt{D}}(X) \stackrel{\text{def}}{=} \{x | \langle x, y\rangle \in X\}$.

In the following, we provide some results that allow to characterize when the composition $\downarrow_{\mathtt{D}} \circ\rho \circ\Pi_{\mathtt{T}}$ is an upper closure operator. This is important in order to derive property of narrow or abstract non-interference without time in programs where the semantics measures time, and therefore for understanding the relation existing between the notions of abstract non-interference with and without time.

Lemma 9.12. *Let $\eta \in uco(\wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N}))$. For each $L \in \wp(\mathbb{V}^{\mathtt{L}})$ the following facts are equivalent:*

- $\downarrow_{\mathtt{D}} \circ\eta \circ\Pi_{\mathtt{T}} \circ\eta(\langle L, \mathbb{N}\rangle) = \downarrow_{\mathtt{D}} \circ\eta \circ\Pi_{\mathtt{T}}(\langle L, \mathbb{N}\rangle)$
- $\downarrow_{\mathtt{D}} \circ\eta \circ\Pi_{\mathtt{T}}(\langle L, \mathbb{N}\rangle) = \downarrow_{\mathtt{D}} \circ\Pi_{\mathtt{T}} \circ\eta(\langle L, \mathbb{N}\rangle)$
- $\downarrow_{\mathtt{D}} \circ\eta \circ\Pi_{\mathtt{T}} = \downarrow_{\mathtt{D}} \circ\Pi_{\mathtt{T}} \circ\eta \circ\Pi_{\mathtt{T}}$.

Proof. Let us denote, for simplicity, $\pi \stackrel{\text{def}}{=} \Pi_{\mathtt{T}}$:

(1) $\Leftrightarrow$ (2) ($\Leftarrow$) Trivial by idempotence of η.
($\Rightarrow$) Note that $\downarrow_{\mathtt{D}} \circ\pi \circ\eta(\langle L, \mathbb{N}\rangle) \subseteq \downarrow_{\mathtt{D}} \circ\eta \circ\pi \circ\eta(\langle L, \mathbb{N}\rangle) = \downarrow_{\mathtt{D}} \circ\eta \circ\pi(\langle L, \mathbb{N}\rangle)$ (*), by extensivity of η and by hypothesis. Moreover, we have $\downarrow_{\mathtt{D}} \circ\eta \circ\pi(\langle L, \mathbb{N}\rangle) \subseteq \downarrow_{\mathtt{D}} \circ\pi \circ\eta \circ\pi(\langle L, \mathbb{N}\rangle)$, by extensivity of π, while, we have that $\downarrow_{\mathtt{D}} \circ\eta \circ\pi(\langle L, \mathbb{N}\rangle) = \downarrow_{\mathtt{D}} \circ\eta \circ\pi \circ\pi(\langle L, \mathbb{N}\rangle)$, by idempotence of π. Since $\forall L \in \wp(\mathbb{V}^{\mathtt{L}})$ we have $\pi(\langle L, \mathbb{N}\rangle) = (\langle L, \mathbb{N}\rangle)$, property (*) holds and therefore $\downarrow_{\mathtt{D}} \circ\eta \circ\pi \circ\pi(\langle L, \mathbb{N}\rangle) \supseteq \downarrow_{\mathtt{D}} \circ\pi \circ\eta \circ\pi(\langle L, \mathbb{N}\rangle)$. Then we proved $\downarrow_{\mathtt{D}} \circ\eta \circ\pi(\langle L, \mathbb{N}\rangle) = \downarrow_{\mathtt{D}} \circ\pi \circ\eta \circ\pi(\langle L, \mathbb{N}\rangle)$, therefore $\downarrow_{\mathtt{D}} \circ\eta \circ\pi(\langle L, \mathbb{N}\rangle) = \downarrow_{\mathtt{D}} \circ\pi \circ\eta \circ\pi(\langle L, \mathbb{N}\rangle) = \downarrow_{\mathtt{D}} \circ\pi \circ\eta(\langle L, \mathbb{N}\rangle)$.
(2) $\Leftrightarrow$ (3) The proofs comes trivially by the fact that $\pi(\langle L, \mathbb{N}\rangle) = \langle L, \mathbb{N}\rangle$.

Proposition 9.13. $\rho_{-\mathtt{T}} \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$ *iff* $\forall L \in \wp(\mathbb{V}^{\mathtt{L}}).\ \downarrow_{\mathtt{D}} \circ\rho \circ\Pi_{\mathtt{T}}(\langle L, \mathbb{N}\rangle) = \downarrow_{\mathtt{D}} \circ\Pi_{\mathtt{T}} \circ\rho(\langle L, \mathbb{N}\rangle)$, *i.e.,* $\downarrow_{\mathtt{D}} \circ\Pi_{\mathtt{T}} \circ\rho \circ\Pi_{\mathtt{T}} = \downarrow_{\mathtt{D}} \circ\rho \circ\Pi_{\mathtt{T}}$.

Proof. Let us denote $\pi \stackrel{\text{def}}{=} \Pi_{\mathtt{T}}$. Let us prove monotonicity. Consider $L_1, L_2 \in \wp(\mathbb{V}^{\mathtt{L}})$ such that $L_1 \subseteq L_2$. Then we have that $\langle L_1, \mathbb{N}\rangle \subseteq \langle L_2, \mathbb{N}\rangle$ and since $\eta \circ\pi$ is monotone being composition of monotone maps, we have:

$$\eta_{-\mathtt{T}}(L_1) = \downarrow_{\mathtt{D}} \circ\eta \circ\pi(\langle L_1, \mathbb{N}\rangle) \subseteq \downarrow_{\mathtt{D}} \circ\eta \circ\pi(\langle L_2, \mathbb{N}\rangle) = \eta_{-\mathtt{T}}(L_2)$$

Let us prove extensivity. Consider $L \in \wp(\mathbb{V}^{\mathtt{L}})$, then we can show that in general $\eta_{-\mathtt{T}}(L) = \downarrow_{\mathtt{D}} \circ\eta \circ\pi(\langle L, \mathbb{N}\rangle) \supseteq L$. Finally let us consider idempotence. Suppose that the two closure commutes:

$$\begin{aligned}\eta_{-\mathtt{T}} \circ\eta_{-\mathtt{T}}(L) &= \downarrow_{\mathtt{D}} \circ\eta \circ\pi(\langle\downarrow_{\mathtt{D}} \circ\eta \circ\pi(\langle L,\mathbb{N}\rangle),\mathbb{N}\rangle)\\ (*) &= \downarrow_{\mathtt{D}} \circ\eta \circ\pi \circ\eta \circ\pi(\langle L,\mathbb{N}\rangle)\\ &= \downarrow_{\mathtt{D}} \circ\eta \circ\pi \circ\eta(\langle L,\mathbb{N}\rangle)\\ &= \downarrow_{\mathtt{D}} \circ\eta \circ\pi(\langle L,\mathbb{N}\rangle) \text{ (by Lemma 9.12)}\\ &= \eta_{-\mathtt{T}}(L)\end{aligned}$$

where $X{\downarrow_{\mathtt{D}}} \stackrel{\text{def}}{=} \downarrow_{\mathtt{D}}(X)$ and the equality $(*)$ holds since it is trivial to check that, if $X \in \wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N})$, $\pi(\langle X{\downarrow_{\mathtt{D}}},\mathbb{N}\rangle) = \pi(X)$. On the other hand if the closure is idempotent, then the equalities above implies that the two closures commute.

Lemma 9.14. $\downarrow_{\mathtt{D}} \circ\rho \circ\pi \circ\rho \neq \downarrow_{\mathtt{D}} \circ\rho \circ\pi$ *implies* $\rho(X) = \rho(Y) \not\Rightarrow \rho_{-\mathtt{T}}(X{\downarrow_{\mathtt{D}}}) = \rho_{-\mathtt{T}}(Y{\downarrow_{\mathtt{D}}})$ *and* $\rho_{-\mathtt{T}}(X{\downarrow_{\mathtt{D}}}) = \rho_{-\mathtt{T}}(Y{\downarrow_{\mathtt{D}}}) \not\Rightarrow \rho(X) = \rho(Y)$.

Proof. Suppose $\downarrow_{\mathtt{D}} \circ\rho \circ\pi \circ\rho(X) \neq \downarrow_{\mathtt{D}} \circ\rho \circ\pi(X)$ for some $X \in \wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N})$. Consider $Y \stackrel{\text{def}}{=} \rho(X)$, then $\rho(X) = \rho(Y)$. Note that $\rho_{-\mathtt{T}}(X{\downarrow_{\mathtt{D}}}) = \downarrow_{\mathtt{D}} \circ\rho \circ\pi(\langle X{\downarrow_{\mathtt{D}}},\mathbb{N}\rangle) = \downarrow_{\mathtt{D}} \circ\rho \circ\pi(X)$. Similarly $\rho_{-\mathtt{T}}(Y{\downarrow_{\mathtt{D}}}) = \downarrow_{\mathtt{D}} \circ\rho \circ\pi(Y)$. Therefore, if $\rho_{-\mathtt{T}}(X{\downarrow_{\mathtt{D}}}) = \rho_{-\mathtt{T}}(Y{\downarrow_{\mathtt{D}}})$ then we obtain $\downarrow_{\mathtt{D}} \circ\rho \circ\pi(X) = \downarrow_{\mathtt{D}} \circ\rho \circ\pi(Y) = \downarrow_{\mathtt{D}} \circ\rho \circ\pi \circ\rho(X)$ which is absurd.
Consider now $Y = \pi(X)$, then we have $\rho_{-\mathtt{T}}(X{\downarrow_{\mathtt{D}}}) = \rho_{-\mathtt{T}}(Y{\downarrow_{\mathtt{D}}})$. Suppose, towards a contradiction, that $\rho(X) = \rho(Y)$, i.e., $\rho(X) = \rho \circ\pi(X)$ (*). Then $\downarrow_{\mathtt{D}} \circ\rho \circ\pi \circ\rho(X) = \downarrow_{\mathtt{D}} \circ\rho \circ\pi \circ\rho \circ\pi(X) = \downarrow_{\mathtt{D}} \circ\rho \circ\pi(X)$ for the relation (*) and for what we proved in Proposition 9.13.

The following theorem says that the semantics for the timed notion of abstract non-interference is an abstract interpretation of the one for abstract non-interference with $\rho_{-\mathtt{T}}$ iff the data projection commutes on elements closed under $\Pi_{\mathtt{T}}$.

Theorem 9.15. *Consider* $\eta, \rho \in uco(\wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N}))$ *and* $\eta_{-\mathtt{T}}, \rho_{-\mathtt{T}} \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$, $\downarrow_{\mathtt{D}} \circ\Pi_{\mathtt{T}} \circ\eta = \downarrow_{\mathtt{D}} \circ\eta \circ\Pi_{\mathtt{T}}$, *then we have* $[\eta]P^{+\mathtt{T}}(\rho) \Rightarrow [\eta_{-\mathtt{T}}]P(\rho_{-\mathtt{T}})$ *if and only if* $\downarrow_{\mathtt{D}} \circ\rho \circ\Pi_{\mathtt{T}} = \downarrow_{\mathtt{D}} \circ\rho \circ\Pi_{\mathtt{T}} \circ\rho$.

Proof. Consider $\eta, \rho \in uco(\wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N}))$, a program P with time, and $\pi \stackrel{\text{def}}{=} \Pi_{\mathtt{T}}$.

($\Leftarrow$) Note that

$$\eta_{-\mathtt{T}}(l) = \downarrow_{\mathtt{D}} \circ\eta \circ\pi(\langle l,\mathbb{N}\rangle) = \downarrow_{\mathtt{D}} \circ\eta \circ\pi(\langle l,0\rangle) = \downarrow_{\mathtt{D}} \circ\pi \circ\eta(\langle l,0\rangle) = \eta(\langle l,0\rangle)^{\mathtt{L}},$$

since $\downarrow_{\mathtt{D}} \circ\pi \circ\eta = \downarrow_{\mathtt{D}} \circ\eta \circ\pi$. This means that the premixes are the same. We prove that, $\eta(l_1,0)^{\mathtt{L}} = \eta(l_2,0)^{\mathtt{L}} \Rightarrow \rho([\![P]\!](h_1,l_1,0)^{\mathtt{L}\,\mathtt{T}}) = \rho([\![P]\!](h_2,l_2,0)^{\mathtt{L}\,\mathtt{T}})$ implies $\rho_{-\mathtt{T}}([\![P]\!](h_1,l_1)^{\mathtt{L}}) = \rho_{-\mathtt{T}}([\![P]\!](h_2,l_2)^{\mathtt{L}})$. We prove something stronger, namely we prove that $\rho(X) = \rho(Y)$ implies $\rho_{-\mathtt{T}}(X{\downarrow_{\mathtt{D}}}) = \rho_{-\mathtt{T}}(Y{\downarrow_{\mathtt{D}}})$ for any $X, Y \in \wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N})$.

$$\begin{aligned}\rho_{-\mathtt{T}}(X{\downarrow_{\mathtt{D}}}) &= \downarrow_{\mathtt{D}} \circ\rho \circ\pi(\langle X{\downarrow_{\mathtt{D}}},\mathbb{N}\rangle)\\ &= \downarrow_{\mathtt{D}} \circ\rho \circ\pi(X) = \downarrow_{\mathtt{D}} \circ\rho \circ\pi \circ\rho(X)\\ &= \downarrow_{\mathtt{D}} \circ\rho \circ\pi \circ\rho(Y) = \downarrow_{\mathtt{D}} \circ\rho \circ\pi(Y)\\ &= \downarrow_{\mathtt{D}} \circ\rho \circ\pi(\langle Y{\downarrow_{\mathtt{D}}},\mathbb{N}\rangle)\\ &= \rho_{-\mathtt{T}}(Y{\downarrow_{\mathtt{D}}})\end{aligned}$$

noting that $X{\downarrow_{\mathtt{D}}} = X^{\mathtt{L}}$ if $X \in \wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N})$.

($\Rightarrow$) Let us prove that if $\downarrow_{\mathtt{D}} \circ\rho \circ\pi \circ\rho \neq \downarrow_{\mathtt{D}} \rho \circ\pi$ then timed non-interference doesn't imply abstract non-interference. By Lemma 9.14 this implies that there exist $X, Y \in \wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N})$ such that $\rho(X) = \rho(Y) \not\Rightarrow \rho_{-\mathtt{T}}(X) = \rho_{-\mathtt{T}}(Y)$, and therefore, for what we said above, in general we have that timed non-interference does not imply abstract one.

Non-relational attackers.

