A Static Analysis Framework for Security Properties in Mobile and Cryptographic Systems

Benyamin Y. Y. Aziz, M.Sc.

School of Computing, Dublin City University

A thesis presented in fulfillment of the requirements
for the degree of Doctor of Philosophy

Supervisor: Dr Geoff Hamilton

September 2003

"Start by doing what's necessary; then do what's possible; and suddenly you are doing the impossible"

St. Francis of Assisi

To Yowell, Olivia and Clotilde

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of the degree of Doctor of Philosophy (Ph.D.) is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed:_____

I.D. No.:_____

Date:_____

Acknowledgements

I would like to thank all those people who were true sources of inspiration, knowledge, guidance and help to myself throughout the period of my doctoral research. In particular, I would like to thank my supervisor, Dr. Geoff Hamilton, without whom this work would not have seen the light. I would also like to thank Dr. David Gray, with whom I had many informative conversations, and my colleagues, Thomas Hack and Frédéric Oehl, for their advice and guidance. Finally, I would like to mention that the work of this thesis was partially funded by project IMPROVE (Enterprise Ireland Strategic Grant ST/2000/94).

Benyamin Aziz

Abstract

We introduce a static analysis framework for detecting instances of security breaches in infinite mobile and cryptographic systems specified using the languages of the π-calculus and its cryptographic extension, the spi calculus. The framework is composed from three components: First, standard denotational semantics of the π-calculus and the spi calculus are constructed based on domain theory. The resulting model is sound and adequate with respect to transitions in the operational semantics. The standard semantics is then extended correctly to non-uniformly capture the property of term substitution, which occurs as a result of communications and successful cryptographic operations. Finally, the non-standard semantics is abstracted to operate over finite domains so as to ensure the termination of the static analysis. The safety of the abstract semantics is proven with respect to the non-standard semantics. The results of the abstract interpretation are then used to capture breaches of the secrecy and authenticity properties in the analysed systems. Two initial prototype implementations of the security analysis for the π-calculus and the spi calculus are also included in the thesis.

The main contributions of this thesis are summarised by the following. In the area of denotational semantics, the thesis introduces a domain-theoretic model for the spi calculus that is sound and adequate with respect to transitions in the structural operational semantics. In the area of static program analysis, the thesis utilises the denotational approach as the basis for the construction of abstract interpretations for infinite systems modelled by the π-calculus and the spi calculus. This facilitates the use of computationally significant mathematical concepts like least fixed points and results in an analysis that is fully compositional. Also, the thesis demonstrates that the choice of the term-substitution property in mobile and cryptographic programs is rich enough to capture breaches of security properties, like process secrecy and authenticity. These properties are used to analyse a number of mobile and cryptographic protocols, like the file transfer protocol and the Needham-Schroeder, SPLICE/AS, Otway-Rees, Kerberos, Yahalom and Woo Lam authentication protocols.

Contents

1 Introduction **1**

1.1 The Problem of Computer Security 1

1.2 Mobile Systems 3

1.3 Denotational Semantics 6

1.4 Static Program Analysis 8

1.4.1 Data Flow Analysis 8

1.4.2 Control Flow Analysis 11

1.4.3 Abstract Interpretation 13

1.4.4 Type Systems 15

1.5 Security Properties 18

1.5.1 Secrecy 18

1.5.2 Authenticity 21

1.6 Further Reading 23

1.7 Outline of Our Approach 23

2 Related Work **25**

2.1 Introduction 25

2.2 Static Analysis Techniques for Program Security 25

2.2.1 Control Flow Analysis 25

2.2.2 Abstract Interpretation 26

2.2.3 Type Systems 27

2.2.4 Other Approaches 30

2.3 Denotational Semantics of Nominal Calculi 31

2.4 Conclusion 33

3 Nominal Calculi **34**

3.1 Introduction 34

3.2 The π-calculus . . . 34
3.2.1 Syntax . . . 35
3.2.2 Structural Operational Semantics . . . 36
3.2.3 Denotational Semantics . . . 37
3.3 The Spi Calculus . . . 42
3.3.1 Syntax . . . 43
3.3.2 Structural Operational Semantics . . . 45
3.3.3 Denotational Semantics . . . 46
3.4 Conclusion . . . 52

4 Abstract Interpretation **53**
4.1 Introduction . . . 53
4.2 The π-calculus . . . 54
4.2.1 Non-standard Semantics . . . 54
4.2.2 Abstract Semantics . . . 57
4.2.3 The Intruder I . . . 62
4.2.4 The FTP Server Example . . . 63
4.3 The Spi Calculus . . . 67
4.3.1 Non-standard Semantics . . . 68
4.3.2 Abstract Semantics . . . 71
4.3.3 The Intruder I . . . 78
4.3.4 The Needham-Schroeder Public-Key Protocol Example . . . 79
4.3.5 The SPLICE/AS Protocol Example . . . 83
4.3.6 The Otway-Rees Protocol Example . . . 86
4.3.7 The Kerberos Protocol Example . . . 90
4.3.8 The Yahalom Protocol Example . . . 93
4.3.9 The Woo-Lam One-Way Authentication Protocol Example . . . 95
4.4 Conclusion . . . 98

5 Security Properties **99**
5.1 Introduction . . . 99
5.2 Secrecy . . . 99
5.2.1 Mobile Systems . . . 100
5.2.2 Cryptographic Protocols . . . 101
5.3 Authenticity . . . 102
5.3.1 Mobile Systems . . . 103

5.3.2 Cryptographic Protocols . . . 104
5.4 Examples . . . 105
5.4.1 The FTP Server Example . . . 105
5.4.2 The Needham-Schroeder Public-Key Protocol Example . . . 107
5.4.3 The SPLICE/AS Protocol Example . . . 109
5.4.4 The Otway-Rees Protocol Example . . . 110
5.4.5 The Kerberos Protocol Example . . . 112
5.4.6 The Yahalom Protocol Example . . . 114
5.4.7 The Woo-Lam Protocol Example . . . 115
5.5 Conclusion . . . 117

6 Automatic Tools **118**
6.1 Introduction . . . 118
6.2 Picasso: A Pi-Calculus Analyser for Secrecy and Security Objectives . . . 118
6.3 Spicasso: A Spi-Calculus Analyser for Secrecy and Security Objectives . . . 123
6.4 Conclusion . . . 127

7 Conclusion and Future Work **128**
7.1 Research Contributions . . . 130
7.1.1 Denotational Semantics . . . 130
7.1.2 The Static Analysis of Nominal Calculi . . . 130
7.1.3 Program Security . . . 131
7.2 Future Work . . . 132
7.2.1 Communication Secrecy . . . 132
7.2.2 Message Independence . . . 132
7.2.3 Language Extensions . . . 134

A Proofs **150**
A.1 Safety of the $\cup_\phi$ operation in the π-calculus (Lemma 1) . . . 150
A.2 Safety of the abstract semantics of the π-calculus (Theorem 5) . . . 152
A.3 Safety of the $\cup_\phi$ operation in the spi calculus (Lemma 2) . . . 160
A.4 Safety of the abstract semantics for the spi calculus (Theorem 8) . . . 162

List of Figures

1.1 The flow graph of the factorial program. 9
1.2 The solution of the reaching definitions analysis for the factorial program. . . 10
1.3 Types for the reaching definitions analysis. 16

3.1 Syntax of the π-calculus. 35
3.2 Rules of the labelled transition relation in the π-calculus. 36
3.3 Elements of *In*, *Out* and $Pi_{\perp}$. 38
3.4 The definition of *new* for the π-calculus. 39
3.5 The denotational semantics of the π-calculus. 40
3.6 The syntax of the spi calculus. 44
3.7 Rules of the labelled transition relation in the spi calculus. 46
3.8 Elements of the predomain of terms T. 48
3.9 Elements of *In*, *Out*, and $Spi_{\perp}$. 48
3.10 The definition of *new* for the spi calculus. 49
3.11 The denotational semantics of the spi calculus. 50

4.1 The non-standard semantics of the π-calculus. 56
4.2 The abstract semantics of the π-calculus. 59
4.3 The non-standard semantics of the spi calculus. 69
4.3 The non-standard semantics of the spi calculus (continued). 70
4.4 The abstract semantics of the spi calculus. 74
4.4 The abstract semantics of the spi calculus (continued). 75
4.5 Specification of the Dolev-Yao attacker in the spi calculus. 79
4.6 Specification of the Needham-Schroeder protocol. 81
4.7 Results of analysing the Needham-Schroeder protocol. 82
4.8 Specification of the SPLICE/AS protocol. 84
4.9 Results of analysing the SPLICE/AS protocol. 85

4.10 The specification of the Otway-Rees protocol. 87
4.11 Results of analysing the Otway-Rees protocol. 88
4.12 The specification of the Kerberos protocol. 91
4.13 The results of analysing the Kerberos protocol. 92
4.14 The specification of the Yahalom protocol. 94
4.15 The results of analysing the Yahalom protocol. 95
4.16 The specification of the Woo-Lam protocol. 96
4.17 The results of analysing the Woo-Lam protocol. 97

5.1 Rules of the $\mathcal{Z}(\lceil P \rceil^l)$ ζ function for π-calculus processes. 101
5.2 Rules of the $\mathcal{Z}(\lceil P \rceil^l)$ ζ function for spi calculus processes. 102
5.3 Rules of the $\mathcal{U}(\lfloor P \rfloor^a)$ θ function for π-calculus processes. 104
5.4 Rules of the $\mathcal{U}(\lfloor P \rfloor^a)$ θ function for spi calculus processes. 105

6.1 The `name` and `process` data types in Picasso. 119
6.2 The `proc_level` data type in Picasso. 121
6.3 The `term` data type in Spicasso. 123
6.4 The `process` data type in Spicasso. 124

Chapter 1

Introduction

1.1 The Problem of Computer Security

The issue of security in computing systems developed as an independent research discipline in the early 1970s, although much of the earlier work had been carried out as part of other research areas, particularly in relation to operating systems and databases. Despite the fact that many of the early papers in computer security are difficult to obtain nowadays, there are some entities that maintain collections of such papers. One example is the comprehensive collection provided by NISTs Computer Security Resource Center (CSRC)[1].

The early forms of computer security were restricted to the protection of data and programs running on isolated monolithic mainframe machines or on limited multi-user or multi-program systems. Such protection mainly consisted of access control and authorization mechanisms and relied, to a large extent, on protecting the physical access to these machines. Later, the introduction and success of computer networking gave a new perspective to computer security. Networks meant that users and machines could share information over a wide range of distances. The implications soon became clear: data and files could be vulnerable to attacks from other users/programs not only within the machines themselves, but also over the network. Cryptography was utilised as an effective solution to the protection of data and programs from these new threats.

In the last decade or so, the issue of computer security has become even more intricate and sophisticated with the advent of wide area distributed systems like the Internet, and dynamic mobile technologies. Such advancements not only facilitated the integration of computers into every day life activities, but also provided features like transparency, anonymity, and

[1]The collection is available at: `http://csrc.nist.gov/publications/history/`

mobility, all of which opened the gates for new forms of computer and information attacks to be launched. In almost every research area related to the modern technologies underlying e-commerce, global communications and pervasive computing, the element of security is a necessity for the protection of our identities, data and material property.

Nowadays, it is often a common practice to use credit cards to purchase goods and pay for services online over the Internet. Taking this practice as an example, the security threats posed are several: fraud resulting from compromising the secrecy of credit card information by malicious attackers, the integrity of the data transmitted during a payment session, the authenticity of the site offering goods or services, anonymity and non-repudiation issues related to transactions, etc. Cryptographic protocols have been devised to resolve the problems of secrecy and authenticity by using secret-key cryptography and public-key infrastructure to ensure that communications maintain certain levels of security requirements.

As a second example, consider the downloadable code fragments that often enter the address space of computers connected to the Internet. Special treatment of mobile code is necessary in order to avoid leaking private information to external sources as well as damaging internal data. Therefore, safety measures like bounded sandboxes, which prevent applets from accessing local file systems and initiating ad-hoc network connections, have been devised.

All these new demands for computer security have prompted the task of designing and implementing more robust systems that are inherently secure. An important aspect of this task is *program analysis*, which helps the understanding of the way programs will behave and their properties once executed. Broadly speaking, one may divide program analysis into two categories: runtime and compile-time analysis. Runtime analysis relies on dynamic techniques that gather information about programs during their execution or testing. This information is then used to reason about the properties of a program and subsequently will affect the way data and control flow in that program. In general, runtime analysis has focused on traditional ad-hoc monitoring techniques used in performance monitoring, distributed debugging, etc. and has not yet demonstrated much use of formal methods techniques (except for a few examples like runtime type checking and program specification guidance). On the other hand, compile-time analysis has benefited much from the use of formal methods techniques, in particular model checking, theorem proving and abstract interpretation. Unlike the runtime analysis, which has a real-time narrow view of a program computation, a compile-time analysis is only applicable at the compilation stage and can approximate properties about all the possible computations of that program.

The approach to the problem of detecting security threats in computer systems as con-

tributed by this thesis involves the introduction of an abstract interpretation-based static analysis framework for the verification of security-related properties of infinite mobile and cryptographic systems whose meaning is defined by a denotational semantics. The framework is general; it can be accommodated to different languages, properties and abstractions. However, in our case, we target closed systems specified with the language of the π-calculus and its cryptographic-extension, the spi calculus, and deal with open systems by modelling the intruder within the specification. We capture the name-passing behaviour within mobile systems and the term-passing and term-processing[2] behaviours within cryptographic protocols through the abstract interpretation of programs. This abstract interpretation is non-uniform; the number of copies of each term appearing in the results of the interpretation can be adjusted depending on the nature of properties sought. The information obtained from the abstract interpretation is then used to further analyse these programs to detect the presence of any secrecy or authenticity breaches.

In the rest of this introduction, we describe the basics underlying four main components of our static analysis framework. These include mobile systems (and their security-related extensions), denotational semantics, static program analysis, and computer security. Each of these components constitutes a major area of research that overlaps with the other areas. We only give a brief overview of each of these components in a manner that is specific to our framework and avoid going into too much detail, referring enthusiastic readers to references on each subject involved.

1.2 Mobile Systems

The word *mobile* in the real world is normally used to describe the state of any object, location, condition etc. that is moving with respect to some reference. In the context of computing systems, mobility may refer to the movement of communication channels, code, or whole computing environments. For example, HTML links can be created, sent to other entities and destroyed later. References to objects in object-oriented programming are created and passed around as capabilities of communication. In Java, applet code embedded in Web pages can be downloaded and executed dynamically at runtime by the host machine. The movement of "intelligent" mobile agents constitutes a form of code and state mobility. Finally, the emerging component-based technologies and pervasive/ubiquitous computing are interesting examples of mobile computing environments.

[2]Term processing is the term we use to refer to the cryptographic operations performed by processes on terms in the spi calculus.

From a process algebraic point of view, there are two widely accepted definitions of mobility [96]:

1. *Link-based mobility.*

   The concept of link-based mobility states that the movement of a process among other processes can in fact be described as the proliferation, change and extinction of the communication channels linking that process to the rest. In other words, this corresponds to the movement of links in the virtual space of linked processes. The basic notion of communication here is that of process interaction, where processes interact to exchange links. This induces the behaviour of message passing. The resulting mobility is expressively powerful and can be used to encode the higher-order notion of process-based mobility, in which whole processes (not just links) can move in the virtual space of linked processes. The π-calculus [97, 96, 114] is among the most authoritative models that embrace the link-based definition of mobility.