A sufficient condition, in order to make the non-interference notions comparable, consists in considering only closures defined on $\wp(\mathbb{V}^{\mathtt{L}}) \times \wp(\mathbb{N})$, which are particular closures of $\wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N})$. These two domains are related by the Galois insertion $\alpha : \wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N}) \longrightarrow \wp(\mathbb{V}^{\mathtt{L}}) \times \wp(\mathbb{N})$ and $\gamma : \wp(\mathbb{V}^{\mathtt{L}}) \times \wp(\mathbb{N}) \longrightarrow \wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N})$ defined as follows:

$$\begin{array}{l} \alpha(X) \stackrel{\text{def}}{=} \langle \{\, x \,|\, \langle x, y\rangle \in X \,\}, \{\, y \,|\, \langle x, y\rangle \in X \,\}\rangle \stackrel{\text{def}}{=} \langle X{\downarrow_{\mathtt{D}}}, X{\downarrow_{\mathtt{T}}}\rangle \\ \gamma(\langle X, Y\rangle) \stackrel{\text{def}}{=} \{\, \langle x, y\rangle \,|\, x \in X,\ y \in Y \,\} \end{array}$$

Proposition 9.16. *Consider $\rho \in uco(\wp(A) \times \wp(B))$, then there exist two closures $\rho_A \in uco(\wp(A))$ and $\rho_B \in uco(\wp(B))$ such that $\rho = \langle \rho_A, \rho_B\rangle$, namely such that $\forall \langle X, Y\rangle \in \wp(A) \times \wp(B)$. $\rho(\langle X, Y\rangle) = \langle \rho_A(X), \rho_B(Y)\rangle$.*

Proof. Consider $\rho \in uco(\wp(A) \times \wp(B))$, and the closure $\rho_A \stackrel{\text{def}}{=} \{\, X \,|\, \langle X, Y\rangle \in \rho \,\}$, on the first component, and the closure $\rho_B \stackrel{\text{def}}{=} \{\, Y \,|\, \langle X, Y\rangle \in \rho \,\}$ on the second one. We first show that these domains are Moore families respectively of $\wp(A)$ and of $\wp(B)$. Consider $X_1, X_2 \in \rho_A$, then by definition there exist $Y_1, Y_2 \in B$ such that $\langle X_1, Y_1\rangle, \langle X_2, Y_2\rangle \in \rho$. But ρ is a closure, therefore $\langle X_1, Y_1\rangle \cap \langle X_2, Y_2\rangle \in \rho$. But we have that $\langle X_1, Y_1\rangle \cap \langle X_2, Y_2\rangle \in \rho = \langle X_1 \cap X_2, Y_1 \cap Y_2\rangle$ and therefore we have $X_1 \cap X_2 \in \rho_A$. Analogously we can prove that also ρ_B is a Moore family. At this point it is trivial to verify that $\rho(\langle X, Y\rangle) = \langle \rho_A(X), \rho_B(Y)\rangle$.

Note that each closure on $\wp(\mathbb{V}^{\mathtt{L}}) \times \wp(\mathbb{N})$ is an abstraction of $\gamma \circ\alpha(\wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N})$, since it has its fix points in the domain $\wp(\mathbb{V}^{\mathtt{L}}) \times \wp(\mathbb{N}))$. Consider $\rho \in uco(\wp(\mathbb{V}^{\mathtt{L}}) \times \wp(\mathbb{N})))$, we can obtain a closure $\rho^* \in uco(\wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N}))$ in the following way: $\rho^* \stackrel{\text{def}}{=} \gamma \circ\rho \circ\alpha$.

Proposition 9.17. *Let $\rho \in uco(\wp(\mathbb{V}^{\mathtt{L}}) \times \wp(\mathbb{N}))$, consider $\rho^* \in uco(\wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N}))$ defined $\rho^* \stackrel{def}{=} \gamma \circ\rho \circ\alpha$, then we have $\Pi_{\mathtt{T}} \circ\rho^* = \rho^* \circ\Pi_{\mathtt{T}}$ and $\Pi_{\mathtt{D}} \circ\rho^* = \rho^* \circ\Pi_{\mathtt{D}}$.*

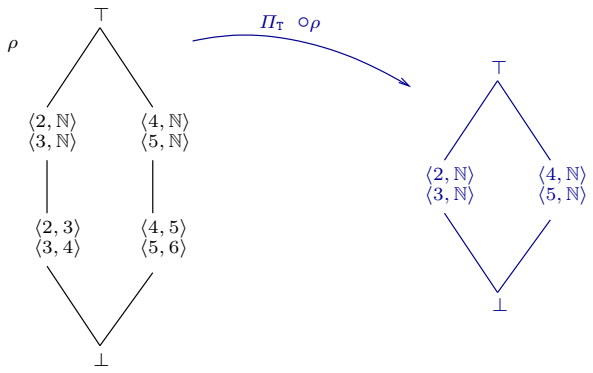
Proof. Let us prove the thesis only with $\Pi_{\mathtt{T}}$, the other proof is similar. Consider $X \in \wp(\mathbb{V}^{\mathtt{L}} \times \mathbb{N})$ and $\rho \in uco(\wp(\mathbb{V}^{\mathtt{L}}) \times \wp(\mathbb{N}))$ such that $\rho(\langle X, Y\rangle) = \langle \rho_{\mathtt{D}}(X), \rho_{\mathtt{T}}(Y)\rangle$:

$$\begin{array}{l} \Pi_{\mathtt{T}} \circ\rho^*(X) = \Pi_{\mathtt{T}} \circ\gamma \circ\rho(\langle X{\downarrow_{\mathtt{D}}}, X{\downarrow_{\mathtt{T}}}\rangle) = \Pi_{\mathtt{T}} \circ\gamma(\langle \rho_{\mathtt{D}}(X{\downarrow_{\mathtt{D}}}), \rho_{\mathtt{T}}(X{\downarrow_{\mathtt{T}}})\rangle \\ \quad = \Pi_{\mathtt{T}}(\{\, \langle x, y\rangle \,|\, x \in \rho_{\mathtt{D}}(X{\downarrow_{\mathtt{D}}}),\ y \in \rho_{\mathtt{T}}(X{\downarrow_{\mathtt{T}}}) \,\}) \\ \quad = \{\, \langle x, y\rangle \,|\, x \in \rho_{\mathtt{D}}(X{\downarrow_{\mathtt{D}}}),\ y \in \mathbb{N} \,\} \end{array}$$

On the other hand we have

$$\begin{aligned}\rho^* \circ \Pi_{\mathtt{T}}(X) &= \gamma \circ \rho \circ \alpha(\{\ \langle x, y\rangle \,|\, x \in X{\downarrow_{\mathtt{D}}},\ y \in \mathbb{N}\ \}) \\ &= \gamma \circ \rho(\langle X{\downarrow_{\mathtt{D}}}, \mathbb{N}\rangle) = \gamma(\langle \rho_{\mathtt{D}}(X{\downarrow_{\mathtt{D}}}), \mathbb{N}\rangle) = \{\ \langle x, y\rangle \,|\, x \in \rho_{\mathtt{D}}(X{\downarrow_{\mathtt{D}}}),\ y \in \mathbb{N}\ \}\end{aligned}$$

Therefore, by Theorem 9.9 and Theorem 9.15, this means that in the conditions of the Proposition above, timed abstract non-interference implies abstract non-interference.
This is only a sufficient condition, since there are closures $\rho \notin uco(\wp(\mathbb{V}^{\mathtt{L}}) \times \wp(\mathbb{N})$ such that $\Pi_{\mathtt{T}} \circ \rho = \rho^{-\mathtt{T}}$ or $\Pi_{\mathtt{D}} \circ \rho = \rho^{-\mathtt{D}}$. We can see an example in the following picture, where we denote by $\langle n, \mathbb{N}\rangle \stackrel{\text{def}}{=} \{\ \langle n, m\rangle \,|\, m \in \mathbb{N}\ \}$.



In particular in the example above we can also note that $\Pi_{\mathtt{T}} \circ \rho = \Pi_{\mathtt{T}} \sqcup \rho$.

9.4 Discussion

In this chapter, we extend abstract non-interference in order to check even timing channels, namely those channels of informations created by the capability of the attacker to observe the time elapsed during computation. We obtain, in this way the notion of timed abstract non-interference, which is based on a trace semantics observing time, and considers time as a public variable. Afterwards, we study the relation between this new notion and the abstract non-interference defined in the previous chapters. This is the last example, in this thesis, that shows how, changing the semantics, we can change the enforced notion of non-interference. In the same way, we would like to define a *probabilistic* abstract non-interference, where we can also check probabilistic channels and where it would be interesting to check non-interference considering also properties of the probabilistic distribution of values.

Enriching of the semantics is not sufficient in order to cope with notions of non-interference defined on systems different from imperative languages, such as process algebras and timed automata. This also because the given notion of non-interference is based on the notion of *variable*, that is significant only in programming languages, and uses denotational semantics, that cannot model all the

computational systems. Moreover, in literature, there exist different notions of non-interference that model the confidentiality problem in systems modeled by finite state automata. In particular, while when considering the semantics, we allow the private to interfere with the public as long as this interference is not visible observing the output, in process algebras, for example, non-interference is not violated when private actions do not interfere with the sequence of public actions at all. In order to define a uniform setting for defining abstract non-interference in different computational systems we have to further abstract our point of view, obtaining the generalized abstract non-interference introduced in the following chapter.

10

Generalized Abstract Non-Interference

The real voyage of discovery consists not in seeking new lands,
but in seeing with new eyes.
Marcel Proust

In the previous chapters, the notion of abstract non-interference has been introduced only for imperative programs, modeled by using a denotational or a trace semantics and where the security policy states that information about the initial values of *private variables* has not to be revealed through the observation of the output of *public variables*. Clearly non-interference is a more general notion and can be applied in any system where there is a distinction between two (or more) classes of users, and one class has not to *interfere* with the others [69]. Indeed, as we have seen in Chap. 5, non-interference has been defined in different ways and for different systems. The problem is that all these notions are not comparable and share only the intuitive definition of non-interference, being based on different computational systems. In particular, in the previous chapters, we describe non-interference by using the notion of *variable*, but clearly we cannot speak of variables when we consider computational systems different from programming languages, such as process algebras or timed automata. In this case, the notion of non-interference has been defined by considering *actions* and by saying that the sequence of visible/public actions has not to be altered by the execution of private actions [47,12]. More precisely, the notion of non-interference generally used in this kind of systems says that the computations where private actions are *hidden* have to be equivalent (w.r.t. some given relation) to the computations where private actions are *avoided*.

In order to describe all these notions of non-interference in the same formalism we have, first, to find a model of computation that allows to describe all these computational systems. For this reason, we note that in sequential systems the *private actions* are the actions of writing on the private memory, i.e., the actions

of modifying private variables, while the public actions are the actions of reading the public memory, i.e., of reading of public variables' values. Clearly, in this case, the private actions have also a qualitative aspect which is the value written in the memory. For this reason, the requirement that the executions where high-level actions are hidden have to be the same as the executions where high-level actions are avoided, would be clearly too strong. In particular, this would mean that, if we take the program P, and we erase all the statements that execute a private action (i.e., modify a private variable), then we obtain a *slice* of the original program. This exactly means that all the low-level actions do not depend on the high-level ones. When observing values, this is too strong since we can admit that the observer knows that the private memory has been modified unless it is not able to get information about the initial value of private memory. For this reason, in sequential systems, we consider the abstract non-interference as introduced in Sect. 6.1.

At this point, in order to consider a more general setting, we decide to model also programs with labeled transition systems, as described in Sect. 4.1.1. Therefore, the model that we are going to use for modeling generic computational systems are labeled transition systems and the corresponding computational trees. By considering this new context, we prove that abstract non-interference introduced in Sect. 6.1 can be generalized in order to cope with many well-known models of secrecy in sequential, concurrent and real-time systems and languages. This is achieved by factoring abstractions in order to identify sub-abstractions modeling the different properties of the system on which the notions of non-interference are based. Abstract interpretation [28] and the theory of abstract domain transformers [31, 61] plays a key role in this generalization: The abstraction represents here both what an attacker may observe about a computation (as in abstract non-interference) and which aspects of the computation are relevant for checking non-interference, depending on the specific notion of non-interference that we have to enforce. Therefore, non-interference corresponds to asking that the behavior of the chosen relevant aspects of the computation is independent from what an attacker may observe. In particular, we consider three abstractions: The first decides the model of computation for which we are defining non-interference, e.g., denotational semantics; The second decides which aspects of computations the attackers can observe, e.g., the computations where private actions are avoided; Finally, the third one, decides the observational capability of the attacker. By composing these three abstractions we obtain generalized abstract non-interference (shortly GANI). At this point, we prove that both narrow and abstract non-interference are instances of this generalized abstract non-interference. Then we prove that NNI (Non-deterministic Non-Interference), SNNI (Strong NNI), NDC (Non-Deducibility on Compositions), BNDC (Bisimulation NDC), BNNI (Bisimulation NNI), and BSNNI (Bisimulation SNNI) in [47] (see Sect. 5.3.3) for Security Process Algebras (SPA), are all instances of GANI. Finally, we prove that decidable notions of non-interference introduced for timed automata in [12] (see Sect. 5.3.4)

are again instances of GANI. In all these constructions, the model of an attacker is specified as an abstract interpretation of the system semantics. This is a key point in order to introduce systematic methods for deriving attackers by transforming abstract domains. We generalize the method introduced in Sect. 6.3 to derive harmless attackers for GANI, i.e., abstractions of the semantics of systems which guarantee non-interference. This section is based on the unpublished paper [53].

10.1 Generalized Abstract Non-Interference

In this section, we introduce a generalization of abstract non-interference, called *generalized abstract non-interference* (shortly GANI), which subsumes many of the known notions of non-interference based on tree-like computations and automata. Abstract interpretation plays a key role in this generalization: The abstraction represents here both what an attacker may observe about a computation (as in abstract non-interference) and which aspects of the computation are relevant for checking non-interference, aspects determined by the specific notion of non-interference that we have to enforce on the system. Non-interference corresponds to asking that relevant aspects of the private computation have no effects on what an attacker may observe of the computation. Moreover, what an attacker may observe is indeed composed by two aspects: what the particular notion of non-interference allows to observe, and what effectively the attacker can observe. We consider, as concrete semantics the tree semantics defined in Sect. 4.1, denoted as $\{\!|\cdot|\!\}$. Let us consider a system P and its semantics $\{\!|P|\!\}$. We define generalized non-interference by using three abstractions, each one with a specific and precise meaning, depending on the given notion of non-interference, and depending on the attacker model. The notion of non-interference decides two of these abstractions:

α_{OBS}: The first abstraction α_{OBS} abstracts the computational tree, modeling the system, in the model used in the notion of non-interference that has to be enforced. For instance, if we want to check standard non-interference for imperative programming languages, then, given a program, α_{OBS} corresponds to the abstraction that derives the denotational semantics from the computational tree modeling the program. We call this abstraction the *observation abstraction*.