2. *Location-based mobility.*

   Unlike the link-based definition of mobility, which describes a virtual movement of links, the location-based definition relies on the physical aspect of mobility. According to this definition, mobility is the movement of processes in the space of locations. For example, the movement of a laptop from one local area network to another is viewed as the movement of a computing process (the laptop) from one parent location (first LAN) to another (second LAN). The best example of a formalism that adopts the concept of location-based mobility is the Mobile Ambients calculus [38], which is mainly influenced by Internet programming and the presence of administrative domains that divide wide area networks in general.

By and large, the modelling of mobility in computing systems has benefited a good deal from the body of theory that was developed earlier for the modelling of static concurrent and distributed systems. Formalisms such as Petri nets [107], CSP [75] and CCS [93] provided the necessary mature ground for the arrival of the first substantial theory of mobility; the π-calculus. The development of the π-calculus was directly inspired by the calculus of [50], where label-passing was added to the theory of CCS to model the dynamic configuration of networks. In the π-calculus, the theory was further simplified by adopting the unique notion of a *name* (hence identifying variables and constants). Names refer to channels of communication and can be communicated over other names. The concept of mobility is grasped by allowing processes to exchange names of channels, and hence modify the network configuration dynamically.

The π-calculus is a highly expressive language that is also capable of encoding statically distributed systems and functional programming (λ-calculus). The language is characterized as being directly executable and has formed the basis for other programming languages like Piccola [9], Join [64], Pict [108] and Nomadic Pict [132]. It has also provided the basis for many extended models that are concerned with different aspects and properties of mobile systems. The extension we are interested in is the spi calculus [5], which extends the language of the π-calculus by the addition of cryptographic primitives like encryption/decryption and digital signing/verification. Reasoning about the different properties of cryptographic protocols is based on a theory of testing-equivalence. Properties like privacy and authentication are defined as equivalences in the presence of intruder processes.

The popularity of the π-calculus led to the development of the Mobile Ambients calculus that adopts a different approach in the modelling of mobility. Unlike the location-transparent π-calculus, the Mobile Ambients adopts the notion of an *ambient* as its main idea and it is most suitable for the modelling of computing agents that can move from one location to another. Ambients are bounded places where computations can take place. The boundary of an ambient is significant to its movement, since it determines exactly what entity will move. It is also significant from the security perspective as it acts as an access control mechanism determining what boundaries can or cannot be crossed. Examples of ambients include a virtual address space, a laptop, a Unix file system and a single Java object. Furthermore, ambients may be nested within other ambients.

Amongst other formalisms, which adopt definitions of mobility that overlap definitions (1) and (2) above and that have been shown to be interesting, is the seal calculus [129]. *Seals* are named locations that are passed around and can have portals opened for remote communications occurring with the parent and child seals, while allowing for local communications to take place directly within the same seal. The seal calculus extends the polyadic π-calculus [95], and in comparison to the Mobile Ambients calculus, it adopts an *objective* movement of locations; movement is initiated by the environment surrounding a seal. The Mobile Ambients calculus, on the other hand, adopts *subjective* mobility allowing ambients to initiate the movement.

The seal calculus elegantly models Internet programming. It adheres to a number of principles that are reminiscent of Internet-like programming. These include the distinction between remote and local communications, locations, restricted connectivity and access control. In particular, the modelling of security notions that rely on locations and scoping, like the perfect firewall equation, is straightforward.

One of the interesting issues currently debated within the area of mobility formalisms

and security is whether there is a need for combining location-based mobility languages, like the Mobile Ambients and seal calculi, with cryptographic primitives in a similar manner to the spi calculus. The analogy between the concepts of a mobile location and a ciphertext encrypted with a symmetric key is prevalent. However, it is less obvious when dealing with asymmetric-key cryptography and operations like hashing. Nonetheless, the modelling of mechanisms, like remote communications (e.g. RPC and RMI), could well benefit from such a combination.

In our static analysis framework, we adopt the π-calculus as the main specification language for mobile systems. We also adopt its security extension, the spi calculus, for the specification of mobile systems enhanced with cryptography. This decision is motivated by two reasons: first, the fact that both languages are more mature than any of the location-based languages means that more theory is available. Second, there are very few cryptographic extensions of the location-based languages [111], therefore, cryptographic protocols have not been studied in light of such models. Nonetheless, extending the current framework to the Mobile Ambients calculus and/or the seal calculus should benefit our cause and will provide additional understanding of how security mechanisms, in general, behave in mobile systems.

1.3 Denotational Semantics

The denotational (also known as the mathematical) approach to the definition of the semantics of programming languages was initiated by Christopher Strachey and instrumented by Dana Scott in the late 1960s and early 1970s [118, 117, 123, 115, 122, 131]. The idea suggested by Scott and Strachey was to develop a mathematical framework within which the formal semantics of programming languages could be specified without the traditional implementation-dependant problems associated with operational semantics and that would rely on the rigor that mathematics offers. Indeed, this framework later became an inspiration for computer language designers and implementers.

The basis of any denotational model is that syntactic phrases are realisations of abstract mathematical objects. For example, in a calculator device, strings of digits are perceived as abstract numerical ideas regardless of the format in which those digits are presented on the screen. Another example is the functional view that programs stand for mathematical functions, and the execution of a program, given some input data, resembles the application of a function to its parameters.

In general, for any syntactically correct program, there exists a mathematical object known as the *denotation* of that program that expresses the meaning of the program in a

clear and non-circular (i.e. needless of further definitions) manner. Based on this, one may divide a denotational model into three components:

1. *Syntactic domains.* These are the collection of entities that constitute the syntactical representation of the language constructs and whose meaning is sought. Syntactic domains include digits, numerals, expressions, instructions, phrases and programs. The syntactic domains we adopt here are *abstract* [89], as opposed to the concrete syntactic domains, which normally introduce unnecessary syntactic sugar useful only for the parsing of programs and does not contribute to their meaning. Furthermore, the syntactic notation we adopt is a version of the Backus-Naur Form (BNF) [17], which describes context-free grammars. Classical readings on the subject of syntax include [41, 16].

2. *Semantic domains.* These are collections of mathematical objects (denotations) that convey the meaning of the syntactical entities. Elements of these objects usually have some structure, like complete partial orders (CPOs), lattices or domains, whose algebra is determined by domain theory. The semantic elements are usually classified as either primitive or complex elements, where the bottom element in these domains often denotes the undefined program. Primitive elements constitute the *atomic* semantic elements whereas complex elements are necessary to convey more sophisticated ideas that can be decomposed back to the primitive elements. For example, it is common in the theory of the π-calculus to interpret parallelism in terms of the simpler notions of input/output and non-determinism.

3. *Semantic functions.* These are special functions that map programs, phrases etc. from their syntactic domains to their denotations in the semantic domains. Often, there are certain requirements that need to be satisfied by semantic functions, for example, being monotonic and continuous.

In general, a denotational semantics has to respect the principle of *compositionality*. Compositionality states that the meaning of a program can be defined in terms of the meanings of its subprograms. A discussion on the principle of compositionality in the definition of language semantics can be found in [125]. Also, the well known *Pisa Notes*[3] on domains compiled by Gordon Plotkin are a traditional reading in the subject of domain theory and its application to semantics.

One of the main advantages that the denotational semantics approach introduced is that the behaviour of any program could be determined directly through the mathematics of

[3]Notes can be downloaded from `http://www.dcs.ed.ac.uk/home/gdp/publications/Domains.ps.gz`

domain theory without the need to execute that program, and consequently, without the need to design any language compilers or interpreters. As a result, program verification and comparison becomes an easier task compared to other approaches to the definition of language semantics.

The fact that the denotational approach relies, to a large degree, on domain theory means that several powerful mathematical tools become available for reasoning about program semantics. Mainly, concepts like CPOs, continuous functions and least fixed points are easier to express and implement. For example, it is often the case that demonstrating the termination of a static analysis is dependent on the evaluation of least fixed points.

Finally, the idea of using semantic functions to denote the meaning of programs created a close relationship between the theory of denotational semantics and the functional programming paradigm. The task of implementing denotational interpreters using functional languages becomes fairly straightforward. Any static analysis that, in turn, is based on the denotational semantics of the language can also be directly implemented as a higher-level abstraction (i.e. semantics-directed). The implementations of fixed points are also standard in functional languages.

1.4 Static Program Analysis

Quite often, it is desirable to predict in advance the set of values resulting from programs and verify certain properties regarding their runtime behaviour. For this purpose, the area of static program analysis offers compile-time computable techniques that can be used to safely approximate properties and values of programs without the need to execute them directly on computers. The functionality covered by static analysers is wide and ranges from simple syntactic verifications that can be used in program transformations to complex runtime properties related to issues of security and optimisation. In general, many approaches exist for building static analysers. Here, we distinguish four main approaches [102].

1.4.1 Data Flow Analysis

Motivated by the aim of producing smaller and faster programs, the main application area of the data flow analysis approach has always been the performance optimisation of program code generated by compilers. Other important applications that benefit from data flow analysis include program testing, validation, verification, parallelization and vectorization. Classical data flow analyses include reaching definitions, available expressions and live variables analyses [102].

A data flow analysis is primarily designed to gather information about the use and definition of data in a program as well as the dependencies between the different sets of data. To achieve this, a program is often seen as a graph, where nodes represent blocks of the program and edges represent flows between those blocks. For example, the following program computes the factorial of a number n:

$$F \stackrel{\texttt{def}}{=} [fac := 1]^1;\ \texttt{while}\ [n > 1]^2\ \texttt{do}\ ([fac := fac * n]^3; [n := n - 1]^4)$$

This program can be represented by the graph of Figure 1.1, and can also be expressed by the function $\mathit{flow} = \{(1,2),(2,3),(3,4),(4,2)\}$. Sets of equations or constraints are then

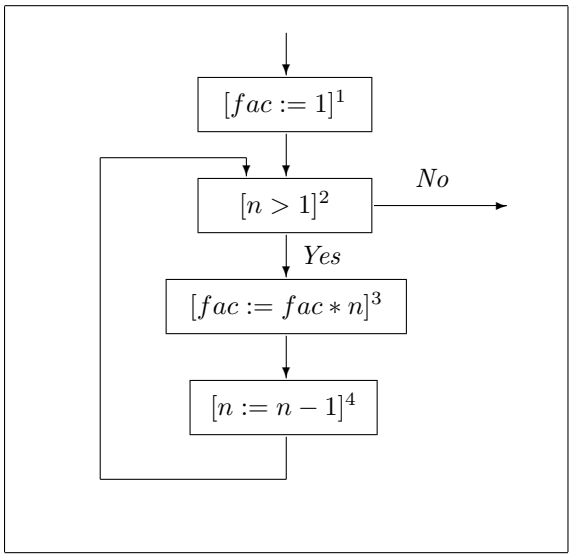

Figure 1.1: The flow graph of the factorial program.

constructed that relate the entry and exit information of each program node and among the different nodes. The least solution of these equations represents the result of the analysis. In the above example of the factorial program, one may construct a *reaching definitions analysis* by defining the following set of equations [102]:

$$\begin{aligned}
\mathit{Rout}(1) &= (\mathit{Rin}(1) \backslash \{(fac, l)\}) \cup \{(fac, 1)\} \\
\mathit{Rout}(2) &= \mathit{Rin}(2) \\
\mathit{Rout}(3) &= (\mathit{Rin}(3) \backslash \{(fac, l)\}) \cup \{fac, 3\} \\
\mathit{Rout}(4) &= (\mathit{Rin}(4) \backslash \{(n, l)\}) \cup \{(n, 4)\} \\
\mathit{Rin}(2) &= \mathit{Rout}(1) \cup \mathit{Rout}(4) \\
\mathit{Rin}(3) &= \mathit{Rout}(2) \\
\mathit{Rin}(4) &= \mathit{Rout}(3)
\end{aligned}$$

Where l is any label, $l \in \{1, 2, 3, 4\}$. $Rout(l)$ here relates information at the exit point of a node, l, in terms of the inputs to that node, and $Rin(l)$ relates information about the entry point of node, l, in terms of the outputs from that point. The restriction, $Rin(l') \backslash \{(x, l)\}$, and the union, $Rin(l') \cup \{(x, l')\}$, refer to the change that occurs in the value of variable x at program node l', where the previous value of x was obtained at node l. On the other hand, $Rin(l)$ reflects the information collected from nodes whose outputs flow into the input of node l. Hence, $Rout(l')$ is added to the value of $Rin(l)$ whenever control flow is possible from node l' to node l.

In a reaching definitions analysis, the information collected about a program point represents the assignments that can reach that point, i.e. assignments not overwritten by the time the point is reached. Hence, solving the above equations will result in the least solution of Figure 1.2 (the question mark '?' symbol denotes unknown value).

l	$Rin(l)$	$Rout(l)$
1	$(n, ?), (fac, ?)$	$(n, ?), (fac, 1)$
2	$(n, ?), (n, 4), (fac, 1), (fac, 3)$	$(n, ?), (n, 4), (fac, 1), (fac, 3)$
3	$(n, ?), (n, 4), (fac, 1), (fac, 3)$	$(n, ?), (n, 4), (fac, 3)$
4	$(n, ?), (n, 4), (fac, 3)$	$(n, 4), (fac, 3)$

Figure 1.2: The solution of the reaching definitions analysis for the factorial program.

Data flow analysis is a mature research area, where frameworks have been developed to classify and solve different classes of data flow problems. These problems can be described using the formalisms of the different frameworks and a solution algorithm is then selected. The earliest framework for data flow analysis can be ascribed to Kildall [84], who was also the first to use semi-lattices in such analyses. A survey can be found in [88], which includes the rapid, continuous, distributive, monotone and k-bound frameworks.

The most popular framework for data flow analysis is the monotone framework due to [83]. In this framework, data flow problems are defined through lattices (either complete or semi-lattices) of values with a meet (join) operator, often called the *property space*, and a family of transfer functions defined over those lattices. To create an instance of the framework, a (directed) flow graph is required, where the transfer functions are bound to the graph nodes using a function map. The association of a function to a particular node means that all the information gathered at the node is mapped to lattice values. This also allows for data flow problems to be phrased as equations that encode the information at each node in terms of the information at the predecessor (successor) nodes and the local transfer

functions. The solution to a problem is then obtained by computing a fixed point (least or greatest) of the equation system, where a lattice value is associated with each node of the flow graph. An interesting treatment of the monotone framework for concurrent systems is given in [48].

The only application of the data flow paradigm to the analysis of the π-calculus has been given in [81], where the issue of causality among processes and true concurrency is examined. However, as we are unaware of any security-related data flow analyses, we shall not discuss this paradigm any further.

1.4.2 Control Flow Analysis

A control flow analysis is concerned with answering the following question: Given a particular point in a program, what is the set of subprograms, functions, commands etc. that can be reached from that point? In other words, a control flow analysis attempts to record information about the different execution paths in the program. This information is then used to conduct program optimisation and transformation as well as determine runtime properties. Despite the fact that the early control flow analysis techniques were developed mainly for the functional programming paradigm, control flow analyses were later utilised in other paradigms as well, like the object-oriented, concurrent and logic programming paradigms.

Often, a control flow analysis is expressed as a constraint-based analysis and its solution relates the different points of control flow of the basic program blocks. Taking a functional program as an example, the set of constraints may be classified into three main classes, assuming a finite set of program labels. The first two classes relate function abstractions and variable values to their corresponding labels, respectively. The third class expresses which formal parameters of functions are bound to which actual parameters in function applications and whether the results returned from functions can be returned by the applications in which they appear.