α_{INT} The second abstraction is α_{INT}, which characterizes, in the given notion of non-interference, which should be the maximal amount of information that an attacker may observe. For example, if we have to check non-interference in SPA, then we want the computations where private actions are hidden to be equivalent to the computations where private actions are avoided. Namely the set of all the computations where private actions are avoided should be the maximal information that the attacker can observe, therefore in this case α_{INT} selects only those computations where private actions are not executed.

This abstraction, called *interference abstraction*, forgets about all information which should not be observed by an attacker.

These two abstractions tells us that, in general, non-interference holds whenever the amount of information that an attacker can grasp from a computation is precisely what, for the given notion of non-interference, that attacker *can* observe about it.
Finally, we have to model the observational capability of the attacker, for this reason we consider a further abstraction α_{ATT}, called the *attacker abstraction*, which characterizes what the model of the attacker can observe about the system behavior. By using these three abstractions we define generalized abstract non-interference in the following equation, where P is a system.

$$\boxed{\alpha_{ATT} \circ \alpha_{OBS}(\{\!|P|\!\}) = \alpha_{ATT} \circ \alpha_{INT} \circ \alpha_{OBS}(\{\!|P|\!\})}$$

This equation says that, in the model chosen by α_{OBS}, the maximal information that the attacker *is allowed to* observe, determined by $\alpha_{ATT} \circ \alpha_{INT}$, is exactly what the attacker *does* observe, determined by α_{ATT}. We show that this is a very general formalization and that many of the known notions of non-interference can be obtained as instances of this definition. In Fig. 10.1 we have a graphical representation of the equation above.

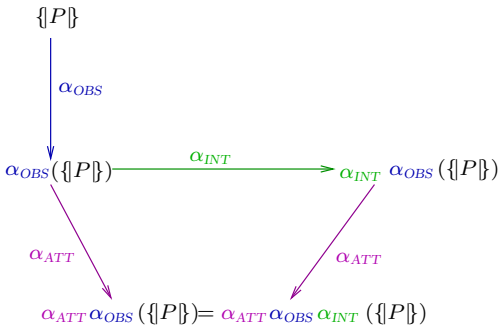


Fig. 10.1. The global picture.

10.1.1 Deriving GANI attackers

The advantage of specifying different notions of non-interference for sequential, concurrent and timed systems as GANI relies upon the possibility offered by abstract interpretation to systematically derive abstractions. As observed in Sect. 6.3, deriving abstractions in abstract non-interference corresponds precisely to derive models of attackers. In this section, we generalize this construction to GANI.

Let P be a system and consider the observation, interference, and attacker abstractions for non-interference: α_{OBS}, α_{INT} and α_{ATT}, such that the system P is not

secure. As we said, α_{OBS} and α_{INT} depend on the definition of non-interference that we choose, while α_{ATT} depends on what we decide to observe about the computation. Therefore, if non-interference is not satisfied, i.e., the system is not secure under the chosen notion of non-interference, then we can think of further abstracting the attacker abstraction in order to achieve security. The resulting abstraction provides a certificate of the security level of the system P with respect to the fixed observation and interference abstractions. In order to find the most concrete abstraction α_{ATT} that makes equal the sets $\alpha_{ATT} \circ \alpha_{OBS}$ and $\alpha_{ATT} \circ \alpha_{INT} \circ \alpha_{OBS}$ we have to merge elements in both sets in order to make them containing the same new abstract objects.

It is worth noting that the definition of GANI in general is characterized by a *possibilistic* interpretation of equality. This means that in order to make GANI hold, it is sufficient that the sets of abstract objects resulting from the abstractions are the same. This is in general a possibilistic definition and it doesn't provide a criterion for collecting elements of these sets in order to make their abstraction the same. On the other hand, from the notion of abstract non-interference seen in Sect. 6.1, we know that all the computations with the same public input has to provide the same results. In this case, there is a clear criterion for collecting elements in order to build the abstraction: we have to abstract in the same object all the elements resulting from computations that differ only for private inputs. We want to find a similar construction for GANI. We can think of generalizing the construction collecting elements that share a common maximal partial execution. Consider the tree of computations $\{\!|A|\!\}$ of the system A. Recall that when the system A fails GANI, we have $\alpha_{ATT} \circ \alpha_{OBS}(\{\!|A|\!\}) \leq \alpha_{ATT} \circ \alpha_{INT} \circ \alpha_{OBS}(\{\!|A|\!\})$.

$$\boxed{\begin{array}{c}\text{A system } A \text{ is } \textit{secure} \text{ if } \forall \sigma \in \alpha_{INT} \circ \alpha_{OBS}(\{\!|A|\!\}), \forall \delta \in \alpha_{OBS}(\{\!|A|\!\})\ . \\ \delta \preccurlyeq_{\max} \sigma \ \Rightarrow\ \alpha_{ATT}(\delta) = \alpha_{ATT}(\sigma)\end{array}}$$

where the relation $\preccurlyeq_{\max}$, specifies the maximal subtree which δ shares with σ. The definition above clearly, depends on the subtree relation $\preccurlyeq$. Consider the tree $\sigma \in \alpha_{INT} \circ \alpha_{OBS}(\{\!|A|\!\})$ and consider $\delta \in \alpha_{OBS}(\{\!|A|\!\})$: $\delta \preccurlyeq_{\max} \sigma$ if

$$\exists \pi \preccurlyeq \delta\ .\ \pi \preccurlyeq \sigma\ \wedge \forall \pi' \neq \pi\ .\ \pi \preccurlyeq \pi' \preccurlyeq \delta, \forall \sigma' \in \alpha_{INT} \circ \alpha_{OBS}(\{\!|A|\!\}) \text{ then } \pi' \not\preccurlyeq \sigma'$$

Note that the security condition given above is in general stronger than GANI. It is clear that, at this point, we can define a family of generalized non-interferences, depending on how we define the relation $\preccurlyeq_{\max}$, with the constraint that the definition for sequential systems has to collapse to abstract non-interference. We use the relation above for defining the sets of objects that need to have the same abstraction in order to achieve secrecy: $\forall \sigma \in \alpha_{INT} \circ \alpha_{OBS}(\{\!|A|\!\})$

$$\Upsilon(\sigma) = [\sigma] \stackrel{\text{def}}{=} \left\{\ \delta \in \alpha_{OBS}(\{\!|A|\!\}) \,\middle|\, \delta \preccurlyeq_{\max} \sigma\ \right\}$$

Example 10.1. Consider a system A, and suppose that

$$\alpha_{OBS}(\{\!|A|\!\}) = \{1 \to 2 \to 3, 1 \to 2 \to 4, 1 \to 3 \to 2\}$$
$$\alpha_{INT} \circ \alpha_{OBS}(\{\!|A|\!\}) = \{1 \to 2 \to 3, 1 \to 3 \to 2\}$$

then we have that $[1 \to 2 \to 3] = \{1 \to 2 \to 3, 1 \to 2 \to 4\}$ and $[1 \to 3 \to 2] = \{1 \to 3 \to 2\}$. If, instead, we have that

$$\begin{aligned}\alpha_{OBS}(\{\!|A|\!\}) = \{&1 \to 2 \to 4, 1 \to 2 \to 3, 1 \to 5 \to 3, 3 \to 5 \to 4,\\ &1 \to 2 \to 5, 1 \to 3 \to 2, 3 \to 2 \to 1\}\\ \alpha_{INT} \circ \alpha_{OBS}(\{\!|A|\!\}) = \{&1 \to 2 \to 3, 1 \to 3 \to 2, 3 \to 2 \to 1\}\end{aligned}$$

therefore $[1 \to 2 \to 3] = \{1 \to 5 \to 3, 1 \to 2 \to 3, 1 \to 2 \to 4\}$ and $[1 \to 3 \to 2] = \{1 \to 5 \to 3, 1 \to 3 \to 2\}$. Finally, $[3 \to 2 \to 1] = \{3 \to 2 \to 1, 3 \to 5 \to 4\}$. These sets represent what we have to abstract into a unique abstract element in order to achieve GANI.

The definition above is based on the observation that if a *computation* has a maximal partial computation in common with what can be surely observed by the attacker, then it is in those points, where the common partial computations end, that some private action has interfered in the computation. Similarly to what we have done in Sect. 6.3, we define the set $\mathbb{D}_{\{\!|A|\!\}}$ collecting all computations that may induce a failure of secrecy, and the set $Irr_{\{\!|A|\!\}}$, collecting all computations for which secrecy cannot fail.

$$\begin{aligned}\mathbb{D}_{\{\!|A|\!\}} &= \left\{\ [\sigma] \,\middle|\, \sigma \in \alpha_{INT} \circ \alpha_{OBS}(\{\!|A|\!\})\ \right\}\\ Irr_{\{\!|A|\!\}} &= \left\{\ X \,\middle|\, \forall \sigma \in \alpha_{INT} \circ \alpha_{OBS}(\{\!|A|\!\})\,.\, X \notin \uparrow([\sigma])\ \right\}\end{aligned}$$

The predicate $Secr_{\{\!|P|\!\}}$ defined on programs P, is now generalized to any computational system A:

$$Secr_{\{\!|A|\!\}}(X) \text{ iff } \forall \sigma \in \alpha_{INT} \circ \alpha_{OBS}(\{\!|A|\!\})\,.\,(\exists Z \in [\sigma]\,.\, Z \subseteq X \ \Rightarrow \forall W \in [\sigma]\,.\, W \subseteq X)$$

By the construction in Sect. 6.3, we can prove that $\mathcal{S}\{\!|A|\!\} \stackrel{\text{def}}{=} \left\{\ X \,\middle|\, Secr_{\{\!|A|\!\}}(X)\ \right\}$ is the most concrete abstraction that enforces the notion of GANI to hold, w.r.t. the relation $\preccurlyeq_{\max}$.

Theorem 10.2. *$\mathcal{S}\{\!|A|\!\}$ is the most concrete abstract domain such that, given a system A, then $\forall \sigma \in \alpha_{INT} \circ \alpha_{OBS}(\{\!|A|\!\}), \forall \delta \in \alpha_{OBS}(\{\!|A|\!\})\,.\,\delta \preccurlyeq_{\max} \sigma \ \Rightarrow\ \mathcal{S}\{\!|A|\!\}(\delta) = \mathcal{S}\{\!|A|\!\}(\sigma)$.*

Proof. The proof that $\mathcal{S}\{\!|A|\!\}$ is a Moore family comes directly from the definition of $Secr_{\{\!|A|\!\}}$ and it is straightforward.
Let us prove that $\mathcal{S}\{\!|A|\!\}$ satisfies the given notion of non-interference. By construction it is composed only by secret sets. Consider $\sigma \in \alpha_{ATT} \circ \alpha_{INT} \circ \alpha_{OBS}(\{\!|A|\!\})$, and $\delta \in [\sigma]$, clearly we have that $\mathcal{S}\{\!|A|\!\}(\delta) \supseteq [\sigma]$ and $\mathcal{S}\{\!|A|\!\}(\sigma) \supseteq [\sigma]$ since they are both secret. We have to prove that they are the same. Suppose that $\mathcal{S}\{\!|A|\!\}(\delta) \not\supseteq \mathcal{S}\{\!|A|\!\}(\sigma)$, then $\sigma \not\subseteq \mathcal{S}\{\!|A|\!\}(\delta)$ (otherwise, by monotonicity and idempotence we would have $\mathcal{S}\{\!|A|\!\}(\delta) \supseteq \mathcal{S}\{\!|A|\!\}(\sigma)$) and $\delta \subseteq \mathcal{S}(\{\!|A|\!\})(\delta)$ by extensivity, therefore we have an absurd since $\mathcal{S}(\{\!|A|\!\})(\delta)$ is secret. When $\mathcal{S}\{\!|A|\!\}(\delta) \not\subseteq \mathcal{S}\{\!|A|\!\}(\sigma)$ the proof is similar.

Finally we prove that it is the most concrete. Suppose that we have a closure $\rho \sqsubseteq \mathcal{S}\{\!|A|\!\}$ that makes the system A secret. Clearly there exists $X \in \rho$ such that $\neg \mathit{Secr}_{\{\!|A|\!\}}(X)$. This means that $\exists \sigma \in \alpha_{ATT} \circ \alpha_{INT} \circ \alpha_{OBS}(\{\!|A|\!\})$ such that $\exists \delta \in [\sigma]\,.\,\delta \in X$ and $\exists \delta' \in [\sigma]\,.\,\delta' \notin X$. Suppose $\sigma \notin X$, then we have that $\rho(\delta) \subseteq X$ and $\rho(\sigma) \not\subseteq X$, namely $\rho(\delta) \neq \rho(\sigma)$. On the other hand if $\sigma \in X$ then $\rho(\sigma) \subseteq X$ and $\rho(\delta') \not\subseteq X$, therefore again we have $\rho(\delta') \neq \rho(\sigma)$, absurd for the hypothesis on ρ.

Finally, we prove that $\mathcal{S}(\{\!|A|\!\})$ is $\mathcal{S}(\uparrow(\mathbb{D}_{\{\!|A|\!\}})) \cup \mathit{Irr}_{\{\!|A|\!\}}$.

Proposition 10.3. $\mathcal{S}(\{\!|A|\!\}) = \mathcal{S}(\uparrow(\mathbb{D}_{\{\!|A|\!\}})) \cup \mathit{Irr}_{\{\!|A|\!\}}$.

Proof. The inclusion $\supseteq$ is straightforward. We prove that $\mathcal{S}(\{\!|A|\!\}) \subseteq \mathcal{S}(\uparrow(\mathbb{D}_{\{\!|A|\!\}})) \cup \mathit{Irr}_{\{\!|A|\!\}}$. Consider $X \in \mathcal{S}(\{\!|A|\!\})$ and $X \notin \mathcal{S}(\uparrow(\mathbb{D}_{\{\!|A|\!\}})) \cup \mathit{Irr}_{\{\!|A|\!\}}$. Then $X \notin \mathcal{S}(\uparrow(\mathbb{D}_{\{\!|A|\!\}}))$ and $X \notin \mathit{Irr}_{\{\!|A|\!\}}$. The first condition holds iff $\neg \mathit{Secr}_{\{\!|A|\!\}}(X)$ (absurd for the hypothesis on X) or $X \notin \uparrow(\mathbb{D}_{\{\!|A|\!\}})$. Therefore we summarize the condition on X: $\mathit{Secr}_{\{\!|A|\!\}}(X)$, $X \notin \mathit{Irr}_{\{\!|A|\!\}}$ and $X \notin \uparrow(\mathbb{D}_{\{\!|A|\!\}})$. The second condition implies that $\exists \sigma\,.\,\delta \in [\sigma]\,.\,\delta \in X$. Namely we can conclude that $X \not\supseteq [\sigma]$ since $X \notin \uparrow(\mathbb{D}_{\{\!|A|\!\}})$, but this would mean that X is not secret, which is absurd.

10.1.2 Abstract non-interference as GANI

In this section, we prove that abstract non-interference, introduced in Sect. 6.1, is an instance of the generalized abstract non-interference. Recall that, if P is a program, its denotational semantics is defined: $[\![P]\!] = \alpha^{\mathcal{D}}(\alpha_{\mathtt{T}}(\{\!|P|\!\}))$ (see Table 4.1), where $\sigma_{\vdash}$ and $\sigma_{\dashv}$ denote respectively the initial and the final states of the trace σ. Given two closures $\phi \in \mathit{uco}(\mathbb{V}^{\mathtt{H}})$ and $\eta \in \mathit{uco}(\mathbb{V}^{\mathtt{L}})$, we define the abstraction $\alpha^{\eta}_{\phi} : (\Sigma \to \Sigma) \longrightarrow \wp(\wp(\Sigma) \times \wp(\Sigma))$ such that for any $f : \Sigma \longrightarrow \Sigma$:

$$\alpha^{\eta}_{\phi}(f) = \left\{ \langle S_{\vdash}, S_{\dashv} \rangle \,\middle|\, S_{\vdash} = \langle \phi(h), \eta(l) \rangle, h \in \mathbb{V}^{\mathtt{H}}, l \in \mathbb{V}^{\mathtt{L}}\, S_{\dashv} = f(\phi(h), \eta(l)) \right\}$$

The idea is to abstract the denotational input/output semantics to the set of all the possible associations between the corresponding input/output abstract states. In this way, we model the observation made by the attacker, which consists precisely in the ability of observing input/output abstract values. Consider a function $\mathcal{C}_{\mathtt{H}} : \wp(\mathbb{V}^{\mathtt{H}}) \longrightarrow \mathbb{V}^{\mathtt{H}}$ that uniquely chooses an element in the domain of values $\mathbb{V}^{\mathtt{H}}$. Note that the equation $\forall h_1, h_2.\ \rho([\![P]\!](\langle \phi(h_1), \eta(l) \rangle)^{\mathtt{L}}) = \rho([\![P]\!](\langle \phi(h_2), \eta(l) \rangle)^{\mathtt{L}})$ is equivalent to the equation: $\forall h.\ \rho([\![P]\!](\langle \phi(h), \eta(l) \rangle)^{\mathtt{L}}) = \rho([\![P]\!](\langle \phi(\mathcal{C}_{\mathtt{H}}(\mathbb{V}^{\mathtt{H}})), \eta(l) \rangle)^{\mathtt{L}})$, by the transitive property of equality. Therefore, abstract non-interference can be formulated as follows:

$$\forall h \in \mathbb{V}^{\mathtt{H}}\ .\ \rho([\![P]\!](\phi(h), \eta(l))^{\mathtt{L}}) = \rho([\![P]\!](\phi(\mathcal{C}_{\mathtt{H}}(\mathbb{V}^{\mathtt{H}})), \eta(l))^{\mathtt{L}})$$

We can define the *interference* abstraction $\alpha_{ANI} : \wp(\wp(\Sigma) \times \wp(\Sigma)) \longrightarrow \wp(\wp(\Sigma) \times \wp(\Sigma))$, which embodies the meaning of abstract non-interference and which is such that, for any $\mathcal{F} \in \wp(\wp(\Sigma) \times \wp(\Sigma))$:

$$\alpha_{ANI}(\mathcal{F}) = \left\{ \langle S_{\vdash}, S_{\dashv} \rangle \in \mathcal{F} \,\middle|\, \exists l \in \mathbb{V}^{\mathtt{L}} \,.\, S_{\vdash} = \langle \phi(\mathcal{C}_{\mathtt{H}}(\mathbb{V}^{\mathtt{H}})), \eta(l) \rangle \right\}$$

In order to obtain abstract non-interference, we assume that the attacker may observe only the ρ abstraction of the low output. This corresponds to the attacker modeled by $\eta, \rho \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$: $\alpha^{\rho\eta}_{ATT} : \wp(\wp(\Sigma) \times \wp(\Sigma)) \longrightarrow \wp(\wp(\mathbb{V}^{\mathtt{L}}) \times \wp(\mathbb{V}^{\mathtt{L}}))$ where

$$\alpha^{\rho\eta}_{ATT}(\mathcal{F}) = \left\{ \langle \eta(X_{\mathtt{L}}), \rho(Y_{\mathtt{L}}) \rangle \,\middle|\, \langle \langle X_{\mathtt{H}}, X_{\mathtt{L}} \rangle, \langle Y_{\mathtt{H}}, Y_{\mathtt{L}} \rangle \rangle \in \mathcal{F} \right\}$$

Finally, we can specify abstract non-interference as follows:

$$\boxed{\alpha^{\rho\eta}_{ATT} \circ \alpha^{\eta}_{\phi}(\llbracket P \rrbracket) = \alpha^{\rho\eta}_{ATT} \circ \alpha_{ANI} \circ \alpha^{\eta}_{\phi}(\llbracket P \rrbracket)}$$

Theorem 10.4. $\alpha^{\rho\eta}_{ATT} \circ \alpha^{\eta}_{\phi}(\llbracket P \rrbracket) = \alpha^{\rho\eta}_{ATT} \circ \alpha_{ANI} \circ \alpha^{\eta}_{\phi}(\llbracket P \rrbracket)$ *iff* $(\eta)P \ (\phi \rightsquigarrow\!\!\!\!| \rho)$.

Proof. Let us prove the implication ($\Rightarrow$). Suppose $0 = \mathcal{C}_{\mathtt{H}}(\mathbb{V}^{\mathtt{H}}))$. Suppose that $\alpha^{\rho\eta}_{ATT} \circ \alpha^{\eta}_{\phi}(\llbracket P \rrbracket) = \alpha^{\rho\eta}_{ATT} \circ \alpha_{ANI} \circ \alpha^{\eta}_{\phi}(\llbracket P \rrbracket)$ and that $\not\models (\eta)P \ (\phi \rightsquigarrow\!\!\!\!| \rho)$, towards a contradiction. The latter hypothesis implies that, for what we underlined above, there exists $h \in \mathbb{V}^{\mathtt{H}}$ such that $\rho(\llbracket P \rrbracket(\phi(h), \eta(l))^{\mathtt{L}}) \neq \rho(\llbracket P \rrbracket(\phi(0), \eta(l))^{\mathtt{L}})$. Suppose that we have

$$\langle \langle \phi(h), \eta(l) \rangle, \llbracket P \rrbracket(\phi(h), \eta(l)) \rangle \in \alpha^{\eta}_{\phi}(\llbracket P \rrbracket)$$

so $\langle \eta(l), \rho(\llbracket P \rrbracket(\phi(h), \eta(l))^{\mathtt{L}}) \rangle \in \alpha^{\rho\eta}_{ATT} \circ \alpha^{\eta}_{\phi}(\llbracket P \rrbracket)$. On the other hand, we have also that $\langle \langle \phi(0), \eta(l) \rangle, \llbracket P \rrbracket(\phi(0), \eta(l)) \rangle \in \alpha^{\eta}_{\phi}(\llbracket P \rrbracket)$ and therefore it is in $\alpha_{ANI} \circ \alpha^{\eta}_{\phi}(\llbracket P \rrbracket)$. Namely, $\langle \eta(l), \rho(\llbracket P \rrbracket(\phi(0), \eta(l))^{\mathtt{L}}) \rangle \in \alpha^{\rho\eta}_{ATT} \circ \alpha_{ANI} \circ \alpha^{\eta}_{\phi}(\llbracket P \rrbracket)$. Since in this last set we have, by construction, only one association with $\eta(l)$, in order to have in the abstraction the pair $\langle \eta(l), \rho(\llbracket P \rrbracket(\phi(h), \eta(l))^{\mathtt{L}}) \rangle$, we need that $\rho(\llbracket P \rrbracket(\phi(h), \eta(l))^{\mathtt{L}}) = \rho(\llbracket P \rrbracket(\phi(0), \eta(l))^{\mathtt{L}})$ which is a contradiction. Therefore $\models (\eta)P \ (\phi \rightsquigarrow\!\!\!\!| \rho)$.
Consider the implication ($\Leftarrow$), we note that in general it holds $\alpha^{\rho\eta}_{ATT} \circ \alpha^{\eta}_{\phi}(\llbracket P \rrbracket) \supseteq \alpha^{\rho\eta}_{ATT} \circ \alpha_{ANI} \circ \alpha^{\eta}_{\phi}(\llbracket P \rrbracket)$, we have to prove the other inclusion. Suppose $\models (\eta)P \ (\phi \rightsquigarrow\!\!\!\!| \rho)$, then for what we noted above we have $\forall h \in \mathbb{V}^{\mathtt{H}}$ that $\rho(\llbracket P \rrbracket(\phi(h), \eta(l))^{\mathtt{L}}) = \rho(\llbracket P \rrbracket(\phi(0), \eta(l))^{\mathtt{L}})$. Hence, consider $\langle \eta(l), \rho(\llbracket P \rrbracket(\phi(h), \eta(l))^{\mathtt{L}}) \rangle \in \alpha^{\rho\eta}_{ATT} \circ \alpha^{\eta}_{\phi}(\llbracket P \rrbracket)$, then $\langle \langle \phi(h), \eta(l) \rangle, \llbracket P \rrbracket(\phi(h), \eta(l)) \rangle \in \alpha^{\eta}_{\phi}(\llbracket P \rrbracket)$. On the other hand, we have that $\langle \langle \phi(0), \eta(l) \rangle, \llbracket P \rrbracket(\phi(0), \eta(l)) \rangle \in \alpha^{\eta}_{\phi}(\llbracket P \rrbracket)$ and therefore $\langle \eta(l), \rho(\llbracket P \rrbracket(\phi(0), \eta(l))^{\mathtt{L}}) \rangle \in \alpha^{\rho\eta}_{ATT} \circ \alpha_{ANI} \circ \alpha^{\eta}_{\phi}(\llbracket P \rrbracket)$, hence, by hypothesis, we conclude $\langle \eta(l), \rho(\llbracket P \rrbracket(\phi(h), \eta(l))^{\mathtt{L}}) \rangle \in \alpha^{\rho\eta}_{ATT} \circ \alpha_{ANI} \circ \alpha^{\eta}_{\phi}(\llbracket P \rrbracket)$.

Note that the observation abstraction is precisely the composition $\alpha^{\eta}_{\phi} \circ \alpha^{\mathcal{D}} \circ \alpha_{\mathtt{T}}$.

As far as the narrow case is concerned, we have to check if the possible executions with the high variable ranging on the whole concrete domain $\mathbb{V}^{\mathtt{H}}$ and the low variables ranging on the set of values with the same property η are equal to the interference abstraction obtained by setting the high variable to $\mathcal{C}_{\mathtt{H}}(\mathbb{V}^{\mathtt{H}})$ and the low one to any fixed value in the given property of low variables. This means that we have to change the interference abstraction given above as follows, where $\mathcal{C}_{\mathtt{L}} : \wp(\mathbb{V}^{\mathtt{L}}) \longrightarrow \mathbb{V}^{\mathtt{L}}$ is a function that uniquely selects an element from sets of values: $\alpha_{NANI} : \wp(\wp(\Sigma) \times \wp(\Sigma)) \longrightarrow \wp(\wp(\Sigma) \times \wp(\Sigma))$

$$\alpha_{NANI}(\mathcal{F}) = \left\{ f \,\middle|\, \begin{array}{l} \exists l \in \mathbb{V}^{\mathtt{L}} \,.\, f = \langle\langle \mathcal{C}_{\mathtt{H}}(\mathbb{V}^{\mathtt{H}}), \eta(l)\rangle, S_{\dashv}\rangle,\ \langle\langle \mathcal{C}_{\mathtt{H}}(\mathbb{V}^{\mathtt{H}}), l'\rangle, S_{\dashv}\rangle \in \mathcal{F}, \\ l' = \mathcal{C}_{\mathtt{L}}(\{\ y \in \mathbb{V}^{\mathtt{L}} \,|\, \eta(y) = \eta(l)\ \}) \end{array} \right\}$$

Therefore, we can rewrite also narrow abstract non-interference (NANI).

$$\boxed{\alpha^{\rho\eta}_{ATT} \circ \alpha^{id}_{id}([\![P]\!]) = \alpha^{\rho\eta}_{ATT} \circ \alpha_{NANI} \circ \alpha^{id}_{id}([\![P]\!])}$$

Theorem 10.5. $\alpha^{\rho\eta}_{ATT} \circ \alpha^{id}_{id}([\![P]\!]) = \alpha^{\rho\eta}_{ATT} \circ \alpha_{NANI} \circ \alpha^{id}_{id}([\![P]\!])$ *iff* $[\eta]P\ (\rho)$.