For example, consider the following functional program [102]:

$$[[\texttt{fn x => [x]}^{1}]^{2}\ [\texttt{fn y => [y]}^{3}]^{4}]^{5}$$

If we assume functions $C(l)$ and $label(x)$, where the former indicates the set of expressions that l may evaluate to, and the latter the values that variable x may be bound to, then the following classes of constraints may be stated for the above program. The first class relates

values of function abstractions to their labels:

$\{\texttt{fn x => [x]}^1\} \subseteq C(2)$

$\{\texttt{fn y => [y]}^3\} \subseteq C(4)$

The second class of constraints relates values of variables to their labels:

$\texttt{label}(x) \subseteq C(1)$

$\texttt{label}(y) \subseteq C(3)$

The final class expresses information about function applications, where the following conditional constraints are introduced (the inclusion of functions `fn x => x` and `fn y => y` provides for the possibility that both functions may be applied):

$\{\texttt{fn x => [x]}^1\} \subseteq C(2) \quad \Rightarrow \quad C(4) \subseteq \texttt{label(x)}$

$\{\texttt{fn x => [x]}^1\} \subseteq C(2) \quad \Rightarrow \quad C(1) \subseteq C(5)$

$\{\texttt{fn y => [y]}^3\} \subseteq C(2) \quad \Rightarrow \quad C(4) \subseteq \texttt{label(y)}$

$\{\texttt{fn y => [y]}^3\} \subseteq C(2) \quad \Rightarrow \quad C(3) \subseteq C(5)$

The least solution to the above equations is given as follows:

$$\begin{aligned}
C(1) &= \{\texttt{fn y => [y]}^3\} \\
C(2) &= \{\texttt{fn x => [x]}^1\} \\
C(3) &= \{\ \} \\
C(4) &= \{\texttt{fn y => [y]}^3\} \\
C(5) &= \{\texttt{fn y => [y]}^3\} \\
\texttt{label(x)} &= \{\texttt{fn y => [y]}^3\} \\
\texttt{label(y)} &= \{\ \}
\end{aligned}$$

This solution reveals that the function, (`fn y => y`), is never applied in the program (since we have that `label(y)`$=\{\}$), and the program may only evaluate to the function, `fn y => y`, (since we have that $C(5) = \{\texttt{fn y => [y]}^3\}$). Hence, the last two conditional constraints have false left sides leading to false right sides.

To achieve a more precise control flow analysis, the concept of k-CFA analysis (CFA stands for *C*ontrol *F*low *A*nalysis) was developed in [119], where k stands for the level of context information taken into account in the analysis. Hence, a 0-CFA denotes a context-insensitive or *monovariant* analysis. On the other hand, when $k > 0$, the analysis is described as context-sensitive or *polyvariant*. The presence of dynamic context information allows for

the different instances of variables and program blocks to be distinguished and therefore, arrive at more precise results for the analysis. Several variations of the k-CFA analysis exist, for example the uniform k-CFA, polynomial k-CFA and the Cartesian product algorithm.

Control flow analysis has been combined with other approaches, like data flow analysis, abstract interpretation and flow graphs, to achieve better quality, understanding and presentation of the final results. Data flow information may be included in the final set of results obtained from the control flow analysis, which then results in better quality of control flow information. The use of abstract interpretation techniques is necessary for tuning the complexity of the analysis versus its precision. As with the data flow analysis, flow graphs are a handy tool to visualise results and gain better understanding of the flow of control among different program blocks.

1.4.3 Abstract Interpretation

An abstract interpretation allows for programs to be analysed by running their specifications over finite approximated semantic domains that are less precise than the concrete semantic domains, but that are characterised as being safe computable abstractions of the concrete domains. In fact, the original work by [45] has developed from being a specific framework for imperative languages to a general framework offering solutions on the design of static analyses for different programming paradigms. It even became closely linked with other approaches like data flow analysis, control flow analysis and type systems.

The first step in designing an abstract interpretation is to determine whether the standard semantics of the language is sufficiently rich to be able to capture the property under consideration. If not, a non-standard semantics is designed as an extension or modification of the standard semantics to capture the property of interest. The resulting non-standard semantics is sometimes proven to be *correct* with respect to the standard semantics. Different approaches exist for proving the existence of a correctness relation. The approach we follow in our framework is to show that for all the non-standard semantics elements, the standard semantic component can be extracted from these elements.

More formally, assume P is a program, $(\!|P|\!) = v \in V$ is the standard semantic evaluation, and $[\![P]\!] = l \in L$ is the non-standard semantic evaluation, then the correctness relation R is formalised as a compositional function:

$$\begin{aligned} \forall P \quad &: \quad R([\![P]\!]) = (\!|P|\!) \\ \forall v_1, v_2 \in V, l_1, l_2 \in L \quad &: \quad R(l_1) = v_1 \ \wedge \ R(l_2) = v_2 \ \Rightarrow \ R(l_1 \star l_2) = v_1 \star v_2 \end{aligned}$$

Where, $\star$, is a composition operation that constructs the complex meaning, $l_1 \star l_2$, from

the primitive meanings l_1 and l_2. Often proving the existence of R is largely dependent on the amount of information about the standard semantics retained in the non-standard semantics.

Since the concrete semantics (standard or non-standard) may operate over infinite domains, the computation of this semantics is not guaranteed to terminate, even with the use of least fixed points. Therefore, a suitable approximation (abstraction) is required to keep the semantic domain finite. This abstraction is shown to be safe with respect to the concrete semantics by proving that a safety relation is preserved across the abstract semantic values. This relation expresses the fact that every concrete computation maps to a corresponding abstract computation, although the latter of course, is less precise.

Hence, if we assume that, $[\![P]\!]^\sharp = l^\sharp \in L^\sharp$, is the abstract semantics function, then the safety requirement can be formulated as a relation, S, defined as follows:

$$\begin{aligned} &\forall P, [\![P]\!] = l, [\![P]\!]^\sharp = l^\sharp \quad : \quad (l, l^\sharp) \in S \\ &\forall l_1, l_2 \in L, l_1^\sharp, l_2^\sharp \in L^\sharp \quad : \quad (l_1, l_1^\sharp) \ \wedge \ (l_2, l_2^\sharp) \in S \ \Rightarrow \ (l_1 \star l_2, l_1^\sharp \star l_2^\sharp) \in S \end{aligned}$$

The definition of the safety relation S is highly dependent on the semantic domains and the choice of the abstraction adopted.

In the theory of abstract interpretation, the set of abstract semantics values, $L^\sharp$, is interesting because it is augmented with some ordering relation, $\sqsubseteq$, which results in $L^\sharp$ having some structure, like a complete lattice. Moreover, we can impose the following implications between the complete lattice, $L^\sharp$, and the safety relation, S:

$$\begin{aligned} \forall l \in L, l_1^\sharp, l_2^\sharp \in L^\sharp : (l, l_1^\sharp) \in S \ \wedge \ l_1^\sharp \sqsubseteq l_2^\sharp \quad &\Rightarrow \quad (l, l_2^\sharp) \in S \\ \forall l \in L, l^\sharp \in L'^\sharp, L'^\sharp \subseteq L^\sharp : (l, l^\sharp) \in S \quad &\Rightarrow \quad (l, \bigsqcap L'^\sharp) \in S \end{aligned}$$

The first implication states that if an abstract value, $l_1^\sharp$, is safe, then a larger value, $l_2^\sharp$, will also be safe and, therefore, the smaller value, $l_1^\sharp$, would constitute a more precise or better solution. The second implication states that for a set of safe abstract values, their greatest lower bound is also a safe value. In other words, for a concrete semantic value, l, there is always a smallest abstract value, $\bigsqcap L'^\sharp$, among a sub-lattice, $L'^\sharp \subseteq L^\sharp$, of safe values, that is itself safe with respect to l. Practically, this has the effect that an abstract interpretation needs to be performed only once for a program to obtain the best solution.

An alternative approach to the safety proof involves the introduction of a pair of abstraction/concretisation relations that are shown to form a *Galois connection* between the concrete and abstract domains. Hence, a tuple, $(L, \alpha, \gamma, L^\sharp)$, is a Galois connection between the two complete lattices, $(L, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$, and, $(L^\sharp, \sqsubseteq, \sqcap, \sqcup, \bot, \top)$, where $\alpha : L \to L^\sharp$ and

$\gamma : L^\sharp \to L$ are monotone functions, if the tuple satisfies the requirement that:

$$\begin{aligned} \gamma \circ \alpha &\sqsupseteq \lambda l.l \\ \alpha \circ \gamma &\sqsubseteq \lambda m.m \end{aligned}$$

Which express the safety of the abstract semantics at the cost of losing precision. On certain occasions, *widening* and *narrowing* techniques may also be used in combination with Galois connections to further approximate fixed points.

The main application of abstract analysers has been in combination with language compilers that could be used in program optimisation and to prove that the program is safe with respect to a certain security policy. Examples of popular non-security-related implementations of abstract interpretation-based analyses include the strictness analysis, sharing analysis, the ground substitutions analysis in logic programs and approximations of n-dimensional vector spaces over integers and rational numbers.

1.4.4 Type Systems

Type systems are widely used in programming languages to avoid unwanted behaviour of their programs during runtime. Some of this behaviour could be crucial to security violations, like private information leaks and restricted address space accesses. In general, a *type* is regarded as holding information that is true about the program entity it types. Moreover, if one thinks of a type as being a collection of values, then a subset of that collection constitutes a *subtype*. Subtyping expresses an ordering relation among the different types. It also may be thought of as giving more refined information about the subtyped entity.

One may also define a *principal type* as the most general type of an expression. For example, a lambda abstraction, $\lambda x.x$, has the principal type, $a \to a$, where a could be instantiated with any type. Principal types of language expressions are often computed from the types of their subexpressions using Robinson's unification algorithms [113].

A *typing environment* is used to map the different program entities (statements, expressions, constants, variables etc.) to their types. Such environment may be constructed manually according to a set of axioms and rules. Consider the following imperative language:

$$\begin{aligned} S, S_1, S_2 \quad := \quad & [x := a]^l \\ | \quad & [\texttt{skip}]^l \\ | \quad & S_1; S_2 \\ | \quad & \texttt{if } [b]^l \texttt{ then } S_1 \texttt{ else } S_2 \\ | \quad & \texttt{while } [b]^l \texttt{ do } S \end{aligned}$$

Where $l, l' \in \{1, 2, 3, 4\}$. Then each statement may be considered as typed by the type $\Sigma \to \Sigma$, where, Σ, is the type of the state of the program. This type could denote the result of a reaching definitions analysis, RD. Based on this interpretation of Σ, one may introduce the axioms and rules of Figure 1.3 [102].

$$
\begin{array}{ll}
(assign) & [x := a]^l : RD \to ((RD \backslash \{(x, l')\}) \cup \{(x, l)\}) \\
(skip) & [\texttt{skip}]^l : RD \to RD \\
(seq) & \dfrac{S_1 : RD_1 \to RD_2 \quad S_2 : RD_2 \to RD_3}{S_1; S_2 : RD_1 \to RD_3} \\
(if) & \dfrac{S_1 : RD_1 \to RD_2 \quad S_2 : RD_1 \to RD_2}{\texttt{if } [b]^l \texttt{ then } S_1 \texttt{ else } S_2 : RD_1 \to RD_2} \\
(while) & \dfrac{S : RD \to RD}{\texttt{while } [b]^l \texttt{ do } S : RD \to RD} \\
(sub) & \dfrac{S : RD_2 \to RD_3}{S : RD_1 \to RD_4} \quad \text{if } RD_1 \subseteq RD_2 \text{ and } RD_3 \subseteq RD_4
\end{array}
$$

Figure 1.3: Types for the reaching definitions analysis.

These axioms and rules are explained as follows. Axiom $(assign)$ states that any previous assignments carried out on the variable, x, are removed from the final state RD, and x is declared as having the value assigned at current statement marked l. Axiom $(skip)$ does not alter RD and rule seq composes the type of a sequential statement, $S_1; S_2$, from the types of the composed statements, S_1 and S_2. Rule (if) deals with the conditional statement, which is given the type of either branch. Rule $(while)$ assigns to a `while` statement the type of its body. Finally, rule (sub) deals with the subtyping of statements.

Given the factorial programme of Section 1.4.1:

$$F \stackrel{\texttt{def}}{=} [fac := 1]^1; \texttt{ while } [n > 1]^2 \texttt{ do } ([fac := fac * n]^3; [n := n - 1]^4)$$

It is possible to apply the axioms and rules of Figure 1.3 to arrive at the final type for F as follows:

$$F : \{(n, ?), (fac, ?)\} \to \{(n, 4), (fac, 3)\}$$

Which confirms to the results of Figure 1.2, Section 1.4.1, which shows that $Rin(1) = (n, ?), (fac, ?)$ and $Rout(4) = (n, 4), (fac, 3)$.

Alternatively, a *type inference algorithm* may be constructed to infer the types of language entities from the types of their surrounding context. The set of typing axioms and

rules can then be used to make *judgements* about whether a particular program construct is well typed within the typing environment or not. Unification in the algorithm may also be used to generate principal types. Whenever all the programs of a language are shown to be well typed (by proving a type soundness theorem stating that well-typed programs are well behaved), the language is then said to be *type sound*.

Using *static type checking*, the layout of types in a program are tested against the typing rules to reveal any violations. Programs that do not enforce the typing rules and that contain *trapped errors* (errors detected by the typing system) are eliminated. Again, there is an element of approximation in the manner that static type checking works: if a program passes the type-checker, it is surely well typed. Otherwise, it is ill typed or cannot not guaranteed to be well typed. Since defining the typing system is a separate problem from constructing a type-checking algorithm, it is often the case that type system designers run into the problem of defining type systems that will only admit infeasible algorithms or sometimes, no algorithms at all.

Most type systems found in procedural languages are *first order*, i.e. they lack type parameterisation and abstraction. Whenever these are present, the system is called *second order*. Type parameterisation refers to those programs that are of the type $\lambda X.M$, where M is the program and X is a type variable. This means that M is parameterised with respect to X, which is instantiated only by the context. Parameterisation is a feature often found in polymorphic languages, like ML. *Type abstraction* is a feature that often appears as opaque types in interfaces in modular languages, like CLU [86].

Type systems can also use special annotations, called *effects*, that express the effect of using each of the types defined by the typing environment. For example, a function type of the form $\tau_1 \xrightarrow{\varphi} \tau_2$ maps values of type τ_1 to values of type τ_2 and as a result of this application, it may perform a call to function φ. Effect systems are often implemented as extensions of type inference algorithms and are useful in providing information about the internal steps of computation for each program expression.

Finally, one of the interesting topics in the subject of type systems is that of *type equivalence*. Type equivalence relates separately written type expressions. *By-name* equivalence refers to those types that match by having the same names. On the other hand, *structural* equivalence refers to those types that match by having the same structure. It is usual to find typing systems that adopt a mixture of both equivalence relations.

1.5 Security Properties

As we highlighted earlier, the task of protecting private and critical information from unauthorised actions conducted by malicious intruders or resulting from innocent errors has grown into an area of major importance in the design and analysis of computing systems. In the following paragraphs, we attempt to cover briefly some of the properties often dealt with when considering the security of computing systems and that are of interest to our framework. These include the secrecy and authenticity of information.

1.5.1 Secrecy

The problem of protecting the secrecy of private information is a long-standing problem that finds its roots in the pre-computing age. Models were devised to prevent unauthorized subjects accessing classified physical resources (like paperwork) owned by other subjects. The success of these models in the real world encouraged researchers to adapt the same ideas to protect the secrecy of information in the computing world.