Proof. Let us prove the implication ($\Rightarrow$). Consider the choice function $\mathcal{C}_{\mathtt{H}}(\mathbb{V}^{\mathtt{H}}) = 0$, and suppose that $\alpha^{\rho\eta}_{ATT} \circ \alpha^{id}_{id}([\![P]\!]) = \alpha^{\rho\eta}_{ATT} \circ \alpha_{NANI} \circ \alpha^{id}_{id}([\![P]\!])$ and, towards a contradiction, that $\not\models [\eta]P\ (\rho)$. Suppose that $l' = \mathcal{C}_{\mathtt{L}}(\{\ y \in \mathbb{V}^{\mathtt{L}} \,|\, \eta(y) = \eta(l)\ \})$. The hypothesis above implies that there exists $h \in \mathbb{V}^{\mathtt{H}}$ such that $\rho([\![P]\!](h,l)^{\mathtt{L}}) \neq \rho([\![P]\!](0,l')^{\mathtt{L}})$, since if all the elements are equal to $\rho([\![P]\!](0,l')^{\mathtt{L}})$ then by transitivity we would have secrecy. Consider now that by definition we have $\langle\langle h,l\rangle, [\![P]\!](h,l)\rangle \in \alpha^{id}_{id}([\![P]\!])$, therefore $\langle \eta(l), \rho([\![P]\!](h,l)^{\mathtt{L}})\rangle \in \alpha^{\rho\eta}_{ATT} \circ \alpha^{id}_{id}([\![P]\!])$. On the other hand we have also that $\langle\langle 0,l'\rangle, [\![P]\!](0,l')\rangle \in \alpha^{id}_{id}([\![P]\!])$ and therefore in $\alpha_{NANI} \circ \alpha^{id}_{id}([\![P]\!])$. Namely $\langle \eta(l'), \rho([\![P]\!](0,l')^{\mathtt{L}})\rangle \in \alpha^{\rho\eta}_{ATT} \circ \alpha_{NANI} \circ \alpha^{id}_{id}([\![P]\!])$. Since in this last set we have, by construction, only one association with $\eta(l)$, in order to have $\langle \eta(l), \rho([\![P]\!](h,l)^{\mathtt{L}})\rangle$ in it we need that $\rho([\![P]\!](h,l)^{\mathtt{L}}) = \rho([\![P]\!](0,l')^{\mathtt{L}})$ which is a contradiction. Therefore $\models [\eta]P\ (\rho)$.
Consider the implication ($\Leftarrow$), we note that in general it holds $\alpha^{\rho\eta}_{ATT} \circ \alpha^{id}_{id}([\![P]\!]) \supseteq \alpha^{\rho\eta}_{ATT} \circ \alpha_{NANI} \circ \alpha^{id}_{id}([\![P]\!])$, we have to prove the other inclusion. Suppose $\models [\eta]P\ (\rho)$, then for what we noted above we have $\forall h \in \mathbb{V}^{\mathtt{H}}$ that $\rho([\![P]\!](h,l)^{\mathtt{L}}) = \rho([\![P]\!](0,l')^{\mathtt{L}})$, where $l' = \mathcal{C}_{\mathtt{L}}(\{\ y \in \mathbb{V}^{\mathtt{L}} \,|\, \eta(y) = \eta(l)\ \})$. Hence, consider $\langle \eta(l), \rho([\![P]\!](h,l)^{\mathtt{L}})\rangle \in \alpha^{\rho\eta}_{ATT} \circ \alpha^{id}_{id}([\![P]\!])$ then $\langle\langle h,l\rangle, [\![P]\!](h,l)\rangle \in \alpha^{id}_{id}([\![P]\!])$. On the other hand, we have $\langle\langle 0,l'\rangle, [\![P]\!](0,l')\rangle \in \alpha^{id}_{id}([\![P]\!])$ and so, $\langle \eta(l'), \rho([\![P]\!](0,l')^{\mathtt{L}})\rangle \in \alpha^{\rho\eta}_{ATT} \circ \alpha_{NANI} \circ \alpha^{id}_{id}([\![P]\!])$, hence we obtain $\langle \eta(l), \rho([\![P]\!](h,l)^{\mathtt{L}})\rangle \in \alpha^{\rho\eta}_{ATT} \circ \alpha_{NANI} \circ \alpha^{id}_{id}([\![P]\!])$.

Let us show now that the method given for deriving attackers in the generalized abstract non-interference is, indeed, a generalization of the method given in [52]. First of all, since the relation $\preccurlyeq_{\max}$ is applied to maps, namely traces with length 2, then to have a common prefix means to have the same input, therefore the relation $\preccurlyeq_{\max}$ collects together all the maps starting from the same input. Consider the abstract case, then, by definition, we have that the elements in $\alpha_{ATT} \circ \alpha^{\eta}_{\phi}$ are of the kind $\eta(l) \mapsto [\![P]\!](\phi(h), \eta(l))^{\mathtt{L}}$, therefore $[\sigma]$ collects together all this kind of maps starting from the same $\eta(l)$. Namely

$$[\eta(l) \mapsto [\![P]\!](\phi(h), \eta(l))^{\mathtt{L}}] = \left\{\ \eta(l) \mapsto [\![P]\!](\phi(h'), \eta(l))^{\mathtt{L}} \,\middle|\, h' \in \mathbb{V}^{\mathtt{H}}\ \right\},$$

which is isomorphic to $\Upsilon^{\eta,\,\phi}_{[\![P]\!]}$ in [52]. Therefore, we obtain that

$$\mathbb{D}_{[\![P]\!]} = \left\{\ \eta(l) \mapsto [\![P]\!](\mathbb{V}^{\mathtt{H}}, \eta(l))^{\mathtt{L}} \,\middle|\, l \in \mathbb{V}^{\mathtt{L}}\ \right\},$$

while

$$Irr_{[\![P]\!]} = \left\{\ X \,\middle|\, \forall h \in \mathbb{V}^{\mathtt{H}}, l \in \mathbb{V}^{\mathtt{L}}\ .X \notin \uparrow(\eta(l) \mapsto [\![P]\!](\phi(h), \eta(l))^{\mathtt{L}})\ \right\}$$

which is $Irr^{\phi,\,\eta}_{[\![P]\!]}$. Finally consider the predicate $Secr_{[\![P]\!]}$, $Secr_{[\![P]\!]}(X)$ iff

$$\forall [\![P]\!](\phi(h), \eta(l))^{\mathtt{L}} \ .(\exists Z \in [\eta(l) \mapsto [\![P]\!](\phi(h), \eta(l))^{\mathtt{L}}] \,.\, Z \subseteq X \ \Rightarrow \\ \forall W \in [\eta(l) \mapsto [\![P]\!](\phi(h), \eta(l))^{\mathtt{L}}] \,.\, W \subseteq X),$$

which is precisely isomorphic to the notion given in [52].

10.1.3 Timed abstract non-interference as GANI

In the previous chapter, we introduced a notion of timed abstract non-interference, which consists in enriching the notion of abstract non-interference in order to capture also timing channels. We can formulate also the timed abstract non-interference as generalized abstract non-interference. Let us define the functions:

$$safe[\alpha^{\eta}_{\phi}]([\![P]\!]) = \left\{ S_{\vdash} \mapsto^{n} S_{\dashv} \;\middle|\; \begin{array}{l} S_{\vdash} = \langle \phi(h), \eta(l) \rangle, h \in \mathbb{V}^{\mathtt{H}}, l \in \mathbb{V}^{\mathtt{L}} \\ S_{\dashv} = [\![P]\!]^{safe}_{n}(\langle \phi(h), \eta(l) \rangle) \end{array} \right\} \text{ and}$$

$$stu[\alpha^{\eta}_{\phi}]([\![P]\!]) = \left\{ S_{\vdash} \mapsto^{n} S_{\dashv} \;\middle|\; \begin{array}{l} S_{\vdash} = \langle \phi(h), \eta(l) \rangle, h \in \mathbb{V}^{\mathtt{H}}, l \in \mathbb{V}^{\mathtt{L}} \\ S_{\dashv} = [\![P]\!]^{stu}_{n}(\langle \phi(h), \eta(l) \rangle) \end{array} \right\}$$

Then we have that the two notions above are respectively

$$\boxed{\begin{array}{c} \alpha^{\rho\eta}_{ATT} \circ safe[\alpha^{\eta}_{\phi}]([\![P]\!]) = \alpha^{\rho\eta}_{ATT} \circ \alpha_{ANI} \circ safe[\alpha^{\eta}_{\phi}]([\![P]\!]) \text{ and} \\ \alpha^{\rho\eta}_{ATT} \circ stu[\alpha^{\eta}_{\phi}]([\![P]\!]) = \alpha^{\rho\eta}_{ATT} \circ \alpha_{ANI} \circ stu[\alpha^{\eta}_{\phi}]([\![P]\!]) \end{array}}$$

Theorem 10.6.

- $\alpha^{\rho\eta}_{ATT} \circ safe[\alpha^{\eta}_{\phi}]([\![P]\!]) = \alpha^{\rho\eta}_{ATT} \circ \alpha_{ANI} \circ safe[\alpha^{\eta}_{\phi}]([\![P]\!])$ *if and only if* P *is secure for timed abstract non-interference.*
- $\alpha^{\rho\eta}_{ATT} \circ stu[\alpha^{\eta}_{\phi}]([\![P]\!]) = \alpha^{\rho\eta}_{ATT} \circ \alpha_{ANI} \circ stu[\alpha^{\eta}_{\phi}]([\![P]\!])$ *if and only if* P *is secure for abstract non-interference.*

Proof. We prove only one point since the two proofs are similar. Let us prove the implication ($\Rightarrow$). Consider the choice function $\mathcal{C}_{\mathtt{H}}(\mathbb{V}^{\mathtt{H}}) = 0$ and suppose that $\alpha^{\rho\eta}_{ATT} \circ safe[\alpha^{\eta}_{\phi}]([\![P]\!]) = \alpha^{\rho\eta}_{ATT} \circ \alpha_{ANI} \circ safe[\alpha^{\eta}_{\phi}]([\![P]\!])$ and, towards a contradiction, that P is not secure as regards timed abstract non-interference. This implies that there exists $h \in \mathbb{V}^{\mathtt{H}}$ and $i \in \mathbb{N}$ such that $\rho([\![P]\!]^{safe}_{i}(\phi(h), \eta(l))^{\mathtt{L}}) \neq \rho([\![P]\!]^{safe}_{i}(\phi(0), \eta(l))^{\mathtt{L}})$. By definition we have $\langle \langle \phi(h), \eta(l) \rangle, [\![P]\!]^{safe}_{i}(\phi(h), \eta(l)) \rangle \in safe[\alpha^{\eta}_{\phi}]([\![P]\!])$, therefore $\langle \eta(l), \rho([\![P]\!]^{safe}_{i}(\phi(h), \eta(l))^{\mathtt{L}}) \rangle \in \alpha^{\rho\eta}_{ATT} \circ safe[\alpha^{\eta}_{\phi}]([\![P]\!])$. On the other hand we have, by definition, that $\langle \langle \phi(0), \eta(l) \rangle, [\![P]\!]^{safe}_{i}(\phi(0), \eta(l)) \rangle \in safe[\alpha^{\eta}_{\phi}]([\![P]\!])$ and therefore in $\alpha_{ANI} \circ safe[\alpha^{\eta}_{\phi}]([\![P]\!])$. Namely we can prove that $\langle \eta(l), \rho([\![P]\!]^{safe}_{i}(\phi(0), \eta(l))^{\mathtt{L}}) \rangle \in \alpha^{\rho\eta}_{ATT} \circ \alpha_{ANI} \circ safe[\alpha^{\eta}_{\phi}]([\![P]\!])$. Since in this last set we have only one association with $\eta(l)$, in order to have $\langle \eta(l), \rho([\![P]\!]^{safe}_{i}(\phi(h), \eta(l))^{\mathtt{L}}) \rangle$ in it we need that the equality $\rho([\![P]\!]^{safe}_{i}(\phi(h), \eta(l))^{\mathtt{L}}) = \rho([\![P]\!](\phi(0), \eta(l))^{\mathtt{L}})$ holds. But this is a contradiction. Therefore P is secure for timed abstract non-interference.

As far as the implication ($\Leftarrow$) is concerned, we note that in general it holds $\alpha^{\rho\eta}_{ATT} \circ safe[\alpha^{\eta}_{\phi}](\llbracket P \rrbracket) \supseteq \alpha^{\rho\eta}_{ATT} \circ \alpha_{ANI} \circ safe[\alpha^{\eta}_{\phi}](\llbracket P \rrbracket)$, we have to prove the other inclusion. Suppose the program P secure, then $\forall h \in \mathbb{V}^{\mathtt{H}}$ and $\forall i \in \mathbb{N}$ we have that $\rho(\llbracket P \rrbracket^{safe}_i(\phi(h), \eta(l))^{\mathtt{L}}) = \rho(\llbracket P \rrbracket^{safe}_i(\phi(0), \eta(l))^{\mathtt{L}})$. Hence, consider the element $\langle \eta(l), \rho(\llbracket P \rrbracket^{safe}_i(\phi(h), \eta(l))^{\mathtt{L}}) \rangle \in \alpha^{\rho\eta}_{ATT} \circ safe[\alpha^{\eta}_{\phi}](\llbracket P \rrbracket)$. In this way we obtain $\langle \langle \phi(h), \eta(l) \rangle, \llbracket P \rrbracket^{safe}_i(\phi(h), \eta(l)) \rangle \in safe[\alpha^{\eta}_{\phi}](\llbracket P \rrbracket)$. On the other hand, for the same reasoning, we have $\langle \langle \phi(0), \eta(l) \rangle, \llbracket P \rrbracket^{safe}_i(\phi(0), \eta(l)) \rangle \in safe[\alpha^{\eta}_{\phi}](\llbracket P \rrbracket)$ and therefore, we obtain the element $\langle \eta(l), \rho(\llbracket P \rrbracket^{safe}_i(\phi(0), \eta(l))^{\mathtt{L}}) \rangle \in \alpha^{\rho\eta}_{ATT} \circ \alpha_{ANI} \circ safe[\alpha^{\eta}_{\phi}](\llbracket P \rrbracket)$. So, by hypothesis, $\langle \eta(l), \rho(\llbracket P \rrbracket^{safe}_i(\phi(h), \eta(l))^{\mathtt{L}}) \rangle \in \alpha^{\rho\eta}_{ATT} \circ \alpha_{ANI} \circ safe[\alpha^{\eta}_{\phi}](\llbracket P \rrbracket)$.

The interesting aspect of translating timed abstract non-interference in the framework of generalized abstract non-interference is the possibility, in GANI, of deriving attackers. In this way, we are able to characterize which is the most concrete property concerning time, such that the attacker is not able to disclose private information through timing channels.

10.2 GANI in concurrency

In [47], the authors introduced a classification of security properties for *security process algebras*, an extension of CCS. Since, process algebras can be modeled by computational trees, we show how, different security properties defined in [47] can be re-interpreted as instances of the generalized abstract non-interference. See Sect. 4.2.2 for the syntax and the semantics of SPA.

Consider a process $P \in$ SPA, whose computational tree is $\{\!|P|\!\}$. We start by considering NNI (non-deterministic non-interference) which is defined by using the trace equivalence $\approx_{\mathtt{T}}$ in the following way: $(P \backslash_I \mathtt{H})/\mathtt{H} \approx_{\mathtt{T}} P/\mathtt{H}$, where $P/\mathtt{H}$ means that all the actions in $\mathtt{H}$ (high) are hidden, i.e., they are substituted by the internal action ε, while $P \backslash_I \mathtt{H}$ means that all the actions in $\mathtt{H}$ which are input actions cannot be executed by P. We can translate this definition obtaining an instance of generalized abstract non-interference. It is clear that the definition of NNI considers the concrete system P, this means that $\alpha_{OBS} \stackrel{\text{def}}{=} id$. On the other hand, we have that what an external user can observe is the system having the high actions hidden. Therefore, we have to define the attacker abstraction that hides high-level actions. Let $t \in \mathbb{T}_{Act}$, where Act is the set of all the possible actions, and consider the following function: $low : \mathbb{T}_{Act} \longrightarrow \mathbb{T}_{\mathtt{L}}$ such that $low(t)$ is the tree where any label $\sigma \in \mathtt{H}$ in t is substituted by τ. Then we define the abstraction $\alpha_{low} : \wp(\mathbb{T}_{Act}) \to \wp(\mathbb{T}_{\mathtt{L}})$ such that $\alpha_{low}(T) = \{\, low(t) \,|\, t \in T \,\}$. This abstraction specifies the attacker abstraction in GANI and it is such that $\{\!|P/\mathtt{H}|\!\} = \alpha_{low}(\{\!|P|\!\})$.

Proposition 10.7. *The map α_{low} is additive and $\{\!|P/\mathtt{H}|\!\} = \alpha_{low}(\{\!|P|\!\})$.*

Proof. Immediate by construction.

Moreover, we can note that NNI is defined by using trace equivalence, therefore this means that the attacker can analyze traces of computations only. By definition two systems are trace equivalent if they accept the same language, therefore we have to make equal the $\alpha_{\mathtt{T}}$ abstraction of the result, namely $\alpha_{ATT} \stackrel{\text{def}}{=} \alpha_{\mathtt{T}} \circ \alpha_{low}$. Finally, consider the operation $P\backslash_I \mathtt{H}$ which avoids high-level inputs. Let I be the set of input actions, we can define the abstraction $\alpha^I_{\mathtt{L}} : \wp(\mathbb{T}_{Act}) \longrightarrow \wp(\mathbb{T}_{Act})$ such that for any $T \subseteq \mathbb{T}_{Act}$: $\alpha^I_{\mathtt{L}}(T) = \left\{ t \in T \,\middle|\, \forall \sigma \in t \,.\, \sigma \notin \mathtt{H} \cap I \right\}$, where $\sigma \in t$ is a shorthand notation for σ being an action (node) in t. Then we have that $\{\!|P\backslash_I \mathtt{H}|\!\} = \alpha^I_{\mathtt{L}}(\{\!|P|\!\})$.

Proposition 10.8. *The map $\alpha_{\mathtt{L}}$ is additive and $\{\!|P\backslash \mathtt{H}|\!\} = \alpha_{\mathtt{L}}(\{\!|P|\!\})$.*

Proof. Immediate by construction.

At this point, we can derive the NNI as:

$$\boxed{\alpha_{\mathtt{T}} \circ \alpha_{low}(\{\!|P|\!\}) = (\alpha_{\mathtt{T}} \circ \alpha_{low}) \circ \alpha^I_{\mathtt{L}}(\{\!|P|\!\})}$$

Consider now the notion of *Strong Non-deterministic Non-Interference* (SNNI) defined in [47]: P satisfies SNNI iff $P/\mathtt{H} \approx_{\mathtt{T}} P\backslash \mathtt{H}$. In order to define SNNI as an instance of the generalized abstract non-interference, we have to define the operator $P/\mathtt{H}$, that hides all the high-level actions. Let us define $\alpha_{\mathtt{L}} : \wp(\mathbb{T}_{Act}) \longrightarrow \wp(\mathbb{T}_{Act})$ such that $\forall T \subseteq \mathbb{T}_{Act}$ we have $\alpha_{\mathtt{L}}(T) = \left\{ t \in T \,\middle|\, \forall \sigma \in t \,.\, \sigma \in \mathtt{L} \right\}$. This defines the interference abstraction in GANI and it is such that $\{\!|P\backslash \mathtt{H}|\!\} = \alpha_{\mathtt{L}}(\{\!|P|\!\})$. The standard notion of SNNI introduced in [47] can be defined as

$$\boxed{\alpha_{\mathtt{T}} \circ \alpha_{low}(\{\!|P|\!\}) = \alpha_{\mathtt{T}} \circ \alpha_{low} \circ \alpha_{\mathtt{L}}(\{\!|P|\!\}).}$$

At this point, since bisimulations are equivalence relations [92], they can be viewed as abstractions of computational trees, i.e., a tree is abstracted into the equivalence class of all the trees bisimilar to it, then we can model the BNNI and BSNNI. We obtain this by substituting to $\alpha_{\mathtt{T}}$, the abstraction $\alpha_{\mathtt{B}}$, corresponding to the chosen bisimulation, which associates with a computation the set of all the computations bisimilar to the given one.

Consider now non-deducibility on compositions (NDC) and the bisimulation-based NDC (BNDC) notions of non-interference. NDC is defined as: $\forall \Pi \,.\, P/\mathtt{H} \approx_{\mathtt{T}} (P||\Pi)\backslash \mathtt{H}$, where Π is a process that can execute only high-level actions. In [47] it is proved that NDC=SNNI, therefore also NDC can be modeled as a generalized abstract non-interference. The situation is different when we consider BNDC, i.e., $\forall \Pi \,.\, P/\mathtt{H} \approx_{\mathtt{B}} (P||\Pi)\backslash \mathtt{H}$, for the bisimulation relation B. In this case, we have that BNDC$\neq$BSNNI, and therefore we have to explicitly model it as generalized abstract non-interference. In [47] the authors also prove that BNDC can be equivalently formalized as: $\forall \Pi \,.\, P\backslash \mathtt{H} \approx_{\mathtt{B}} (P||\Pi)\backslash \mathtt{H}$. At this point, we note that we have to consider $\alpha_{\mathtt{B}} \circ \alpha_{\mathtt{L}}$ as α_{ATT}, since in this definition it is only observable what a low-level user (i.e., a user that can execute only low level actions) can see, namely only the computations without high-level actions. Moreover, we have that α_{OBS} is

the identity, since, in this case, non-interference is defined on computational trees. Finally, we define α_{INT} noting that the semantics (computational tree) of $P||\Pi$ contains the semantics of $P.\Pi$ (which doesn't execute synchronizations), therefore we can define $\alpha_{INT}(\{\!|P||\Pi|\!\}) = \{\!|P.\Pi|\!\}$. Hence we can model BNDC as follows:

$$\boxed{\forall \Pi.(\alpha_{\mathtt{B}} \circ \alpha_{\mathtt{L}}) \circ \alpha_{INT}(\{\!|P||\Pi|\!\}) = \alpha_{\mathtt{B}} \circ \alpha_{\mathtt{L}}(\{\!|P||\Pi|\!\}).}$$

This is BNDC since in the right side of the equality $\alpha_{\mathtt{L}}$ is applied to the semantics of $P.\Pi$, and therefore executes only the high-level actions of P.

10.3 GANI in real-time systems

Let A be a timed automaton, $\{\!|A|\!\}$ be the corresponding computational tree and $\langle\!|A|\!\rangle$ be the corresponding timed language accepted by A (see Sect. 4.2.3). In [12] a notion of non-interference for timed automata is introduced (see Sect. 5.3.4). Given a natural number n, the authors say that high-level actions do not interfere with the system, by considering minimum delay n, if the system behaviour in absence of high-level actions is equivalent to the system behaviour, observed on low-level actions only, when high-level actions can occur with a delay between them greater than n. We recall that in [12] a system A is said to be *n-non-interfering* iff $\langle\!|A|\!\rangle^n_{\mathtt{H}}/\mathtt{H} = \langle\!|A|\!\rangle|_{\mathtt{L}}$.
Let us consider the example in Fig. 10.2, from [12]. This timed automaton have $\mathtt{L}$ = {begin-c,end-c} and $\mathtt{H}$ = {cloche,reset}. There is only one possible trace of only low-level actions, namely:

$$\langle \text{begin-c}, 2\rangle\langle \text{end-c}, 4\rangle \ldots \langle \text{begin-c}, 2+4i\rangle\langle \text{end-c}, 4+4i\rangle \ldots$$

If more than one cloche action is executed and the time elapsed between them is less than 1, then it is possible to execute the action reset, which can change the moment of the execution of begin-c, and therefore in this case we have an interference.

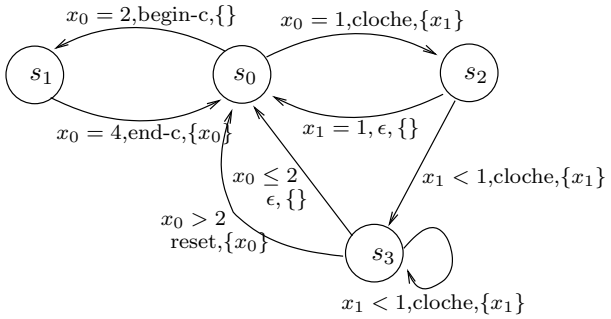


Fig. 10.2. A timed automaton: A simplified airplane control.

In particular, for example, we can have the trace

$$\langle \text{begin-c}, 2\rangle\langle \text{end-c}, 4\rangle\langle \text{cloche}, 5\rangle\langle \text{cloche}, 5.6\rangle\langle \text{cloche}, 6.3\rangle\langle \text{begin-c}, 8.3\rangle \ldots$$

whose projection $\langle \text{begin-c}, 2\rangle\langle \text{end-c}, 4\rangle\langle \text{begin-c}, 6.3\rangle\langle \text{end-c}, 8.3\rangle$, on the low-level actions, is not the one described above. This means that in this system there is interference.

Consider the attacker abstraction α_{low}, defined in Sec. 10.2, the interference abstraction $\alpha_{\mathtt{L}}$ and the language $\langle\!|A|\!\rangle^n_{\mathtt{H}}$. Recall that the timed language accepted is a set of traces on $\Sigma \times \mathbb{R}$, i.e., $(\Sigma \times \mathbb{R})^*$. Therefore, we define the family of abstractions $\alpha_n : \wp((\Sigma \times \mathbb{R})^*) \to \wp((\Sigma \times \mathbb{R})^*)$, with $n \in \mathbb{N}$, as follows:

$$\alpha_n(\langle\!|A|\!\rangle) = \left\{ \tau \in \langle\!|A|\!\rangle \,\middle|\, \forall \langle \sigma_i, t_i\rangle, \langle \sigma_j, t_j\rangle \in \tau \,.\, i \neq j,\ \sigma_i, \sigma_j \in \mathtt{H} \ \Rightarrow\ |t_i - t_j| \geq n \right\}$$

Proposition 10.9. *The map α_n, $\forall n \in \mathbb{N}$, is additive and $\langle\!|A|\!\rangle^n_{\mathtt{H}} = \alpha_n(\langle\!|A|\!\rangle)$.*

Proof. Immediate by construction.

Then the notion of non-interference introduced in [12] for timed automata can be specified as follows:

$$\boxed{\alpha_{low} \circ \alpha_n(\langle\!|A|\!\rangle) = \alpha_{low} \circ \alpha_{\mathtt{L}} \circ \alpha_n(\langle\!|A|\!\rangle),}$$

where $\alpha_{\mathtt{L}} \circ \alpha_n = \alpha_{\mathtt{L}}$. Note that, in this case, $\alpha_{OBS} = \alpha_n \circ \alpha_{tT}$.

10.4 Discussion

In this chapter, we introduce a generalization of abstract non-interference for automata and concurrent systems. We believe that the combination of abstract interpretation and non-interference may provide advanced techniques for analyzing how sub-components of complex systems interact during the computation. On one side, abstract interpretation has been proved to be the most appropriate framework for reasoning about properties of computations at different levels of abstraction. On the other side, strong-dependency, and in particular non-interference, is the most appropriate notion to disclose information-flows among sub-components of a system, when a variation of some of them can be conveyed to the others. Generalized abstract non-interference is intended to bridge these two notions in order to provide adequate methods for studying properties of complex systems by analyzing the properties of computations that are conveyed among system sub-components. In this sense, generalized abstract non-interference may provide a framework for studying the relation between different and interacting entities which may be reciprocally influenced by the action of computing, giving advanced techniques for systematically classifying the information leakage in the lattice of abstractions.

11

Conclusions

The wise seeks the truth,
the foolish thinks to have found it.
Blaise Pascal

In this thesis, we present a weakening of the notion of non-interference in language based security. This weakening is based on the idea of modeling attackers as static program analyzers whose task is to disclose confidential information by observing the input/output public behaviour. The model is, therefore, obtained by using abstract interpretation, and in particular by modeling attackers through two abstract domains, one representing the input observation and one representing the output one. Being, abstract non-interference, based on the denotational semantics, and therefore not very practical, we also introduce, for imperative languages, a compositional proof system, that allows us to derive abstract non-interference certifications, inductively on the syntax of programs. Moreover, the abstract characterization of information flows allows us to formally derive the most abstract property of private information revealed by the semantics of a program. In other words, the same model, which is abstract non-interference, allows us to characterize the most concrete *public observer*, which cannot violate non-interference, i.e., the most powerful harmless attacker, and the most abstract *private observable*, i.e., the most concrete confidential property that the semantics reveals through the observation of public outputs. Moreover, the domain transformers that derives the public observer and the private observable, are adjoint transformers in the standard abstract interpretation framework. This relation formally characterizes their dual nature. In order to prove this duality, we follow [78] modelling non-interference as an abstract domain completeness problem [31, 65]. The interesting aspect is that, this formalization of non-interference shows new ways for approaching the problem. Indeed, given a function f and an abstraction ρ that we want to be complete for the function f, [65] studies how to minimally transform ρ in

order to make it complete for f (and this is what we did in Chapter 8). On the other hand, we are also working on the problem of transforming f in order to make the domain ρ complete for the transformed function. Since in the completeness-based formalization of abstract non-interference, the function is the semantics of the system, we would like to understand how we would have to transform the system in order to guarantee that it does not disclose secrets to a given attacker. In particular, this could be interesting for deriving secret security protocols, since we could derive the minimal transformation of a given protocol, that guarantees that non-interference cannot be violated.

In this thesis, we also show that, one of the most important features, of the semantic approach to non-interference, is that simply by changing the considered semantics we indeed change the notion of non-interference that we can enforce. Hence, we notice that we can define a stronger notion of non-interference, simply by considering the maximal trace semantics instead of the denotational one, or we can define abstract non-interference for even non-deterministic systems, simply by considering the non-deterministic denotational semantics that associates with a state the *set* of all the reachable states [27]. This general aspect is even more powerful, since by considering a semantics that can store the time elapsed during the execution of a program, then we can use the same definition of abstract non-interference for obtaining a security notion that avoids also timing channels. We would like also to understand which semantics we have to consider in order to define an abstract non-interference that avoids also *probabilistic* channels, exactly as it happens with the PER model [106]. Moreover, the strong relation between abstract non-interference and the PER model, suggests us to exploit if we can give a more precise or efficient characterization of abstract non-interference whenever we consider equivalence relations instead of generic upper closure operators.