The early computer security models were *access control* models that regulated the access of users and programs to sensitive data. These models could be classified into two main categories: *mandatory access control* (MAC) and *discretionary access control* (DAC) models. The fundamental difference between the two categories is that the former grants a system-wide policy the right to decree which subjects have access to which resources, whereas in the latter, the owner of each particular resource decrees itself which subjects can have access to the resource. The main example of MAC models is the Bell and La Padula [19] model, which is based on rigid military security policies and can be summarized by the no read-up/no write-down properties that prohibit low-level (unclassified) subjects obtaining high-level data. A variation of the Bell and La Padula model known as the Biba model [20] preserves the integrity of information by reversing the capabilities of subjects. Hence, low-level (unclassified) subjects cannot alter the contents of high-level resources. This property is known as the no read-down/no write-up property.

The rigidity of MAC models is relaxed in DAC models. Every owner of a resource is responsible for granting rights to access and modify that resource to other subjects. These rights may change with time and depending on the appearance of new entities and the disappearance of old ones. Hence, in addition to the system-wide policy that assigns subjects to groups, a local policy exists for each subject that specifies the access rights to its resources. An example of the implementation of DAC models is the permissions list of Unix files.

DAC models, in general, suffer from the problem of malicious programs that run on behalf of a high-level subject and that may modify the access rights of resources belonging to that subject granting access to low-level subjects[4]. This problem is not present in MAC models, as any writing-down or reading-up between high-level and low-level subjects is prohibited. However, the MAC models can only monitor *explicit* flows of information and fail to capture the *implicit* flows.

Implicit flows were first referred to by Butler Lampson in his famous note on *covert channels* [85], in which he raised new questions on how private information could be leaked in computing systems. The note examined the task of confining a program and revealed the real boundaries that this problem could have. The transmission of data may occur not only as a result of sending that data over explicit communication channels, but also over media not intended for communications in the first place. Well-known examples of covert channels are the use of computer storage space to transmit data by filling or emptying the space and the revealing of sensitive information about the guards of conditional and while-loop statements within the bodies of those statements. Moreover, the solution to completely avoid such covert channels in system design, as suggested in [85, page 3], is rather restrictive.

As a result of these subtleties in the way information is passed around, new security models appeared that aimed at capturing the whole flow of information in a system, rather than just modelling the access of subjects to resources. A well-known example of information flow models is the lattice model proposed by Denning [46], which covers both explicit (e.g. assignments) and implicit (e.g. conditionals) flows. The security levels of subjects are arranged in a lattice whose ordering is determined by the system security policy. Information then may flow only from the low-level subjects to the high-level ones.

Other approaches have also been adopted to represent information flows in the presence of covert channels. These include *non-deducibility* [124] and *non-interference* [67]. There are two main problems associated with non-deducibility, as McLean explains [90]: First, it weaker than non-interference, which is problematic for cryptographic-based systems. Also, it may block the flow of information from low-level to high-level entities in particular systems. Second, it is non-compositional, which results in the inability to reason about the secure composition of insecure components.

Non-interference has also been a hot topic in recent years and one may state that there is no general consensus as to what constitutes the definition of non-interference. In their original work [67], Goguen and Messeguer presented a model of non-interference for deterministic systems, i.e. systems whose output is solely determined by their input. The model

[4]Such programs are often called Trojan horses.

is based on the observation that a system is *non-interfering* whenever the output of that system does not differ with respect to variations in its input. Otherwise, the system is *interfering*. The difference in system behaviour is expressed in terms of purging functions, i.e. functions that remove parts of traces of input commands from the resulting output. A non-interfering system identifies a purged output with a non-purged output.

Another definition of non-interference was given by the core calculus of dependency [2], which is an extension of Moggi's computational λ-calculus [98]. The dependency calculus is a general framework for the comparison of the different type-based dependency analyses, like program slicing, call-tracking, binding-time analysis and secure information flow analysis. Notions of *strong* and *weak* non-interference are formalised by a denotational model for the calculus, where the strong version requires the termination of the system and the weak version does not (thereby allowing high inputs to affect the termination of the system without affecting the non-interference result). The different analyses are then given as instances of the calculus satisfying either the strong or the weak versions.

Finally, it is worth mentioning the work achieved so far by Focardi and his team [55, 62, 61, 58, 59, 56, 60, 57] in laying down the basis for a unifying theory of dependency that uses different variations of CCS in reasoning about most of the security properties, like the secrecy, authenticity, integrity and availability properties.

Other more recent definitions of the secrecy property build on some theory of process equivalence. For example, in the spi calculus [5], secrecy is defined in terms of testing equivalence between two instances of a system instantiated with two different messages. An intruder who is observing these instances is unable to differentiate between them.

As we mentioned earlier, the introduction of computer networking meant that intruders could now undermine the security of information while being transmitted from one computer to another. The solution to communication security was found in cryptography. While models of access and information flow control are primarily concerned with protecting sensitive data from unauthorised subjects, cryptography offers a different vision: security by obscurity. Sensitive data can be sent over public communication channels as long as they are encrypted, which means that it is of little use to intruders. Discussing the fundamentals of cryptography is outside the scope of this section. We recommend [91, 116] as sourcebooks for those seeking background on this topic.

In our definition of the secrecy property, we shall adopt a form of explicit information flow model that monitors the movement of data among the different processes in a system. A low-level process obtaining data created by another high-level process results in an instance of the process leakage threat. The explicit meaning of the word "obtaining" here is strictly defined

in terms of the message-passing communications in mobile systems, and in terms of message passing and message processing (e.g. decryption) in cryptographic systems. Issues related to the more complex notions of non-interference and equivalence-based secrecy are left as future work that builds on the current static analysis framework, and recommendations are made in the concluding chapter on future work. This choice of the definition of secrecy as the main working definition for our static analysis framework is in line with the current state of security models. Most of the current implementation systems rely on some form of access or information flow controls. Non-interference and process-equivalence are still immature areas largely confined to the research community and have not yet been proven practically successful, in particular, for the case of non-deterministic systems.

1.5.2 Authenticity

The authenticity property is generally defined as the assurance about the identity of the origin of a particular entity (agent, datum, message etc.). This identification may entail some *trust* in the sense that identifying a trustable entity could initiate different behaviour than in the event of identifying a less-trustable entity. For example, when performing online booking for air flight tickets, we would like to make sure that the booking site, to which the credit card details are submitted, has a valid digital certificate signed by a trusted third party (or a Certification Authority). This will increase the confidence about sending credit card details to the site. However, if no valid certificates were produced, then the site will be less trustworthy of submitting sensitive information.

The problem of authentication when dealing with isolated computing systems becomes a problem of authorization and the protection of the integrity of data from malicious or erroneous behaviour. Therefore, mechanisms like user login, token-based authentication and biometrics authorization become sufficient to protect static information from being modified by unauthorized intruders. Such modifications could be viewed as tampering with the origin of the stored data and, therefore, undermine the authenticity property of that data.

The presence of communications among systems introduces the necessity for authentication protocols since the communicating entities may often be unaware of each other's identities beforehand, thereby undermining techniques like user logins, which require some prior knowledge about the entities involved. The purpose of an authentication protocol then is to provide a well-defined sequence of messages *at the end* of which the two entities participating in the protocol will be assured of each other's identities. This may take place with the aid of a trusted authentication server. Protocols that authenticate both partici-

pating entities are called bilateral protocols. Unilateral protocols, on the other hand, refer to protocols that authenticate either of the two participating entities, but not both.

Most of the existing authentication protocols make use of either secret-key or public-key cryptography. Usually, in secret-key protocols, the presence of a trusted server is required to hold the long-term keys shared with each of the participating agents, while session keys are created for the duration of authentication sessions. Examples of secret-key systems include Otway-Rees [104] and Kerberos [92]. In public-key protocols, a Public-Key Infrastructure (PKI) is usually implemented to manage key distribution issues, especially those related to name-to-key bindings where *certificates* are used to validate or invalidate such bindings (see [63] on the topic of PKI). The presence of a server may not be required beyond the initial stage of creating the key pairs. The protocol could then involve creating session keys for each authentication session. Examples of public-key bilateral protocols include the Needham-Schroeder public-key protocol [100], whereas the Woo-Lam protocol [133] is an example of unilateral protocols. In either the secret-key or public-key protocols, fresh information must be provided in each session to prevent replay attacks.

Recently, other authentication models have been suggested. In the spi calculus [5], for example, authenticity is formalised in terms of a testing equivalence relation. The authenticity of messages is preserved if an instance of a system instantiated with that message is testing equivalent to the system's specification. Other models include the use of proof-carrying code in [13, 18] as a general framework for the design and implementation of authentication and authorization services using high-order logics. Also, in [59], the concept of non-interfering systems is used as the basis for defining the different security properties, including the authenticity of messages. However, it is still to be seen how practical and feasible such models could turn out to be in real world situations.

The model of authentication that we adopt in our framework is based on a web of trust; systems are classified as running at different trust levels. Breaches will occur whenever highly trusted processes obtain data originating at processes with lower levels of trust. Similar to the notion of secrecy, the word "obtain" here has different meaning depending on the presence or absence of cryptographic operations in mobile systems. Generic mobile systems will obtain the untrusted information as a result of the value-passing behaviour, which denotes communications among processes. In cryptographic systems, both value-passing and value-processing behaviours may result in authenticity breaches.

1.6 Further Reading

The main areas underlying the background to the research presented in this thesis were reviewed in this chapter. These areas are classified into four headings: Mobile systems, denotational semantics, static program analysis and computer security.

Recommended readings in the area of formalisms for mobile computing include [96], which is an introductory book on the subject of the π-calculus, and [114], which provides a comprehensive treatment of the theory underlying the π-calculus. The spi calculus is explained in detail in [6]. The main book on the subject of static program analysis is [102], which provides a chapter for each of the main approaches; data flow analysis, control flow analysis, abstract interpretation and type systems.

A recommended reading on the subject of denotational semantics is [122], which discusses the denotational methodology in the definition of programming languages semantics as introduced by Scott and Strachey [123, 117, 118]. A gentler introduction on denotational semantics (and other semantics) is given in [131]. The topic of computer security is wide and many references exist. However, one may suggest here [68] for a reading on general computer security, [63] on digital signatures, [35, 42] on authentication protocols, [116, 91] on general cryptography and [103] on electronic payment systems.

1.7 Outline of Our Approach

As we explained in the previous sections, the problem of computer security is a sophisticated problem that encompasses a wide range of theories and technologies. In this thesis, we tackle the problem of computer security from perspective of the static analysis of computing systems specified formally using nominal calculi. Hence, we attempt to establish static-analysis framework based on abstract interpretations for the analysis of mobile systems specified in the π-calculus and cryptographic systems specified in the spi calculus.

The framework relies on a denotational semantic model and is aimed at capturing one aspect of the behaviour of processes in the π-calculus and the spi calculus that has security significance, i.e. term substitution. This property then can determine which processes can obtain which terms (information), and based on this result, one can define secrecy and authenticity properties as the capturing of names created by high-level or low-level processes, respectively. Crucial to the termination of the analysis is the selection of a suitable abstraction in order to keep finite the semantic domains within which processes run. This abstraction has to be shown safe with respect to the concrete semantics.

The structure of the rest of the thesis is organised as follows:

- Chapter 2: We survey state of the art work related to the area of the analysis of mobile and cryptographic systems as well as review some denotational models that have been proposed for the π-calculus.

- Chapter 3: We introduce the languages of the π-calculus and the spi calculus and review their syntax and structural operational semantics. We define standard denotational semantics for the languages that model the operational behaviour of processes. The denotational semantics is shown to be sound and adequate with respect to the structural operational semantics.

- Chapter 4: We introduce a non-standard semantics that is a correct modification of the standard denotational semantics and we design an abstraction that keeps the semantic domains finite to ensure the termination of the analysis. We prove the safety of the abstract semantics with respect to the non-standard semantics. Finally, we apply the abstract interpretation to a number of examples including a file transfer protocol and a number of authentication protocols, like the Needham-Schroeder public-key, SPLICE/AS, Otway-Rees, Kerberos, Yahalom and Woo-Lam protocols.

- Chapter 5: We utilise the results of the abstract interpretation established in Chapter 4 to define the secrecy and authenticity properties. We apply these definitions to the results of the abstract interpretation of the example systems analysed in Chapter 4.

- Chapter 6: We review two implementation prototypes of the static analysis framework for cases of mobile systems and cryptographic protocols. These are called the Picasso-A Pi-Calculus Analyser for Secrecy and Security Objectives, and Spicasso - A Spi-Calculus Analyser for Secrecy and Security Objectives tools.

- Chapter 7: Finally, we conclude the thesis and discuss its major contributions to the area of the static analysis of computing systems for the detection of security threats. We also give directions for prospective work that could spark from the work achieved so far and presented in this thesis.

Chapter 2

Related Work

2.1 Introduction

Covering research undertaken in all the areas related to the scope of the framework of this thesis and reviewed in the previous chapter demands an enormous effort, which would merit its own body of work and thesis. Therefore, we shall concentrate on two main areas: static analysis techniques with applications in the area of security of mobile and cryptographic systems and denotational semantic models for nominal calculi. Furthermore, we narrow these areas to include only research that specifically deals with the language of the π-calculus and its cryptographic extension, the spi calculus.

2.2 Static Analysis Techniques for Program Security

In recent years, the use of static program analysis in analysing mobile and cryptographic systems and verifying their security properties has grown into a major area of research, where a variety of techniques have been proposed and implemented to varying degrees of success. In this section, we review research carried out in the areas of control flow analysis, abstract interpretation, type systems and other logic-based areas.

2.2.1 Control Flow Analysis

This approach has been adopted on a number of occasions by [24, 25, 27, 26], in which the flow logics approach [101] is used to predict certain security properties pertaining to mobile systems, like having the confinement of private information and the adherence to the Bell and La Padula properties [19].

The main analysis focuses on the usage of channels and the values sent over them in the π-calculus. In particular, a fresh name is associated with a superset of the set of names that can be communicated over that name. Also, an input parameter is associated with a superset of the set of names that may substitute it at runtime.

One major drawback with the manner by which the analysis is carried out is that it identifies all the copies of a fresh name arising from the restriction operator. This implies that non-uniform properties of names that change between the different runs of systems (as explained in Chapter 4) cannot be expressed straightforwardly. The analysis also suffers from the inability to detect certain deadlocks arising from situations where the channel of communication is a fresh name with restricted scope.

The separation between the problem of validating a given solution and the construction of such a solution as formulated by the flow logics approach, meant that the resulting analyses of [24, 25] are targeted towards the validation of a proposed solution rather than offering a constructive algorithm. However, a minimum solution for the validating analysis is proven to always exist, and in [27], a constructive algorithm for this minimum solution is also supplied assuming a finite set of names.

The security properties dealt with in [24, 25] are all based on the main control flow analysis. In [24], the confinement problem of private names is the main property. In [25], the no read-up/no write-down property introduced by Bell and La Padula [19] is investigated. This property is often regarded in the security literature as a rigid form of security.

Finally, [26] provides a control flow analysis for a variant of the spi calculus with history-dependent cryptography using the idea of confounders in encrypted messages. This facilitates the comparison of ciphertexts since, for example, $\{0, (\nu\, r)\}_k \neq \{1, (\nu\, r)\}_k$. The paper establishes two main results. First, it shows that the Dolev-Yao definition of secrecy [47] is expressible in terms of a control flow analysis. No specification of the Dolev-Yao most general attacker is provided; instead, it is computed using an approximation relation. The second result establishes that the analysis can capture the form of non-interference stated in [1]. This result elegantly separates between the issues of confidentiality and non-interference using the control flow analysis approach.

2.2.2 Abstract Interpretation

This approach has been applied effectively to the analysis of mobile systems. Notable works include [126, 127, 99, 51, 52, 53]. In [126], a sound non-uniform description of how topologies evolve in closed systems that do not contain nested replications is presented. The approach is

based on the abstract data structures of [82], where the abstract meaning of a process in the π-calculus is represented as an undirected *hypergraph* (a hypergraph is a graph where an edge can connect more than two vertices) signifying the sequence of internal computations that evolve within a process from its initial specification. The analysis was further developed and greatly simplified in [127], where a refined semantics is introduced to capture the instance of a channel that establishes a link between two processes.

The study of system interactions in [127] has inspired the analysis of [51], where the main theme is detecting the leakage of confidential information in the presence of unknown contexts. The analysis is non-uniform and can distinguish between the different instances of a process. The open semantics takes into account the lack of specification of the intruder processes and unlike [126, 127], the analysis is not restricted to systems with no nested replications. Encoding the ability to deal with unknown specifications into the semantics is a different alternative to our approach, which adopts a specification of the intruder (i.e., the Dolev-Yao most general attacker).

The analysis of [51] was extended in [52] in an occurrence counting analysis for detecting the exhaustion of resources, mutual exclusion and deadlock properties. On the other hand, a more generic parametric framework has been proposed in [53], which is capable of expressing the equality and inequality relations between names, i.e. the dependency among names at their creation point. The abstract interpretation approach has been used by [99] to approximate cryptographic protocols specified in a cryptographic language with finite principals and sessions (the spi calculus allows infinite principals and sessions) using tree automata. An implementation of the analysis exists. However, it has been applied to small protocols with finite runs only (mostly single runs).

2.2.3 Type Systems

Type systems have been extensively researched within the mobile systems and security community and notable examples include the works of [1, 72, 112, 37, 73, 77, 74], where properties related to information privacy, resource access control and trust have been tackled.

In [72], a typing system is suggested for a distributed variant of the π-calculus called Dπ, where processes reside at named locations and communications occur locally. In order to be able to express remote communications, a process movement primitive is introduced. The typing system introduces the notion of *location types*, where each location is typed by the resources available to agents residing at that location. This will control the set of capabilities that an agent can perform at a certain location. For example, the agent may be able to send

but not to receive data. Runtime errors are then used to express the breach of type safety. These errors can detect illegal movements of processes as well as illegal communications.

In [112], dynamic type checking is used to deal with malicious unspecified agents moving between different sites. Open secure semantics are given for the language of Dπ, where special *bad* types are used to distinguish untyped code in what is known as a partial typing system, where only a subset of agents is well-typed. Filters are used by sites to judge the trustworthiness of an incoming agent. Based on this judgement, a notion of authority from which trust between the different sites harbouring processes can be built. Hence, the need for dynamic type checking is, in fact, reduced. The correctness of the behaviour of well-typed processes is shown to be safe with respect to well-behaving sites.

A more extensive treatment of the subject of trust in mobile systems modelled by the π-calculus is given in [74]. Here, a system of Boolean annotations is used to guarantee that only trusted data are used in trusted contexts. The system relies on a notion of trust during run time. An algorithm for creating the general types of the system is also suggested.

In [73], two security properties, the resource access control and information flow properties, are dealt with in a typing system for a variant of the asynchronous π-calculus [31, 76] with security levels. Using a security policy that assigns levels to processes, a process can only access resources that were created by processes running at the same or lower levels. Similarly, a process can only write to resources created by higher-level processes. On the other hand, implicit flow of information is dealt with by a notion of non-interference, which depends on a form of may-testing.

An advanced static type checking system is used in [77] to guarantee secure information flow for general process behaviour in the polyadic π-calculus extended with extra syntactic constructs. In this typing system, non-linear types are used as part of a finite partially ordered set of *action types*. The action type of a process is an abstraction of the causal dependency among the free channels of that process. With the use of secrecy indices, a subject reduction property of the resulting typed processes is given. This implies that composing a typed process with external processes does not change its internal secrecy. The subject reduction property is then used to express a form of non-interference among low-level and high-level processes. The resulting system is also used to embed the system of [130], using typed process representation.

One of the major drawbacks of type systems is their inability to express non-uniform properties due to the fact that they associate the same type with the different runs of the system. In [37], this problem was remedied by introducing a primitive for fresh type creation. Special channel types called *Groups* can be created, where names of a type belonging to a

particular group cannot be communicated over channels of another group type (possibly with a lower secrecy level). This will protect against the leakage of secret channels from one group to another.

It seems, to date, that the work of [1] has assumed an authoritative role in the subject of security and types in the spi calculus. This is simply due to the wealth of basic concepts that the work contains, and in particular, the idea of mixing (primitively) typed and untyped data in the typing system and the idea that every participant in a security protocol must be typable. An overall view of secrecy principles is also given in the context of the spi calculus. The proposed typing system guarantees the protection of secret data against leakage through a notion of non-interference that is based on testing equivalence. However, the system does not clarify the issue of principal types and therefore, no complete and sound typing algorithm is provided.

Non-interfering security has also been treated in the area of type systems, although to a lesser extent. The work of Smith and Volpano [130] presents a type system for a sequential imperative language that can deal with explicit, as well as implicit information flow. Insecure explicit flow is detected as a flow from high-level to low-level variables. Implicit information flow, on the other hand, occurring within conditional and while-loop statements is formalised by a type soundness requirement that guarantees that the command levels of the branches of conditional statements and while-loop bodies be as high as the level of the guards of those statements. However, the type system fails in situations where programs do not terminate successfully as a result of assuming that the sequential composition of typable commands is also typable. This problem was later remedied in [120], and a type system for a multi-threaded imperative language was suggested. However, the resulting solution is, unfortunately, too restrictive; the guard of a while-loop is always limited to a low level of security and the body of the while-loop itself is classified as a low-level command. In general, both works [130, 120] adopt a notion of non-interference, which is closely related to the original view given in [67]: the final values of the low-level variables should be independent of the final values of the high-level variables.

A more refined type system for non-interference within concurrent languages is proposed in [31, 32]. The main work extends the system proposed in [120]. However, the attitude towards the concurrent language semantics is that of small step as compared to the big step semantics adopted by [120]. The implication of this is that intermediate values of low-level variables are also considered (not only the final values). This further results in a stronger notion of non-interference that relieves restriction on the levels of guards and bodies of conditional and while-loop statements. The main difference between [31] and [32] lies in

dealing with scheduling mechanisms of concurrent subprocesses (threads).

A simple treatment of non-interference is also given by [110] for an extension of the π-calculus, which can express the independent execution of parallel processes. The definition of non-interference adopted in [110] is based on weak barbed reduction congruence [114]. The semantics of the language annotates processes with security levels drawn from a complete lattice. The final non-interference result then states that pruning high-level sub-terms does not affect the low-level sub-terms up to the weak barbed congruence.

Another extension of the asynchronous π-calculus, termed the security π-calculus is suggested by [70] that incorporates a type system where security levels and read/write capabilities are associated with input parameters and restricted names. May and must testing equivalences are then formalised for the language and based on these equivalence formalisations, notions of non-interference are enforced. The work is a comprehensive extension of the earlier work presented in [73].

Finally, the *compositional security checker* (CoSeC) developed in [56] is a static analysis tool that builds on the *Concurrency Workbench* [43]. Both tools are forms of semantics-based model checkers that can check for properties of processes based on different equivalences and preordering. The language of input for CoSeC is an extension of CCS with separately marked high-level and low-level observable actions. The treatment of the tool benefits only finite state systems and is targeted towards the checking of the non-deducibility-on-compositions property.

2.2.4 Other Approaches

Other approaches that cannot be directly classified under the previous headings or that may cover more than one approach have also been suggested and implemented. The work of [65] is an early attempt to present an abstract binding-time analysis as part of a meta-interpreter for the π-calculus. The analysis adopts a partial evaluation technique, which aims at using the known part of the input data in specialising programs. The results of the analysis can determine the static communications that can be executed at compile-time as well as the set of static variables that do not change during run-time. The main motivation behind the work of [65] was program optimisation rather than security (although the results are general enough to include some notion of security).

The Mobility Workbench [128] is an example of automated tools that are targeted towards the checking of process equivalences. In particular, the tool checks for the open bisimulation property in the π-calculus [114].

In the work of [3] a typing system is given for a generic version of the spi calculus with constructor/destructor functions. Untyped predicate logic is also used to program a Prolog-based protocol checker that verifies secrecy properties. The methodology is extended, in [21], to check for the authenticity property. The equivalence of the two approaches is established. In [29, 30], a trace analysis is built over a symbolic version of the operational semantics of the spi calculus, and in [12], a well-crafted approach to the symbolic analysis of cryptographic processes, in general, is formalised. Earlier attempts to use symbolic semantics in the analysis of security protocols include [79, 11], where the reachability problem in the cryptographic protocols is discussed.

The works of [49] and [44] have concentrated on the use of automatic checking of equivalences of processes in the spi calculus. The former presents a framework for a trace-based analysis of cryptographic protocols, whereas the latter defines a new equivalence, known as fenced bisimilarity, which removes some of the infinite quantifiers associated with testing equivalence in the spi calculus.

Finally, we mention the works of [35] and [66]. In [35], belief logics have been used to analyse a number of cryptographic protocols and the paper is a notable piece of work. The model checking approach has also been used with the Brutus checker for a dialect of the spi calculus in [66]. More recently, the Mobility Model Checker (MMC) has been introduced in [135], as a model checker for the π-calculus, which uses tabled logic programming.

2.3 Denotational Semantics of Nominal Calculi

Apart from the original structural operational semantics that was proposed for the π-calculus [97], denotational semantic models have also been introduced over the past few years. Some of these include the works of [33, 36, 39, 54, 69, 71, 121]. Apart from a few examples, the majority of the work on denotational semantics of mobile systems has adopted an approach based on category theory. The obvious reason underlying this choice seems to be related to the fact that category theory offers abstract reasoning that can be easily utilised in deriving fully abstract denotational models.

The models of [54] and [121] are the first attempts to define a denotational meaning for processes in the π-calculus. Both are fully abstract with respect to late bisimilarity and equivalence. The major difference between the two is that [54] defines the semantics in terms of an internal meta-language for the main category. The language is, in fact, a type-equational theory based on a typed λ-calculus with non-deterministic summation and recursion. The interpretation of processes is then given by translating into this meta-

language. This renders the model of [54] a general framework that relies on a uniform approach suitable for modelling a variety of nominal calculi. For example, it is possible to add an interrupt operator to the π-calculus. On the other hand, the model of [121] is more abstract and concentrates on properties of the categorical solution. This is quite interesting when dealing with the static analysis framework, as it offers concrete grounds on which the analysis can be built[1].

Stark's model was further refined in [39] using presheaf categories. These are more flexible as they support function spaces and offer models for synchronisation trees and event structures. Unlike the models of [54, 121], the resulting model is capable of expressing early, as well as, late semantics and hence, it is fully abstract with respect to early bisimilarity and equivalence. Since our static analysis framework is not designed to necessarily deal with the early version of the semantics, we opted for the less elaborate model given in [121].

Two set-theoretic denotational models are presented in [71] that are fully abstract with respect to may- and must-testing in the π-calculus. The models are obtained as solutions of domain equations in a functor category and are quite close to the works of [54, 121]. A higher-order version of the π-calculus, where functions as well as names can be communicated over channels, is modelled denotationally in [69]. An internal language is also given for the solution $\mathcal{O}$-category, which is used in the process interpretation. The model is only proven to be fully abstract for transitions (computations) in the higher-order language.

In [33], a new notion of Cartesian closed double categories is developed from the theory of Cartesian closed categories and used as a basis for the interpretation of the π-calculus. λ-notation is also used to develop a type system for the resulting interpretation. A final semantics in the style of [10] is also developed using categorical modelling in [78]. The resulting model gives a higher-order presentation of the π-calculus using the Edinburgh Logical Framework (LF) (based on typed λ-calculus) as the meta-language. Strong and weak late semantics are both dealt with. However, the full abstraction of the model is not shown with respect to the corresponding bisimilarity and congruence relations. The result is only shown for transitions.

Finally, away from the category-based models, in the work of [36], a Petri net semantics is developed for the π-calculus with recursive definitions [114]. The semantics is based on Place/Transition nets with inhibitor arcs (i.e. PTI-nets) and is proven to be sound with respect to early transition semantics of the π-calculus. The model also gives a definition of non-interleaving semantics, which are used to generate causal trees.

[1]The semantic domains were concretely defined in an unpublished note by Stark. This note can be downloaded from `http://www.dcs.ed.ac.uk/home/stark/publications/fuladm.html`

2.4 Conclusion

In this chapter, we have covered related work carried out within the areas of static program analysis and denotational semantics for mobile systems and cryptographic protocols specified in the languages of the π-calculus and the spi calculus, respectively. One may identify from such related work a general lack of overlapping research between these two areas (static analysis and denotational semantics) in the context of security analysis of processes in the π-calculus and spi calculus. Most of the static analysis work has concentrated on the use of operational semantics. The use of a denotational framework would benefit both the theory and implementation of static analysis in this area. Theoretically, concepts like least fixed points are inherent in the nature of the denotational approach and, practically, the implementation of these concepts is straightforward in the functional programming paradigm, which is closely related to the denotational approach.

Chapter 3

Nominal Calculi

3.1 Introduction

The concept of a name as the main abstraction underlying different low-level details of computing systems has resulted in the proliferation of a family of models of computation known as *nominal calculi*. Many such models are *process algebraic*; programs are modelled as processes equipped with operations to express their evolvement. The π-calculus has emerged in recent years as an authoritative and highly expressive nominal calculus for the modelling of mobile systems. In this chapter, we review the syntax and structural operational semantics of the π-calculus and its cryptographic extension, the spi calculus. We give denotational semantics for both languages that are based on domain theory.

3.2 The π-calculus

For years, the π-calculus has provided a powerful, yet simple, theoretical framework for modelling and reasoning about mobility. The notion of a *name* constitutes the central concept on which the calculus is based. Names are allowed to stand for channels as well as data. Therefore, by sending and receiving names, processes can communicate information to other processes and can also change the topology of their network. This view of mobility is described as "the movement of links in the virtual space of linked processes" [96, II-8].

Several versions of the π-calculus exist, including the monadic synchronous version, which was originally developed in [97]. In this version, communications occur by synchronising channels and carry messages that are single names. Other versions exist such as the polyadic π-calculus [95] and the asynchronous π-calculus [31, 76].

3.2.1 Syntax

In the π-calculus, names constitute a countably infinite set $x, y, z \in \mathcal{N}$ possibly subscripted with numbers, as in $x_1, x_2 \ldots$. Processes $P, Q \in \mathcal{P}$ are built from names according to the syntax of Figure 3.1.

P, Q	:=	$\mathbf{0}$	Null
		$x(y).P$	Input action
		$\overline{x}\langle y\rangle.P$	Output action
		if $x = y$ *then* P *else* Q	Conditional
		$(\nu x)P$	Restriction
		$P + Q$	Summation
		$P \mid Q$	Parallel composition
		$!P$	Replication

Figure 3.1: Syntax of the π-calculus.

Informally, these rules are described as follows. A null process, $\mathbf{0}$, is an inactive process that cannot evolve any further. A guarded process may perform an input action, $x(y).P$, or an output action, $\overline{x}\langle y\rangle.P$, and then continues as the residue, P. For the case of $x(y).P$, a message z received along x will substitute y as in $P[z/y]$. A conditional process, *if* $x = y$ *then* P *else* Q, compares two names, x and y, and if they are the same, it continues as P, else it continues as Q. A restriction, $(\nu x)P$, constrains the scope of a name, x, to a process P. The summation, $P + Q$, chooses non-deterministically between two processes, P and Q, discarding the other. The parallel composition, $P \mid Q$, interleaves P and Q allowing for any communications between the two processes to occur. Finally, the replication, $!P$, spawns as many parallel copies of P as required by the context.