This thesis ends with a further generalization of the notion of non-interference, again by using abstract interpretation, whose aim is to model non-interference for many well known computational systems, in an uniform setting. In particular, the idea is that of considering the most concrete model that we can associate with a computational system, which is the computational tree of executions. This allows us to move from the definition for one system to another simply by suitably abstracting this concrete model, following the hierarchy of semantics defined by Cousot [27]. In this way, we can uniformly model several notions of non-interference: abstract non-interference, many of the notions defined for the process algebra Spa [47] and the decidable notion for timed automata [12]. But the most important aspect of this generalization is that we also generalize the derivation of the canonical attacker. Moreover, generalized abstract non-interference provides a pattern for defining even new and, if necessary, ad hoc notions of non-interference.

Another aspect that should be investigated is that the whole work done considers only passive attackers, namely attackers that are simple observers of the computation, but that cannot modify it. It would be interesting, to understand how we can model, in abstract non-interference, active attackers, that can modify

the semantics. But much more interesting would be to be able to use abstract non-interference for characterizing also the secrecy level of programs in presence of active attackers. A possible way for solving this problem, can be to consider the completeness characterization of abstract non-interference. Indeed, by minimally modifying the semantics, in order to guarantee non-interference, we could be able to characterize which can be the minimal transformations of the semantics that guarantees non-interference, namely which is the most concrete *active* harmless attack for the given computational system.

From the practical point of view, it misses an implementation of our derivation of canonical attackers, and therefore of the secrecy certification of programs. In this case, the well known relation between non-interference and slicing, based on the program *dependency* graphs, [1] provides the idea for a possible implementation. Indeed, if we derive the slice of a program, considering as criterion the final values of public variables, whenever the private data are contained in the slice, then it means that the final value of public variables does *depend* on the initial private data. Namely, insecure information flows are not sure, but possible. The idea is to define an algorithm for abstract program slicing, where the slicing criterion considers also a property of public data. Therefore, if we derive such an algorithm, when we take as criterion the final property ρ for the public variables, and the slice contains private data, it means that private data could interfere in the property ρ of the final value of public variables, with a possible violation of abstract non-interference.
Anyway, slicing, or abstract slicing, provides only a method for checking, and therefore certifying non-interference. We would like to derive also methods that allow to *protect* programs from certain kind of attacks. In this direction *code obfuscation* [21, 22] could help, since we could think of obfuscate a code as regards the most concrete public observer in order to avoid any possible attacker from disclosing information about confidential data. In this way, whenever we a have a certificate which states for which kind of attacks a program is not secure, we could derive a protected transformation of the code, preserving the semantics but secure as regards the considered possible attacks. This seems to put in evidence a possible relation between obfuscation used for enforcing abstract non-interference and the transformation of the semantics for guaranteeing abstract non-interference, modeled as a completeness problem.

From a more theoretical, and in particular mathematical, point of view, abstract non-interference can be interpreted as a *non relational* analysis of the relations existing among the objects composing a system, for instance the values of variables in programming languages. It is well known, that in abstract interpretation there exists a lot of work on relational abstract domains for static program *relational* analyses. These domains are devoted to investigate the relations existing between values of variable, e.g., a variable x is even *whenever* the variable y is odd. Therefore, it could be interesting to study the relation existing between relational analyses and abstract non-interference.

The fascinating aspect of this work, is that while working on abstract non-interference, we noticed that in many fields, also in those that seem really far away from language based security, the notion of *non-interference* is relevant. For example, in a biological system an important field of study consists in understanding how the presence of certain kind of proteins *interferes* with the activation/inhibition of other kinds of proteins. This problem is important since it allows to determine, for instance, which proteins have to be contained in a medicine in order to make it useful for destroying a specific virus. This example, only to say that abstract non-interference, here studied in the specific of language based security, can be seen as a very general problem, and our approach can be used and studied for weakening non-interference in order to make it fit with many problems concerning interferences and dependencies in computational systems.

List of Figures

List of Tables

References

1. M. Abadi, A. Banerjee, N. Heintze, and J. Riecke. A core calculus of dependency. In *Proc. of the 26th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '99)*, pages 147–160. ACM-Press, NY, 1999.
2. J. Agat. Transforming out timing leaks. In *Proc. of the 27th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '00)*, pages 40–53. ACM-Press, NY, 2000.
3. B. Alpern, A. J. Demers, and F. B. Schneider. Safety without stuttering. *Information Processing Letters*, 23(4):177–180, 1986.
4. B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21:181–185, 1985.
5. R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
6. T. Amtoft and A. Banerjee. Information flow analysis in logical form. In R. Giacobazzi, editor, *Proc. of The 11th Internat. Static Analysis Symp. (SAS'04)*, volume 3184 of *Lecture Notes in Computer Science*, pages 100–115. Springer-Verlag, 2004.
7. G.R. Andrews and R. P. Reitman. An axiomatic approach to information flow in programs. *ACM Trans. Program. Lang. Syst.*, 2(1):56–76, 1980.
8. A. Appel. Foundational proof-carrying code. In *Proc. of the 16th IEEE Symp. on Logic in Computer Science* (*LICS '01*), pages 247–258. IEEE Computer Society Press, Los Alamitos, Calif., 2001.
9. K. Apt and E.-R. Olderog. *Verification of sequential and concurrent programs.* Springer-Verlag, Berlin, 1997.
10. K. R. Apt and G. D. Plotkin. Countable nondeterminism and random assignment. *J. ACM*, 33(4):724–767, 1986.
11. T. Ball, A. Podelski, and S.K. Rajamani. Relative completeness of abstraction refinement for software model checking. In J.-P. Kaoen and P. Stevens, editors, *Proc. of TACAS: Tools and Algorithms for the Construction and Analysis of Systems*, volume 2280 of *Lecture Notes in Computer Science*, pages 158–172. Springer-Verlag, Berlin, 2002.
12. R. Barbuti, N. De Francesco, A. Santone, and L. Tesei. A notion of non-interference for timed automata. *Fundamenta Informaticae*, 51:1–11, 2002.
13. D. E. Bell and L. J. LaPadula. Secure computer systems: Mathematical foundations and model. Technical Report M74-244, MITRE Corp. Badford, MA, 1973.

14. G. Birkhoff. *Lattice Theory.* AMS Colloquium Publication, 3rd edition. AMS, Providence, RI, 1967.
15. T.S. Blyth and M.F. Janowitz. *Residuation theory.* Pergamon Press, 1972.
16. D. Clark, S. Hunt, and P. Malacaria. Quantitative analysis of the leakage of confidential data. In *Workshop on Quantitative Aspects of Programming Laguages (QAPL '01)*, volume 59 of *Electronic Notes in Theoretical Computer Science*. Elsevier, Amsterdam, 2001.
17. D. Clark, S. Hunt, and P. Malacaria. Quantified interference: Information theory and information flow (extended abstract). In *Workshop on Issues in the Theory of Security (WITS'04)*, 2004.
18. E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Proc. of the 12th Internat. Conf. on Computer Aided Verification* (*CAV '00*), volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer-Verlag, Berlin, 2000.
19. E. S. Cohen. Information transmission in computational systems. *ACM SIGOPS Operating System Review*, 11(5):133–139, 1977.
20. E. S. Cohen. Information transmission in sequential programs. *Foundations of Secure Computation*, pages 297–335, 1978.
21. C. Collberg and C. Thomborson. Watermarking, tamper-proofing, and obduscation-tools for software protection. *IEEE Trans. Software Eng.*, pages 735–746, 2002.
22. C. Collberg, C. Thomborson, and D. Low. A taxonomy of obduscating transformations. Technical Report 148, Department of Computer Science, The University of Auckland, 1997.
23. A. Cortesi, G. Filé, R. Giacobazzi, C. Palamidessi, and F. Ranzato. Complementation in abstract interpretation. *ACM Trans. Program. Lang. Syst.*, 19(1):7–47, 1997.
24. A. Cortesi, B. Le Charlier, and P. Van Hentenryck. Combinations of abstract domains for logic programming. In *Proc. of Conf. Record of the 21st ACM Symp. on Principles of Programming Languages* (*POPL '94*), pages 227–239. ACM Press, New York, 1994.
25. P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation (Invited Paper). In S. Brookes and M. Mislove, editors, *Proc. of the 13th Internat. Symp. on Mathematical Foundations of Programming Semantics* (*MFPS '97*), volume 6 of *Electronic Notes in Theoretical Computer Science*. Elsevier, Amsterdam, 1997. URL: http://www.elsevier.nl/locate/entcs/volume6.html.
26. P. Cousot. The calculational design of a generic abstract interpreter. In M. Broy and R. Steinbrüggen, editors, *Calculational System Design*. NATO ASI Series F. IOS Press, Amsterdam, 1999.
27. P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theor. Comput. Sci.*, 277(1-2):47,103, 2002.
28. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. of Conf. Record of the 4th ACM Symp. on Principles of Programming Languages* (*POPL '77*), pages 238–252. ACM Press, New York, 1977.
29. P. Cousot and R. Cousot. A constructive characterization of the lattices of all retractions, preclosure, quasi-closure and closure operators on a complete lattice. *Portug. Math.*, 38(2):185–198, 1979.

30. P. Cousot and R. Cousot. Constructive versions of Tarski's fixed point theorems. *Pacific J. Math.*, 82(1):43–57, 1979.
31. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. of Conf. Record of the 6th ACM Symp. on Principles of Programming Languages* (*POPL '79*), pages 269–282. ACM Press, New York, 1979.
32. P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *J. Logic Program.*, 13(2-3):103–179, 1992.
33. P. Cousot and R. Cousot. Inductive definitions, semantics and abstract interpretation. In *Proc. of Conf. Record of the 19th ACM Symp. on Principles of Programming Languages* (*POPL '92*), pages 83–94. ACM Press, New York, 1992.
34. P. Cousot and R. Cousot. Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and PER analysis of functional languages) (Invited Paper). In *Proc. of the 1994 IEEE Internat. Conf. on Computer Languages* (*ICCL '94*), pages 95–112. IEEE Computer Society Press, Los Alamitos, Calif., 1994.
35. P. Cousot and R. Cousot. On abstraction in software verification. In D. Brinksma and K.G. Larsen, editors, *Proc. of the 14th Internat. Conf. on Computer Aided Verification* (*CAV '02*), volume 2404 of *Lecture Notes in Computer Science*, pages 37–56. Springer-Verlag, Berlin, 2002.
36. A. Darvas, R. Hähnle, and D. Sands. A theorem proving approach to analysis of secure information flow. In Roberto Gorrieri, editor, *Workshop on Issues in the Theory of Security (WITS'03)*. IFIP WG 1.7, ACM SIGPLAN and GI FoMSESS, 2003.
37. B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, Cambridge, U.K., 1990.
38. D. E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–242, 1976.
39. D. E. Denning and P. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20(7):504–513, 1977.
40. J. Desharnais, B. Möller, and F. Tchier. Kleene under a demonic star. In *Proc. of the 9th Internat. Conf. on Algebraic Methodology and Software Technology* (*AMAST '00*), volume 1816 of *Lecture Notes in Computer Science*, pages 355–370. Springer-Verlag, 2000.
41. A. Di Pierro, C. Hankin, and H. Wiklicky. Approximate non-interference. In *Proc. of the IEEE Computer Security Foundations Workshop*, pages 1–17. IEEE Computer Society Press, 2002.
42. E. W. Dijkstra. *A discipline of programming*. Series in automatic computation. Prentice-Hall, 1976.
43. E.W. Dijkstra. Guarded commands, nondeterminism and formal derivation of programs. *Comm. of The ACM*, 18(8):453–457, 1975.
44. G. Filé, R. Giacobazzi, and F. Ranzato. A unifying view of abstract domain design. *ACM Comput. Surv.*, 28(2):333–336, 1996.
45. G. Filé and F. Ranzato. Complementation of abstract domains made easy. In M. Maher, editor, *Proc. of the 1996 Joint Internat. Conf. and Symp. on Logic Programming* (*JICSLP '96*), pages 348–362. The MIT Press, Cambridge, Mass., 1996.

46. C. Flanagan and S. Qadeer. Pedicate abstraction for software verification. In *Proc. of Conf. Record of the 29th ACM Symp. on Principles of Programming Languages* (*POPL '02*), pages 191–202. ACM Press, New York, 2002.
47. R. Focardi and R. Gorrieri. A classification of security properties for process algebras. *Journal of Computer security*, 3(1):5–33, 1995.
48. R. Focardi and R. Gorrieri. Classification of security properties (part I: Information flow). In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design*, volume 2171 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.
49. R. Giacobazzi and I. Mastroeni. A characterization of symmetric semantics by domain complementation. In *Proc. of the 2nd Internat. Conf. in Principles and Practice of Declarative Programming PPDP'00*, pages 115–126. ACM press, New York, 2000.
50. R. Giacobazzi and I. Mastroeni. Domain compression for complete abstractions. In *In proc. of the 4th Internat. Conf. on Verification, Model Checking and Abstract Interpretation (VMCAI'03)*, volume 2575 of *Lecture Notes in Computer Science*, pages 146–160. Springer-Verlag, 2003.
51. R. Giacobazzi and I. Mastroeni. Non-standard semantics for program slicing. *Higher-Order and Symbolic Computation (HOSC)*, 16(4):297–339, 2003. Special issue on Partial Evalution and Semantics-Based Program Manipulation.
52. R. Giacobazzi and I. Mastroeni. Abstract non-interference: Parameterizing non-interference by abstract interpretation. In *Proc. of the 31st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '04)*, pages 186–197. ACM-Press, NY, 2004.
53. R. Giacobazzi and I. Mastroeni. Generalized abstract non-interference for automata. 2004. Submitted for publication. http://profs.sci.univr.it/ mastroen/abstracts/genANI.abstract.html.
54. R. Giacobazzi and I. Mastroeni. Proving abstract non-interference. In *Annual Conference of the European Association for Computer Science Logic (CSL'04)*, volume 3210, pages 280–294. Springer-Verlag, 2004.
55. R. Giacobazzi and I. Mastroeni. Safety semantics by abstract interpretation, 2004. Submitted for publication. http://profs.sci.univr.it/ mastroen/abstracts/safety.abstract.html.
56. R. Giacobazzi and I. Mastroeni. Timed abstract non-interference. 2004. Submitted for publication. http://profs.sci.univr.it/ mastroen/abstracts/tANI.abstract.html.
57. R. Giacobazzi and I. Mastroeni. Adjoining declassification and attack models by abstract interpretation. In M. Sagiv, editor, *Proc. of the European Symposium on Programming (ESOP'05)*, Lecture Notes in Computer Science. Springer-Verlag, 2005. To appear.
58. R. Giacobazzi and I. Mastroeni. Transforming semantics by abstract interpretation. *Theoretical Computer Science*, 2005. Extended version of [49]. To appear.
59. R. Giacobazzi, C. Palamidessi, and F. Ranzato. Weak relative pseudo-complements of closure operators. *Algebra Universalis*, 36(3):405–412, 1996.
60. R. Giacobazzi and E. Quintarelli. Incompleteness, counterexamples and refinements in abstract model-checking. In P. Cousot, editor, *Proc. of The 8th Internat. Static Analysis Symp. (SAS'01)*, volume 2126 of *Lecture Notes in Computer Science*, pages 356–373. Springer-Verlag, 2001.
61. R. Giacobazzi and F. Ranzato. Refining and compressing abstract domains. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Proc. of the 24th*

Internat. Colloq. on Automata, Languages and Programming (*ICALP '97*), volume 1256 of *Lecture Notes in Computer Science*, pages 771–781. Springer-Verlag, Berlin, 1997.