The standard notions of α-conversion and name substitution apply, as well as the notions of free names, $fn(P)$, bound names, $bn(P)$, and the set of all the names of a process, $n(P) = fn(P) \cup bn(P)$. A name y is bound in $(\nu y)P$ and $x(y).P$, otherwise it is free. Using α-conversion, it is possible to have a set of bound names, $bn(P)$, that is totally distinct, i.e. no two bound names of a process are the same. Also, α-conversion can be used to achieve the requirement that $bn(P) \cap fn(P) = \{\}$. In all the systems we deal with, these distinction requirements are always assumed to hold.

3.2.2 Structural Operational Semantics

We adopt the version of the structural operational semantics for the π-calculus as given in [39]. The semantics is defined in terms of two relations: the *structural congruence* relation, $\equiv$, and the *labelled transition* relation, $\xrightarrow{\pi}$.

The structural congruence relation is defined as the smallest equivalence on processes that satisfies the following rules and is closed by the renaming of bound names (α-conversion):

- $(\mathcal{P}/\equiv, |, \mathbf{0})$ and $(\mathcal{P}/\equiv, +, \mathbf{0})$ are commutative monoids
- $(\nu x)\mathbf{0} \equiv \mathbf{0}$
- $(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$
- $(\nu x)(P \mid Q) \equiv (P \mid (\nu x)Q), \qquad \text{if } x \notin fn(P)$
- $!P \equiv P \mid\ !P$
- *if* $x = y$ *then* P *else* $Q \equiv \begin{cases} P, & \text{if } x = y \\ Q, & \text{otherwise} \end{cases}$

The labelled transition relation is defined by the set of rules of Figure 3.2 and closed by the structural reshaping of processes. In these rules, π-transitions represent input actions, $x(y)$, free output actions, $\overline{x}\langle y\rangle$, bound output actions, $\overline{x}(y)$ ($\equiv\ (\nu\, y)\overline{x}\langle y\rangle$), and silent actions, τ.

(Out)	$\overline{x}\langle y\rangle.P \xrightarrow{\overline{x}\langle y\rangle} P$	
(In)	$x(y).P \xrightarrow{x(y)} P$	
(Open)	$P \xrightarrow{\overline{x}\langle y\rangle} P' \Rightarrow (\nu\, y)P \xrightarrow{\overline{x}(y)} P'$	if $x \neq y$
(Res)	$P \xrightarrow{\pi} P' \Rightarrow (\nu\, x)P \xrightarrow{\pi} (\nu\, x)P'$	if $x \notin fn(\pi)$
(Sum)	$P \xrightarrow{\pi} P' \Rightarrow P + Q \xrightarrow{\pi} P'$	
(Par)	$P \xrightarrow{\pi} P' \Rightarrow P \mid Q \xrightarrow{\pi} P' \mid Q$	
(Comm)	$P \xrightarrow{\overline{x}\langle y\rangle} P', Q \xrightarrow{x(z)} Q' \Rightarrow P \mid Q \xrightarrow{\tau} P' \mid Q'[y/z]$	
(Close)	$P \xrightarrow{\overline{x}(y)} P', Q \xrightarrow{x(z)} Q' \Rightarrow P \mid Q \xrightarrow{\tau} (\nu\, y)(P' \mid Q'[y/z])$	

Figure 3.2: Rules of the labelled transition relation in the π-calculus.

The relation is *late* in the sense that input variables are not instantiated until actual communications take place (in Rules (Comm) and (Close)). The transition rules (Out) and (In) describe transitions of processes guarded by output and input actions. Rule (Open) changes a free output to a bound output by restricting the message of communication. The (Res), (Sum) and (Par) rules preserve transitions under the restriction, summation and

parallel composition operators. Finally, communications are described in rules (Comm) (free output actions) and (Close) (bound output actions), where the external effect of a communication appears as the silent action, τ. In rule (Close), the scope of the communicated message is extruded to include the recipient process.

3.2.3 Denotational Semantics

Our starting point for the development of the denotational semantics is the set of predomain equations given by Stark [121], which describe a process in terms of the most basic actions it can perform, i.e. input, output and silent actions as well as deadlock (termination):

$$Pi \cong 1 + \mathbb{P}(Pi_\perp + In + Out) \tag{3.1}$$

$$In \cong N \times (N \to Pi_\perp) \tag{3.2}$$

$$Out \cong N \times (N \times Pi_\perp + N \to Pi_\perp) \tag{3.3}$$

Where Pi, In and Out represent the predomains of processes, input and output actions, respectively, allowing for bound output actions. N is the predomain of names, and $\mathbb{P}(-)$ is Plotkin's powerdomain whose result is joined with the one-point domain, 1, following Abramsky's result [8, Def. 3.4]. This is necessary to be able to express deadlock (or termination). The domain of processes, $Pi_\perp$, is then obtained by lifting Pi.

The binding effects inherent in input and bound output actions are expressed as functions $N \to Pi_\perp$. These functions accept any name and instantiate a process (the residue) with that name. Functions are also suitable since they allow for the renaming of bound names (α-conversion) to take place without affecting the denotational meaning of a process. In [121], a special function, $\multimap$, is used to achieve a name freshness requirement in bound output actions. As we explain later in this section, we use a special labelling mechanism in the denotational semantics, which achieves a similar effect by maintaining that new copies of bound names created are always distinct.

In addition to equations (3.1)–(3.3), the following mappings, leading into $Pi_\perp$, are defined by Stark [121], who based them on similar mappings introduced by Abramsky [8, Def. 3.3]:

$$\emptyset : 1 \to Pi_\perp \tag{3.4}$$

$$\{\!|-|\!\} : (Pi_\perp + In + Out)_\perp \to Pi_\perp \tag{3.5}$$

$$\uplus : Pi_\perp \times Pi_\perp \to Pi_\perp \tag{3.6}$$

$$new : (N \to Pi_\perp) \to Pi_\perp \tag{3.7}$$

According to Abramsky [8, Def. 3.3], the above mappings are continuous (this was not mentioned by Stark in [121]).

The empty set map, $\emptyset$, denotes a deadlocked or terminated process. The singleton map, $\{\!|-|\!\}$, is used to interpret elements of the input and output predomains as well as processes guarded by silent actions as elements of $Pi_\perp$. The choice map, $\uplus$, is the standard powerdomain union of two elements in $Pi_\perp$. Finally, the *new* operation is useful in interpreting the effects of restriction on processes. In [121], an extra mapping, *par*, is added to interpret the effects of parallelism on processes. Our version of the denotational semantics deals with parallelism by using a special multiset of processes running in parallel with the interpreted process.

A solution to equations (3.1)–(3.3) and the associated mappings (3.4)–(3.7) is given in [121] using symmetric monoidal closed functor categories. In our case, we adopt a more concrete solution based on domain theory. Such a concrete solution is sufficient for the static analysis framework defined in later chapters, where definitions of the concrete domains are required without reference to the underlying abstract functorial meaning.

We now turn our attention to the definition of $Pi_\perp$. Assuming $\mathcal{K}$ is the set underlying any domain, then elements $p, q \in \mathcal{K}(Pi_\perp)$ can be defined in two stages. First, elements arising from $\emptyset$, $\{\!|-|\!\}$ and $\uplus$ are defined as in Figure 3.3.

Elements of In :		
$x \in \mathcal{K}(N), \lambda y.p \in \mathcal{K}(N \to Pi_\perp)$	$\Rightarrow$	$(x, \lambda y.p) \in \mathcal{K}(In)$
Elements of Out :		
$x, y \in \mathcal{K}(N), p \in \mathcal{K}(Pi_\perp)$	$\Rightarrow$	$(x, y, p) \in \mathcal{K}(Out)$
$x \in \mathcal{K}(N), \lambda y.p \in \mathcal{K}(N \to Pi_\perp)$	$\Rightarrow$	$(x, \lambda y.p) \in \mathcal{K}(Out)$
Elements of $Pi_\perp$:		
$\{\!\|\perp\|\!\} \in \mathcal{K}(Pi_\perp), \emptyset \in \mathcal{K}(Pi_\perp)$		
$p, q \in \mathcal{K}(Pi_\perp)$	$\Rightarrow$	$p \uplus q \in \mathcal{K}(Pi_\perp)$
$p \in \mathcal{K}(Pi_\perp)$	$\Rightarrow$	$\{\!\|tau(p)\|\!\} \in \mathcal{K}(Pi_\perp)$
$i \in \mathcal{K}(In)$	$\Rightarrow$	$\{\!\|in(i)\|\!\} \in \mathcal{K}(Pi_\perp)$
$o \in \mathcal{K}(Out)$	$\Rightarrow$	$\{\!\|out(o)\|\!\} \in \mathcal{K}(Pi_\perp)$

Figure 3.3: Elements of *In*, *Out* and $Pi_\perp$.

According to equation (3.2), elements of the predomain of input actions, *In*, are pairs $(x, \lambda y.p)$, where the first element, x, stands for the channel of communication and the second element is a function, $\lambda y.p$, that accepts a name, z, and continues as a process, $p[z/y] \in Pi_\perp$.

According to equation (3.3), elements of Out are either free output actions, (x, y, p), or bound output actions, $(x, \lambda y.p)$. A free output consists of the channel x, the free message y and the residue p. A bound output, on the other hand, is expressed as the channel x and a function, $\lambda y.p$, where y is the message of communication bound to the residue p.

Building on elements of In and Out, elements of the summation $Pi_\perp + In + Out$ are ensured to be distinct by tagging each element with one of the appropriate tags tau, in and out. By applying the singleton mapping, $\{\!|-|\!\}$, it is possible then to create elements of the powerdomain $Pi_\perp$ that are singleton sets containing elements of the summation $Pi_\perp + In + Out$. The undefined process, $\{\!|\perp|\!\}$, is the bottom element necessary in order to perform fixed-point calculations over the domain $Pi_\perp$. Using $\emptyset$, it is possible to include the empty set element in $Pi_\perp$ [8, Def. 3.4]. Finally, $\uplus$, combines two elements of $Pi_\perp$.

The second stage consists of interpreting the effects of restriction on processes. This is done by giving a concrete definition for new (3.7) over elements $p \in Pi_\perp$, as in Figure 3.4.

$$
\begin{array}{lcl}
new(\lambda x.\emptyset) & = & \emptyset \\
new(\lambda x.\{\!|\perp|\!\}) & = & \{\!|\perp|\!\} \\
new(\lambda x.\{\!|in(y, \lambda z.p)|\!\}) & = & \begin{cases} \emptyset, & \text{if } x = y \\ \{\!|in(y, \lambda z.new(\lambda x.p))|\!\}, & \text{otherwise} \end{cases} \\
new(\lambda x.\{\!|out(y, z, p)|\!\}) & = & \begin{cases} \emptyset, & \text{if } x = y \\ \{\!|out(y, \lambda z.p)|\!\}, & \text{if } x = z \neq y \\ \{\!|out(y, z, new(\lambda x.p))|\!\}, & \text{otherwise} \end{cases} \\
new(\lambda x.\{\!|out(y, \lambda z.p)|\!\}) & = & \begin{cases} \emptyset, & \text{if } x = y \\ \{\!|out(y, \lambda z.new(\lambda x.p))|\!\}, & \text{otherwise} \end{cases} \\
new(\lambda x.\{\!|tau(p)|\!\}) & = & \{\!|tau(new(\lambda x.p))|\!\} \\
new(\lambda x.(p_1 \uplus p_2)) & = & new(\lambda x.p_1) \ \uplus \ new(\lambda x.p_2)
\end{array}
$$

Figure 3.4: The definition of new for the π-calculus.

In general, new captures deadlocked situations arising from the attempt to communicate over restricted non-extruded channels. It also turns a free output action into a bound output action once a restricted message is directly sent over a channel (scope extrusion). In all the other cases, restriction has no effect and it is simply passed to the residue or distributed over the choice operation.

After fully defining elements of the domain $Pi_\perp$, it is possible to denote the meaning of processes in the π-calculus in terms of these elements. For a process, P, we define the semantic function $\mathcal{S}^\pi(\!|P|\!)\ \rho\ \phi_\mathcal{S} \in Pi_\perp$ by the set of rules of Figure 3.5. An earlier version of

the semantics was given in [14], based on Stark's original categorical solution.

$$
\begin{array}{lll}
(\mathcal{S}^\pi 1) & \mathcal{S}^\pi(\llbracket \mathbf{0} \rrbracket)\ \rho\ \phi_{\mathcal{S}} & = \emptyset \\
(\mathcal{S}^\pi 2) & \mathcal{S}^\pi(\llbracket x(y).P \rrbracket)\ \rho\ \phi_{\mathcal{S}} & = \{\!|\, in(\varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, x), \lambda y.\mathcal{R}^\pi(\llbracket \{\!|P|\!\}_\rho \rrbracket)\ \phi_{\mathcal{S}})\,|\!\} \\
(\mathcal{S}^\pi 3) & \mathcal{S}^\pi(\llbracket \overline{x}\langle y\rangle.P \rrbracket)\ \rho\ \phi_{\mathcal{S}} & = \\
 & \Big(\biguplus_{x'(z).P' \in \rho} \{\!|\, tau(p)\,|\!\}\Big) \uplus \{\!|\, out(\varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, x), \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, y), \mathcal{R}^\pi(\llbracket \{\!|P|\!\}_\rho \rrbracket)\ \phi_{\mathcal{S}})\,|\!\} & \\
 & \text{where, } p = \mathcal{R}^\pi(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \rho[P'/x'(z).P'] \rrbracket)\ \phi_{\mathcal{S}}[z \mapsto \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, y)] & \\
 & \text{if, } \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, x) = \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, x') & \\
(\mathcal{S}^\pi 4) & \mathcal{S}^\pi(\llbracket \textit{if } x = y \textit{ then } P \textit{ else } Q \rrbracket)\ \rho\ \phi_{\mathcal{S}} = & \\
 & \quad \begin{cases} \mathcal{R}^\pi(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi_{\mathcal{S}}, & \text{if } \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, x) = \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, y) \\ \mathcal{R}^\pi(\llbracket \{\!|Q|\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi_{\mathcal{S}}, & \text{otherwise} \end{cases} & \\
(\mathcal{S}^\pi 5) & \mathcal{S}^\pi(\llbracket P + Q \rrbracket)\ \rho\ \phi_{\mathcal{S}} & = \mathcal{R}^\pi(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi_{\mathcal{S}} \uplus \mathcal{R}^\pi(\llbracket \{\!|Q|\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi_{\mathcal{S}} \\
(\mathcal{S}^\pi 6) & \mathcal{S}^\pi(\llbracket P \mid Q \rrbracket)\ \rho\ \phi_{\mathcal{S}} & = \mathcal{R}^\pi(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \{\!|Q|\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi_{\mathcal{S}} \\
(\mathcal{S}^\pi 7) & \mathcal{S}^\pi(\llbracket (\nu x)P \rrbracket)\ \rho\ \phi_{\mathcal{S}} & = new(\lambda x.\mathcal{R}^\pi(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi_{\mathcal{S}}) \\
(\mathcal{S}^\pi 8) & \mathcal{S}^\pi(\llbracket !P \rrbracket)\ \rho\ \phi_{\mathcal{S}} & = \mathcal{F}^\pi(-1) \\
 & \text{where, } \mathcal{F}^\pi(n) = \textit{let } p_1 = \mathcal{S}^\pi(\llbracket \prod_{i=1}^{n} P[bn_i(P)/bn(P)] \rrbracket)\ \rho\ \phi_{\mathcal{S}} \textit{ in} & \\
 & \qquad \textit{let } p_2 = \mathcal{S}^\pi(\llbracket \prod_{i=1}^{n+2} P[bn_i(P)/bn(P)] \rrbracket)\ \rho\ \phi_{\mathcal{S}} \textit{ in} & \\
 & \qquad \textit{if } p_1 = p_2 \textit{ then } p_1 \textit{ else } \mathcal{F}^\pi(n+1) & \\
 & \text{and, } bn_i(P) = \{x_i \mid x \in bn(P)\} & \\
(\mathcal{R}^\pi 0) & \mathcal{R}^\pi(\llbracket \rho \rrbracket)\ \phi_{\mathcal{S}} & = \biguplus_{P \in \rho} \mathcal{S}^\pi(\llbracket P \rrbracket)\ (\rho \backslash \{\!|P|\!\}_\rho)\ \phi_{\mathcal{S}}
\end{array}
$$

Figure 3.5: The denotational semantics of the π-calculus.

The multiset, ρ, contains all processes composed in parallel with the analysed process, P. The standard singleton, $\{\!|-|\!\}_\rho : \mathcal{P} \to \wp(\mathcal{P})$, and the multiset union, $\uplus_\rho : \wp(\mathcal{P}) \times \wp(\mathcal{P}) \to \wp(\mathcal{P})$, are special mappings defined over ρ and should not be confused with $\{\!|-|\!\}$ and $\uplus$ defined earlier in (3.5) and (3.6). The special environment, $\phi_{\mathcal{S}} : N \to N_\perp$, maps a name to another name that substitutes it in the semantics. In fact, this environment will hold substitutions of input parameters by messages received during communications. Since the standard denotational semantics is a precise semantics, an input parameter can only be mapped to, at most, one name in any possible choice of control flow (i.e. in any side of $\uplus$).

The $\varphi_{\mathcal{S}} : (N \to N_\perp) \times \mathcal{N} \to N$ function is used to retrieve the true value of a name, x,

given substitutions recorded in $\phi_{\mathcal{S}}$:

$$\varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, x) = \begin{cases} z, & \text{if } \phi_{\mathcal{S}}(x) = z \\ x, & \text{otherwise, when } \phi_{\mathcal{S}}(x) = \perp \end{cases}$$

The meaning of the composed processes in ρ is given by rule ($\mathcal{R}^{\pi}0$) as the summation of the individual meaning of each of the composed processes. The concrete semantics of each process construct is defined by the cases of rules ($\mathcal{S}^{\pi}1$–$\mathcal{S}^{\pi}8$). Rule ($\mathcal{S}^{\pi}1$) interprets the meaning of a null process directly as the empty set map, $\emptyset$. Rules ($\mathcal{S}^{\pi}2$) and ($\mathcal{S}^{\pi}3$) deal with the cases of processes guarded by input and output actions after which the residues are composed with elements of the ρ multiset. In ($\mathcal{S}^{\pi}3$), any communications that may take place between the output action and appropriate input actions guarding processes in ρ are taken into consideration and $\phi_{\mathcal{S}}$ is updated accordingly. The interpretation is a summation of all such communications and the no-communication case. This preserves the associativity property of the parallel composition operator, $P \mid Q$.

Rule ($\mathcal{S}^{\pi}4$) interprets conditional statements based on matching the values of names x and y returned by $\varphi_{\mathcal{S}}$. Depending on the values of these names, the equality condition will either hold or not, hence determining the process composed with ρ. Rule ($\mathcal{S}^{\pi}5$) interprets separately each of the processes in a choice as composed with processes in ρ and then distributes the choice over the two results. Rule ($\mathcal{S}^{\pi}6$) is straightforward allowing for two parallel processes to be composed with the rest of processes in ρ. Rule ($\mathcal{S}^{\pi}7$) interprets the meaning of restriction using the *new* operation, defined earlier in Figure 3.4.

Finally, rule ($\mathcal{S}^{\pi}8$) interprets a replicated process, $!P$, by calculating the special function, $\mathcal{F}^{\pi} : \mathbb{N} \rightarrow Pi_{\perp}$, starting from the bottom number, $n = -1$, which computes the bottom element, $\mathcal{S}^{\pi}(\![\prod_{i=1}^{-1} P[bn_i(P)/bn(P)]]\!)\, \rho\, \phi_{\mathcal{S}} = \{\!|\perp|\!\}$. The calculation continues by incrementing n until the condition of the *if*-statement holds true. In other words, the least fixed-point is reached. Due to the fact that the semantic domain, $Pi_{\perp}$, is infinite, this calculation is not guaranteed to terminate at a finite number, n.

A labelling mechanism is also used in the rule to perform α-conversion on the spawned copies, P, by subscripting all the bound names of P. The renaming is necessary to maintain the distinction of bound names from other bound and free names. Hence, input parameters recorded in $\phi_{\mathcal{S}}$ can be mapped to at most one name. It is interesting to note that the compositionality rule of the denotational semantics is preserved under this α-conversion, since the resulting process on the right side of the rule, $P[bn_i(P)/bn(P)]$, is still a subprocess of the replication $!P$, by being structurally equivalent to the subprocess, P. In fact, the substitution, $P[bn_i(P)/bn(P)]$, will only affect bound variables of the λ-abstractions in rules ($\mathcal{S}^{\pi}2$) and ($\mathcal{S}^{\pi}7$), leaving the denotational meaning as given by these two rules intact.

To relate the denotational semantics of the π-calculus to its structural operational semantics, the following theorem states that the elements of the domain of processes, $Pi_\perp$, reflect and preserve transitions in the structural operational semantics.

Theorem 1 (Soundness and Adequacy of $Pi_\perp$ w.r.t. transitions)
The interpretation of processes in $Pi_\perp$ is sound and adequate with respect to transitions in the structural operational semantics of the π-calculus.

Proof. The soundness property relies on the ability of semantic elements to match transitions in the operational model:

$$P \xrightarrow{\overline{x}\langle y\rangle} Q \quad \Rightarrow \quad out(x, y, \mathcal{S}^\pi(\llbracket Q \rrbracket)\ \rho\ \phi_\mathcal{S}) \in \mathcal{S}^\pi(\llbracket P \rrbracket)\ \rho\ \phi_\mathcal{S}$$
$$P \xrightarrow{\overline{x}(y)} Q \quad \Rightarrow \quad out(x, \lambda y.\mathcal{S}^\pi(\llbracket Q \rrbracket)\ \rho\ \phi_\mathcal{S}) \in \mathcal{S}^\pi(\llbracket P \rrbracket)\ \rho\ \phi_\mathcal{S}$$
$$P \xrightarrow{x(y)} Q \quad \Rightarrow \quad in(x, \lambda y.\mathcal{S}^\pi(\llbracket Q \rrbracket)\ \rho\ \phi_\mathcal{S}) \in \mathcal{S}^\pi(\llbracket P \rrbracket)\ \rho\ \phi_\mathcal{S}$$
$$P \xrightarrow{\tau} Q \quad \Rightarrow \quad tau(\mathcal{S}^\pi(\llbracket Q \rrbracket)\ \rho\ \phi_\mathcal{S}) \in \mathcal{S}^\pi(\llbracket P \rrbracket)\ \rho\ \phi_\mathcal{S}$$

On the other hand, adequacy requires that the semantic transitions be mapped correctly to the operational model:

$$out(x, y, q) \in \mathcal{S}^\pi(\llbracket P \rrbracket)\ \rho\ \phi_\mathcal{S} \quad \Rightarrow \quad \exists Q : P \xrightarrow{\overline{x}\langle y\rangle} Q\ \wedge\ \mathcal{S}^\pi(\llbracket Q \rrbracket)\ \rho\ \phi_\mathcal{S} = q$$
$$out(x, \lambda y.q) \in \mathcal{S}^\pi(\llbracket P \rrbracket)\ \rho\ \phi_\mathcal{S} \quad \Rightarrow \quad \exists Q : P \xrightarrow{\overline{x}(y)} Q\ \wedge\ \mathcal{S}^\pi(\llbracket Q \rrbracket)\ \rho\ \phi_\mathcal{S} = q$$
$$in(x, \lambda y.q) \in \mathcal{S}^\pi(\llbracket P \rrbracket)\ \rho\ \phi_\mathcal{S} \quad \Rightarrow \quad \exists Q : P \xrightarrow{x(y)} Q\ \wedge\ \mathcal{S}^\pi(\llbracket Q \rrbracket)\ \rho\ \phi_\mathcal{S} = q$$
$$tau(q) \in \mathcal{S}^\pi(\llbracket P \rrbracket)\ \rho\ \phi_\mathcal{S} \quad \Rightarrow \quad \exists Q : P \xrightarrow{\tau} Q\ \wedge\ \mathcal{S}^\pi(\llbracket Q \rrbracket)\ \rho\ \phi_\mathcal{S} = q$$

Both these properties can be shown to hold for the model of $Pi_\perp$. Soundness can be proven by showing that the rules of the denotational semantics respect rules of the structural operational semantics, whereas adequacy can be proven by defining a formal approximation relation, $p \lhd P$, between elements $p \in Pi_\perp$ and processes $P \in \mathcal{P}$. Such a relation is often used in adequacy proofs for the λ-calculus (for example, as in [109]). □

3.3 The Spi Calculus

Of the many extensions that have been proposed to the π-calculus, the spi calculus remains one of the most popular in the security community. The original work introduced in [5] extended the language of the π-calculus with cryptographic primitives allowing processes to send and receive private information over public channels in a secure manner. This implies that processes exhibit a *message-processing* behaviour in addition to the *message-passing* behaviour inherited from the π-calculus.

The denotational semantics we propose for the spi calculus builds on an extension of Stark's predomain equations for the π-calculus [121]. The extension is capable of handling the complexity inherent in the communicated data structures as well as the possibility of extruding multiple names through those structures.

3.3.1 Syntax

We review here the syntax of the spi calculus [5] without hashing functions. Unlike the π-calculus, where names are the only terms, in the spi calculus, terms are divided into two types: *primitive* and *complex*. The notion of primitive terms is a refined notion of names that distinguishes between constants $a, b, c, k, A, B, C \ldots \in \mathcal{N}$ that cannot be instantiated, and variables $x, y, z, X, Y \ldots \in \mathcal{V}$ that can be instantiated. We allow primitive terms to be subscripted with numbers. Complex terms are obtained by applying cryptographic and tuple creation operations. Based on this, one may define terms $L, M, N \in \mathit{Term}$ and processes $P, Q \in \mathcal{P}$ according to the syntax of Figure 3.6. We write k^+ and k^- to distinguish between the public and private parts of a key pair. These parts are related by the fact that they can reverse each other's encrypting effects.

Informally, the syntactic rules are described as follows. A null process, $\mathbf{0}$, is incapable of evolving any further. An input guard, $M(x).P$, allows for a process to input a message, N, over a channel defined by M and continues as the residue $P[N/x]$. An output guard, $\overline{M}\langle N\rangle.P$, allows a process to send a message, N, over the channel defined by M and continues as P. Although the forms of the input/output actions allow for the generic use of terms as channels, this is only valid for the cases where the term is a name or a variable that is instantiated to a name. The same is required of keys, since keys are treated as names due to their unguessable nature. The restriction, parallel composition, replication and conditional processes are all inherited from the π-calculus.

A tuple splitting operation, *let* $(x_1, \ldots, x_n) = M$ *in* P *else* Q, attempts to split a term, M, and binds the resulting components to local variables, $x_1, \ldots, x_n$, in a process P. However, if this fails (i.e. M has an incorrect arity), then a different process, Q, is chosen. Secret-key decryption, *case* L *of* $\{x\}_N$ *in* P *else* Q, and public-key decryption, *case* L *of* $\{\![x]\!\}_N$ *in* P *else* Q, attempt to decrypt a term L with key N. If successful, the resulting term is bound to a local variable x in some process, P, otherwise, a different process, Q, is chosen. Similarly, the signature verification process, *case* L *of* $[\![x]\!]_N$ *in* P *else* Q, attempts to verify a term, L, using N and if successful, the resulting term is bound to a local variable, x, in P. Otherwise, if the verification fails, a different process, Q, is chosen. For the

M, N	$:=$		Terms
		x, y, z	Variables
		a, b, c	Names
		k^+	Public key
		k^-	Private key
		$(M_1, \ldots, M_n)$	n-tuple
		$\{M\}_N$	Secret-key encryption
		$\{\!\lvert M \rvert\!\}_N$	Public-key encryption
		$[\!\{ M \}\!]_N$	Digital signature
P, Q	$:=$		Processes
		$\mathbf{0}$	Null
		$M(x).P$	Input guard
		$\overline{M}\langle N \rangle.P$	Output guard
		$(\nu a)P$	Restriction
		$P \mid Q$	Parallel composition
		$!P$	Replication
		if $M = N$ *then* P *else* Q	Conditional
		let $(x_1, \ldots, x_n) = M$ *in* P *else* Q	Tuple splitting
		case L *of* $\{x\}_N$ *in* P *else* Q	Secret-key decryption
		case L *of* $\{\!\lvert x \rvert\!\}_N$ *in* P *else* Q	Public-key decryption
		case L *of* $[\!\{ x \}\!]_N$ *in* P *else* Q	Signature verification

Figure 3.6: The syntax of the spi calculus.

sake of conciseness, we omit all the above *else* parts in the *let* and *case* process constructs whenever $Q = \mathbf{0}$.

The standard notions of term substitution and α-conversion, as well as the free and bound variables and free and bound names of processes and terms, $fv(A)$, $bv(A)$, $fn(A)$, $bn(A)$, all apply as usual. We also use the notion of bound names and variables of a process, written as $bnv(P) = bn(P) \cup bv(P)$. A process P or a term M is *closed* if $fv(P) = \{\}$ or $fv(M) = \{\}$. Henceforth, we shall only deal with closed processes and closed terms. Moreover, we only consider systems whose bound names and bound variables are initially distinct from each other and from the set of free names of the particular process.

3.3.2 Structural Operational Semantics

The structural operational semantics of the spi calculus is a direct extension of the structural operational semantics of the π-calculus given in Section 3.2.2. The structural congruence relation, $\equiv$, is defined in a similar manner to before, with the exceptions that the restriction operator is only applicable to names, and the commutative monoid $(\mathcal{P}/\equiv, +, \mathbf{0})$ disappears.