62. R. Giacobazzi and F. Ranzato. Optimal domains for disjunctive abstract interpretation. *Sci. Comput. Program*, 32(1-3):177–210, 1998.
63. R. Giacobazzi and F. Ranzato. Uniform closures: order-theoretically reconstructing logic program semantics and abstract domain refinements. *Inform. and Comput.*, 145(2):153–190, 1998.
64. R. Giacobazzi and F. Ranzato. The reduced relative power operation on abstract domains. *Theor. Comput. Sci.*, 216:159–211, 1999.
65. R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract interpretations complete. *J. of the ACM.*, 47(2):361–416, 2000.
66. R. Giacobazzi and F. Scozzari. A logical model for relational abstract domains. *ACM Trans. Program. Lang. Syst.*, 20(5):1067–1109, 1998.
67. G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, and D. S. Scott. *A Compendium of Continuous Lattices*. Springer-Verlag, Berlin, 1980.
68. J. A. Goguen and J. Meseguer. Security policies and security models. In *Proc. IEEE Symp. on Security and Privacy*, pages 11–20. IEEE Computer Society Press, 1982.
69. J. A. Goguen and J. Meseguer. Unwinding and inference control. In *Proc. IEEE Symp. on Security and Privacy*, pages 75–86. IEEE Computer Society Press, 1984.
70. S. Graf and H. Saïdi. Construction of abstract state graphs with PVS. In *Proc. of the 9th Internat. Conf. on Computer Aided Verification* (*CAV '97*), volume 1254 of *Lecture Notes in Computer Science*, pages 72–83. Springer-Verlag, Berlin, 1997.
71. J. W. Gray III. Toward a mathematical foundation for information flow security. In *Proc. IEEE Symp. on Security and Privacy*, pages 21–34. IEEE Computer Society Press, 1991.
72. H. P. Gumm. Another glance at the alpern-schneider theorem. *Information Processing Letters*, 47:291–294, 1993.
73. M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. On protection in operating systems. In *5th Symp. on Operating Systems Principles*, pages 14–24, 1975.
74. C.A.R Hoare. An axiomatic basis for computer programming. *Comm. of The ACM*, 12(10):576–580, 1969.
75. C.A.R Hoare. *Communicating Sequential Processes*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1985.
76. M. F. Janowitz. Residuated closure operators. *Portug. Math.*, 26(2):221–252, 1967.
77. T. Jensen. Disjunctive program analysis for algebraic data types. *ACM Trans. Program. Lang. Syst.*, 19(5):751–803, 1997.
78. R. Joshi and K. R. M. Leino. A semantic approach to secure information flow. *Science of Computer Programming*, 37:113–138, 2000.
79. W. Just and M. Weese. *Discovering modern set theory. I: The basics*, volume 8 of *Graduate studies in mathematics*. American mathematical society, 1996.
80. B. Lampson. A note on the confinement problem. In *Communications of the ACM*, pages 613–615. ACM, 1973.
81. K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and computation*, 94(1):1–28, 1991.

82. P. Laud. Semantics and program analysis of computationally secure information flow. In *Programming Languages and Systems, 10th European Symp. On Programming, ESOP*, volume 2028 of *Lecture Notes in Computer Science*, pages 77–91. Springer-Verlag, 2001.
83. P. Li and S. Zdancewic. Downgrading policies and relaxed noninterference. In *Proc. of the 32st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '05)*. ACM-Press, NY, 2005. To appear.
84. G. Lowe. Quantifying information flow. In *Proc. of the IEEE Computer Security Foundations Workshop*, pages 18–31. IEEE Computer Society Press, 2002.
85. H. Mantel. Possibilistic definitions of security – an assemply kit –. In *Proc. of the IEEE Computer Security Foundations Workshop*, pages 185–199. IEEE Computer Society Press, 2000.
86. H. Mantel. Unwinding possibilistic security properties. In F. Cuppens et al., editor, *ESORICS*, volume 1895 of *Lecture Notes in Computer Science*, pages 238–254. Springer-Verlag, 2000.
87. G. Markowsky. Chain-complete p.o.sets and directed sets with applications. *Algebra universalis*, 6:53–68, 1976.
88. I. Mastroeni. Numerical power analysis. In *Proc. of the 2nd Symp. on Programs as Data Objects (PADO'01)*, volume 2053 of *Lecture Notes in Computer Science*. Springer, 2001.
89. I. Mastroeni. Algebraic power analysis by abstract interpretation. *Higher-Order and Symbolic Computation (HOSC)*, 17(4):299–347, 2004. Extended version of [88].
90. J. McLean. Security models and information flow. In *Proc. 1990 IEEE Symp. on Security and Privacy*, pages 180–187. IEEE Computer Society Press, 1990.
91. J. McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 79–93, 1994.
92. R. Milner. *Communication and Concurrency*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1989.
93. J. Morgado. Some results on the closure operators of partially ordered sets. *Portug. Math.*, 19(2):101–139, 1960.
94. J. Morgado. Note on complemented closure operators of complete lattices. *Portug. Math.*, 21(3):135–142, 1962.
95. A. Mycroft. Completeness and predicate-based abstract interpretation. In *Proc. of the ACM Symp. on Partial Evaluation and Program Manipulation* (*PEPM '93*), pages 179–185. ACM Press, New York, 1993.
96. A.C. Myers, A. Sabelfeld, and S. Zdancewic. Enforcing robust declassification. In *Proc. IEEE Symp. on Security and Privacy*. IEEE Computer Society Press, 2004.
97. G. Necula. Proof-carrying code. In *Proc. of Conf. Record of the 24th ACM Symp. on Principles of Programming Languages* (*POPL '97*), pages 106–119. ACM Press, New York, 1997.
98. F. Nielson. Tensor products generalize the relational data flow analysis method. In M. Arató, I. Kátai, and L. Varga, editors, *Proc. of the 4th Hungarian Computer Science Conf.*, pages 211–225, 1985.
99. G. Plotkin. A structural approach to operational semantics. DAIMI-19 Aarhus University, Denmark, 1981.
100. F. Ranzato. Closures on CPOs form complete lattices. *Inform. and Comput.*, 152(2):236–249, 1999.

101. F. Ranzato and F. Tapparo. Making abstract model checking strongly preserving. In M. Hermenegildo and G. Puebla, editors, *Proc. of The 9th Internat. Static Analysis Symp. (SAS'02)*, volume 2477 of *Lecture Notes in Computer Science*, pages 411–427. Springer-Verlag, 2002.
102. F. Ranzato and F. Tapparo. Strong preservation as completeness in abstract interpretation. In D. Schmidt, editor, *Proc. of the 13th European Symposium on Programming (ESOP'04)*, volume 2986 of *Lecture Notes in Computer Science*, pages 18–32. Springer-Verlag, 2004.
103. A. Sabelfeld and A. C. Myers. A model for delimited information release. In *Proc. of the International Symp. on Software Security (ISSS'03)*, Lecture Notes in Computer Science. Springer-Verlag, 2004.
104. A. Sabelfeld and A.C. Myers. Language-based information-flow security. *IEEE J. on selected ares in communications*, 21(1):5–19, 2003.
105. A. Sabelfeld and D. Sands. Probabilistic noninterference for multi-threaded programs. In *Proc. of the IEEE Computer Security Foundations Workshop*, pages 200–214. IEEE Computer Society Press, 2000.
106. A. Sabelfeld and D. Sands. A PER model of secure information flow in sequential programs. *Higher-Order and Symbolic Computation*, 14(1):59–91, 2001.
107. F. Scozzari. Logical optimality of groundness analysis. *Theoretical Computer Science*, 277(1-2):149–184, 2002.
108. G. Smith and D. Volpano. Secure information flow in a multi-threaded imperative language. In *Proc. of The 25th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, pages 355–364. ACM Press, New York, 1998.
109. G. Smith and D. Volpano. Confinement properties for multi-threaded programs. volume 20 of *Electronic Notes in Theoretical Computer Science*. Elsevier, Amsterdam, 1999.
110. A. Tarski. A lattice theoretical fixpoint theorem and its applications. *Pacific J. Math.*, 5:285–310, 1955.
111. D. Volpano. Safety versus secrecy. In *Proc. of the 6th Static Analysis Symp. (SAS'99)*, volume 1694 of *Lecture Notes in Computer Science*, pages 303–311. Springer-Verlag, 1999.
112. D. Volpano and G. Smith. Eliminating covert flows with minimum typing. In *Proc. of the IEEE Computer Security Foundations Workshop*, pages 156–168. IEEE Computer Society Press, 1997.
113. D. Volpano and G. Smith. Probabilistic noninterference in a concurrent language. *Journal of Computer Security*, 7(2,3):231–253, 1999.
114. D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(2,3):167–187, 1996.
115. M. Ward. The closure operators of a lattice. *Ann. Math.*, 43(2):191–196, 1942.
116. G. Winskel. *The formal semantics of programming languages: an introduction*. MIT press, 1993.
117. A. Zakinthinos and E. S. Lee. A general theory of security properties. In *Proc. IEEE Symp. on Security and Privacy*, pages 94–102. IEEE Computer Society Press, 1997.
118. S. Zdancewic and A. C. Myers. Robust declassification. In *Proc. of the IEEE Computer Security Foundations Workshop*, pages 15–23. IEEE Computer Society Press, 2001.

Sommario

In questa tesi, mostriamo come è possibile utilizzare l'interpretazione astratta per certificare il grado di sicurezza dei programmi. In particolare, l'idea è quella di modellare gli attaccanti come domini astratti e poi trasformare questi domini al fine di manipolare gli attaccanti. Per questo motivo, le nozioni centrali in questa tesi sono quelle di *trasformatore di domini astratti* e di *flussi di informazione sicura* (chiamati anche non-interferenza). Per prima cosa studiamo come possiamo progettare, classificare e confrontare trasformatori di domini astratti. In particolare, mostriamo che la teoria standard dell'interpretazione astratta dei coniugi Cousot, basata sulla nozione di aggiunzione ovvero di connessione di Galois, può essere direttamente applicata per ragionare su trasformatori di domini astratti, fornendo delle metodologie formali per la progettazione siste-matica di questi trasformatori. Il punto chiave è che nella maggior parte dei casi i trasformatori di domini possono essere visti come precisi problemi di raffinamento della precisione relativamente a qualche aspetto semantico del linguaggio di programmazione che vogliamo analizzare. Questa è esattamente la filosofia che ci ha portato a caratterizzare il trasformatore di domini astratti che costruisce il più concreto attaccante incapace di violare la non-interferenza in un dato programma, ovvero il più concreto osservatore pubblico che non riesce ad acquisire nessuna informazione privata. Di fatto, l'argomento principale di questa tesi è la definizione della nozione di *non-interferenza astratta*, ottenuta parametrizzando la nozione standard di non-interferenza relativamente a ciò che un attaccante è in grado di osservare del comportamento I/O di un programma. Questa nozione è ciò di cui abbiamo bisogno per caratterizzare il grado di sicurezza dei programmi nel reticolo delle interpretazioni astratte, derivando il più potente attaccante per il quale un programma risulta essere sicuro. La definizione di non-interferenza astratta dipende della semantica del programma. Questo significa che possiamo rendere più precisa questa nozione semplicemente arricchendo la semantica corrispondente.
La non-interferenza astratta è un indebolimento della nozione standard di non-interferenza, ma non è il primo lavoro con questo obiettivo, per questa ragione confrontiamo la non-interferenza astratta con due dei lavori più affini: il modello

che utilizza le relazioni di equivalenza parziale (PER) [106] e la declassificazione robusta [118].
Al fine di rendere la certificazione della non-interferenza astratta più pratica, introduciamo un sistema di prova composizionale, il cui scopo è quello di certificare la non-interferenza astratta dei programmi, induttivamente sulla sintassi del linguaggio di programmazione, seguendo una deduzione di segretezza *a la* Hoare. A questo punto mostriamo come il problema della non-interferenza astratta può essere formalizzato come problema di completezza, nella teoria standard dell'interpretazione astratta. Questo permette di caratterizzare la derivazione del più concreto osservatore pubblico, ovvero il più potente attaccante innocuo, e la derivazione del più astratto osservabile privato del programma, come trasformatori di domini che stanno in relazione di aggiunzione tra loro. Questa relazione di aggiunzione formalizza l'intuitivo dualismo esistente tra questi due approcci utilizzati per indebolire la nozione di non-interferenza.
Concludiamo la tesi mostrando come possiamo arricchire ulteriormente la non-interferenza astratta aggiungendo l'osservazione del tempo e generalizzando la non-interferenza astratta al fine di modellare il problema del confinamento anche per sistemi computazionali che non sono linguaggi di programmazione. Per questo introduciamo la nozione di *non-interferenza astratta con tempo*, la quale costituisce l'ambiente ideale in cui studiare come proprietà degli input privati interferiscono con proprietà riguardanti il tempo di esecuzione del programma. In questo modo otteniamo una nozione di non-interferenza astratta che impedisce i canali di informazione dovuti alla capacità dell'attaccante di osservare il tempo di esecuzione. Infine dimostriamo che la non-interferenza astratta può essere generalizzata al fine di modellare molti dei noti modelli usati per la non-interferenza in sistemi sequenziali, concorrenti e real-time. Questo è ottenuto fattorizzando le astrazioni al fine di ottenere delle sotto-astrazioni che modellano i differenti aspetti della nozione di non-interferenza che dobbiamo formalizzare. In particolare, un'astrazione decide il modello del sistema usato per definire la nozione di non-interferenza, ad esempio la semantica denotazionale nei linguaggi imperativi. Un'ulteriore astrazione decide gli aspetti della computazione che possono essere osservabili nel nostro sistema secondo la nostra politica di sicurezza, ad esempio solo le computazioni in cui le azioni private non vengono eseguite. Infine, l'ultima astrazione consi-derata caratterizza quali proprietà della computazione l'attaccante effettivamente osserva. Queste tre astrazioni sono composte tra loro al fine di ottenere la non-interferenza astratta generalizzata, e a seconda di come definiamo queste astrazioni decidiamo la nozione di non-interferenza che vogliamo garantire nel nostro sistema.

Index