The structural congruence relation is defined as the smallest equivalence, closed by the renaming of bound names and variables (α-conversion), that satisfies the following rules:

- $(\mathcal{P}/\equiv, |, \mathbf{0})$ is a commutative monoid
- $(\nu\, a)\mathbf{0} \equiv \mathbf{0}$
- $(\nu\, a)(\nu\, b)P \equiv (\nu\, b)(\nu\, a)P$
- $(\nu\, a)(P \mid Q) \equiv (P \mid (\nu\, a)Q)$ if $a \notin fn(P)$
- $!P \equiv P \mid\ !P$
- $\mathit{if}\ M = N\ \mathit{then}\ P\ \mathit{else}\ Q \quad \equiv \begin{cases} P, & \text{if } M = N \\ Q, & \text{if } M \neq N \end{cases}$
- $\mathit{let}\ (x_1, \ldots, x_n) = L\ \mathit{in}\ P\ \mathit{else}\ Q \quad \equiv \begin{cases} P[M_1/x_1, \ldots, M_n/x_n], & \\ & \text{if } L = (M_1, \ldots, M_n) \\ Q, & \text{otherwise} \end{cases}$
- $\mathit{case}\ L\ \mathit{of}\ \{x\}_k\ \mathit{in}\ P\ \mathit{else}\ Q \quad \equiv \begin{cases} P[M/x], & \text{if } L = \{M\}_k \\ Q, & \text{otherwise} \end{cases}$
- $\mathit{case}\ L\ \mathit{of}\ \{\!|x|\!\}_{k^-}\ \mathit{in}\ P\ \mathit{else}\ Q \quad \equiv \begin{cases} P[M/x], & \text{if } L = \{\!|M|\!\}_{k^+} \\ Q, & \text{otherwise} \end{cases}$
- $\mathit{case}\ L\ \mathit{of}\ [\!\{x\}\!]_{k^+}\ \mathit{in}\ P\ \mathit{else}\ Q \quad \equiv \begin{cases} P[M/x], & \text{if } L = [\!\{M\}\!]_{k^-} \\ Q, & \text{otherwise} \end{cases}$

The labelled transition relation, $\xrightarrow{s\pi}$, is defined by the set of rules of Figure 3.7, closed by the structural congruence of processes. $s\pi$-transitions include input actions, $m(x)$, free output actions, $\overline{m}\langle N\rangle$, bound output actions, $(\nu\, n_1, \ldots, \nu\, n_k)\overline{m}\langle N\rangle$ (where $\{n_1, \ldots, n_k\} \subseteq fn(N)$) and silent actions, τ. The rules are described as follows. Rules (Out) and (In) express free output and input transitions. Rule (Open) transforms a free output to a bound output by restricting names appearing free in the message of communication. Rules (Res)

(Out)	$\overline{m}\langle N\rangle.P \xrightarrow{\overline{m}\langle N\rangle} P$
(In)	$m(x).P \xrightarrow{m(x)} P$
(Open)	$P \xrightarrow{\overline{m}\langle N\rangle} P' \Rightarrow (\nu\, n_1, \ldots, \nu\, n_k)P \xrightarrow{(\nu\, n_1,\ldots,\nu\, n_k)\overline{m}\langle N\rangle} P'$ if $m \notin \{n_1, \ldots, n_k\}$ and $\{n_1, \ldots, n_k\} \subseteq fn(N)$
(Res)	$P \xrightarrow{s\pi} P' \Rightarrow (\nu\, n)P \xrightarrow{s\pi} (\nu\, n)P'$ if $n \notin fn(s\pi)$
(Par)	$P \xrightarrow{s\pi} P' \Rightarrow P \mid Q \xrightarrow{s\pi} P' \mid Q$
(Comm)	$P \xrightarrow{\overline{m}\langle N\rangle} P', Q \xrightarrow{m(x)} Q' \Rightarrow P \mid Q \xrightarrow{\tau} P' \mid Q'[N/x]$
(Close)	$P \xrightarrow{(\nu\, n_1,\ldots,\nu\, n_k)\overline{m}\langle N\rangle} P', Q \xrightarrow{m(x)} Q' \;\Rightarrow\; P \mid Q \xrightarrow{\tau} (\nu\, n_1, \ldots, \nu\, n_k)(P' \mid Q'[N/x])$

Figure 3.7: Rules of the labelled transition relation in the spi calculus.

and (Par) state that transitions are preserved under the restriction and parallel composition operators, provided that the side conditions are satisfied. Finally, communications are carried out in rules (Comm) and (Close) for the cases of free and bound output actions, respectively. The external effects of these communications appear as silent actions, τ.

3.3.3 Denotational Semantics

The denotational semantics of the spi calculus builds on an extension of Stark's predomain equations for the π-calculus (Section 3.2.3). The extended equations are as follows:

$$Spi \cong 1 + \mathbb{P}(Spi_\perp + In + Out) \tag{3.8}$$

$$In \cong N \times (T \to Spi_\perp) \tag{3.9}$$

$$Out \cong N \times (T \times Spi_\perp + N \to \ldots N \to (T \times Spi_\perp)) \tag{3.10}$$

$$T \cong N + Sec + Pub + Sig + Pair \tag{3.11}$$

$$Sec \cong T \times N \tag{3.12}$$

$$Pub \cong T \times N \tag{3.13}$$

$$Sig \cong T \times N \tag{3.14}$$

$$Pair \cong T \times \ldots \times T \tag{3.15}$$

Where *Spi*, *In* and *Out* are the predomains of processes, input and output actions (including bound output actions), respectively. N is the predomain of names and T is the predomain of terms defined using N and the predomains of secret-key ciphertexts, SEC, public-key ciphertexts, PUB, digital signatures, SIG, and finite tuples, TUP. We do not include in the equations a predomain of variables, V, since we only deal with closed terms and processes. All the cryptographic terms are expressed as pairs consisting of a term (plaintext) and a

name (key). Finally, $\mathbb{P}(-)$ is Plotkin's powerdomain operation, with the single element domain, 1, representing deadlocked or terminated processes joined to the result of $\mathbb{P}(-)$ as in [8, Def. 3.4]. Lifting Spi, to give $Spi_\perp$, results in the domain of processes.

In addition to the presence of a predomain of terms, there are other differences between these equations and equations (3.1)–(3.3) of Section 3.2.3. First, input actions contain a functional element, $T \to Spi_\perp$, that is capable of being instantiated with any term, not just names. Second, free output actions are also equipped to send messages that are generic terms, T. Moreover, bound output actions are expressed as a finite number of functions, $N \to \ldots N \to (T \times Spi_\perp)$, that take as arguments names and yield other functions until eventually resulting in a pair representing the message of communication, T, and the residual process, $Spi_\perp$. This expresses the fact that bound outputs in the spi calculus can extrude the scope of multiple names, not just one name, as is the case with the π-calculus.

A number of continuous mappings leading into $Spi_\perp$ are also defined, to express the manner in which semantic elements are constructed [8, Def. 3.3]:

$$\emptyset : 1 \to Spi_\perp \tag{3.16}$$

$$\{\!|-|\!\} : (Spi_\perp + In + Out)_\perp \to Spi_\perp \tag{3.17}$$

$$\uplus : (Spi_\perp \times Spi_\perp) \to Spi_\perp \tag{3.18}$$

$$new : (N \to Spi_\perp) \to Spi_\perp \tag{3.19}$$

The description of these mappings is largely similar to their counterparts in the denotational model for the π-calculus (Section 3.2.3). The empty set map, $\emptyset$, constructs deadlocked or terminated processes. The singleton set map, $\{\!|-|\!\}$, constructs elements of the domain $Spi_\perp$ resulting from input, output and silent actions, as well as the bottom element, $\{\!|\perp|\!\}$. The standard powerdomain union, $\uplus$, represents non-deterministic choice, and finally, *new* is used to interpret the effects of restricting a single name to a process.

We construct a concrete solution for equations (3.8)–(3.15) that is based on domain theory. To specify elements of $Spi_\perp$, it is necessary to determine elements of all the other semantic domains. We start with the predomain of terms, T. Assuming $\mathcal{K}$ is the set underlying any domain, then elements $t \in \mathcal{K}(T)$ are defined in Figure 3.8. Due to the assumption that cryptographic functions are total over their arguments, which are of the appropriate type (e.g. keys are names), the predomain of terms, T, appears to be flat with discrete structure and no bottom elements. The disjoint union guarantees that cryptographic terms with similar elements are distinguished from each other as well as from the 2-element tuples (pairs). This is achieved by tagging the cryptographic terms with appropriate tags, *sec*, *pub* and *sig*, while leaving ordinary pairs without tags.

$a, b, c \ldots \in \mathcal{K}(N)$			Names
$t \in \mathcal{K}(T), k \in \mathcal{K}(N)$	$\Rightarrow$	$(t, k) \in \mathcal{K}(SEC)$	Secret-key ciphertexts
$t \in \mathcal{K}(T), k^+ \in \mathcal{K}(N)$	$\Rightarrow$	$(t, k^+) \in \mathcal{K}(PUB)$	Public-key ciphertexts
$t \in \mathcal{K}(T), k^- \in \mathcal{K}(N)$	$\Rightarrow$	$(t, k^-) \in \mathcal{K}(SIG)$	Digital signatures
$t_1, \ldots, t_n \in \mathcal{K}(T)$	$\Rightarrow$	$(t_1, \ldots, t_n) \in \mathcal{K}(TUP)$	Tuples
$\mathcal{K}(N + SEC + PUB + SIG + TUP) =$			Terms
$\{a \mid a \in N\} \cup \{sec(t, k) \mid (t, k) \in SEC\} \cup$			
$\{pub(t, k^+) \mid (t, k^+) \in PUB\} \cup$			
$\{sig(t, k^-) \mid (t, k^-) \in SIG\} \cup$			
$\{(t_1, \ldots, t_n) \mid (t_1, \ldots, t_n) \in TUP\}$			

Figure 3.8: Elements of the predomain of terms T.

Elements of In :		
$a \in \mathcal{K}(N), \lambda y.p \in \mathcal{K}(T \to Spi_\perp)$	$\Rightarrow$	$(a, \lambda y.p) \in \mathcal{K}(In)$
Elements of Out :		
$a \in \mathcal{K}(N), t \in \mathcal{K}(T), p \in \mathcal{K}(Spi_\perp)$	$\Rightarrow$	$(a, t, p) \in \mathcal{K}(Out)$
$a \in \mathcal{K}(N), \lambda n_1 \ldots \lambda n_m.(t, p) \in$		
$\mathcal{K}(N \to_1 \ldots N \to_m (T \times Spi_\perp))$	$\Rightarrow$	$(a, \lambda n_1 \ldots \lambda n_m.(t, p)) \in \mathcal{K}(Out)$
Elements of $Spi_\perp$:		
$\{\!\lfloor \perp \rfloor\!\} \in \mathcal{K}(Spi_\perp), \emptyset \in \mathcal{K}(Spi_\perp)$		
$p, q \in \mathcal{K}(Spi_\perp)$	$\Rightarrow$	$p \uplus q \in \mathcal{K}(Spi_\perp)$
$p \in \mathcal{K}(Spi_\perp)$	$\Rightarrow$	$\{\!\lfloor tau(p) \rfloor\!\} \in \mathcal{K}(Spi_\perp)$
$i \in \mathcal{K}(In)$	$\Rightarrow$	$\{\!\lfloor in(i) \rfloor\!\} \in \mathcal{K}(Spi_\perp)$
$o \in \mathcal{K}(Out)$	$\Rightarrow$	$\{\!\lfloor out(o) \rfloor\!\} \in \mathcal{K}(Spi_\perp)$

Figure 3.9: Elements of *In*, *Out*, and $Spi_\perp$.

From these semantic terms, the concrete elements of the predomains of *In* and *Out* are given leading to a solution of equations (3.8)–(3.10) specifying elements of the domain of processes, $Spi_\perp$. First, elements arising from $\emptyset$, $\{\!\lfloor - \rfloor\!\}$ and $\uplus$ are defined in Figure 3.9.

The effects of restricting a name to a process are interpreted by giving a concrete definition of *new* in terms of the simpler elements arising from $\emptyset$, $\uplus$ and $\{\!\lfloor - \rfloor\!\}$. This definition is

shown in Figure 3.10. The main difference in the definition of *new* from that of Figure 3.4 (Section 3.2.3) is that a bound output is allowed to have a finite number of bound names, not just a single name. Apart from terminating communications over fresh non-extruded channels and turning a free output action into a bound output action, restriction has no other effects and it is simply passed to the residue or distributed over the choice operator.

$$
\begin{array}{lll}
new(\lambda n.\emptyset) & = & \emptyset \\
new(\lambda n.\{\!|\bot|\!\}) & = & \{\!|\bot|\!\} \\
new(\lambda n.\{\!|in(a,\lambda x.p)|\!\}) & = & \\
\left\{\begin{array}{ll} \emptyset, & \text{if } a = n \\ \{\!|in(a,\lambda x.new(\lambda n.p))|\!\}, & \text{otherwise} \end{array}\right. & & \\
 & & \\
new(\lambda n.\{\!|out(a,t,p)|\!\}) & = & \\
\left\{\begin{array}{ll} \emptyset, & \text{if } a = n \\ \{\!|out(a,\lambda n.(t,p))|\!\}, & \text{if } n \in n(t) \text{ and } n \neq a \\ \{\!|out(a,t,new(\lambda n.p))|\!\}, & \text{otherwise} \end{array}\right. & & \\
 & & \\
new(\lambda n.\{\!|out(a,\lambda m_1 \ldots \lambda m_k.(t,p))|\!\}) & = & \\
\left\{\begin{array}{ll} \emptyset, & \text{if } a = n \\ \{\!|out(a,\lambda n.\lambda m_1 \ldots \lambda m_k.(t,p))|\!\}, & \text{if } n \in n(t) \text{ and } n \neq a \\ \{\!|out(a,\lambda m_1 \ldots \lambda m_k.(t,new(\lambda n.p)))|\!\}, & \text{otherwise} \end{array}\right. & & \\
 & & \\
new(\lambda n.\{\!|tau(p)|\!\}) & = & \{\!|tau(new(\lambda n.p))|\!\} \\
new(\lambda n.(p_1 \uplus p_2)) & = & new(\lambda n.p_1) \uplus new(\lambda n.p_2)
\end{array}
$$

Figure 3.10: The definition of *new* for the spi calculus.

The denotational semantics for the spi calculus can now be given as a semantic function $\mathcal{S}^{spi}(\!|P|\!)\ \rho\ \phi_{\mathcal{S}} \in Spi_{\perp}$ defined by the set of rules of Figure 3.11. The multiset, ρ, is used to hold processes composed in parallel with the analysed process, where rule ($\mathcal{R}^{spi}0$) is used to interpret the contents of ρ. The environment, $\phi_{\mathcal{S}} : V \rightarrow T_{\perp}$, where V is the predomain of variables, captures any term substitutions that occur in the semantics. The special function, $\varphi_{\mathcal{S}} : (V \rightarrow T_{\perp}) \times Term \rightarrow T$, returns the semantic value of a term, given substitutions recorded by $\phi_{\mathcal{S}}$:

$$
\begin{array}{lll}
(\mathcal{S}^{spi}1) & \mathcal{S}^{spi}(\llbracket \mathbf{0} \rrbracket)\ \rho\ \phi_{\mathcal{S}} & = \ \emptyset \\
(\mathcal{S}^{spi}2) & \mathcal{S}^{spi}(\llbracket M(y).P \rrbracket)\ \rho\ \phi_{\mathcal{S}} & = \ \{\!|in(\varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, M), \lambda y.\mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_{\rho} \rrbracket)\ \phi_{\mathcal{S}})|\!\} \\
 & & \quad \text{where, } \varphi(\phi_{\mathcal{E}}, M) \in N \\
(\mathcal{S}^{spi}3) & \mathcal{S}^{spi}(\llbracket \overline{M}\langle L \rangle.P \rrbracket)\ \rho\ \phi_{\mathcal{S}} & = \\
 & \biguplus_{M'(z).P' \in \rho} \{\!|tau(\mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_{\rho} \uplus_{\rho} \rho[P'/M'(z).P'] \rrbracket)\ \phi_{\mathcal{S}}[z \mapsto \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, L)])|\!\} \\
 & \quad \uplus \{\!|out(\varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, M), \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, L), \mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_{\rho} \rrbracket)\ \phi_{\mathcal{S}})|\!\} \\
 & & \quad \text{where, } \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, M) = \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, M') \in N \\
(\mathcal{S}^{spi}4) & \mathcal{S}^{spi}(\llbracket (\nu a)P \rrbracket)\ \rho\ \phi_{\mathcal{S}} & = \ new(\lambda a.\mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_{\rho} \uplus_{\rho} \rho \rrbracket)\ \phi_{\mathcal{S}}) \\
(\mathcal{S}^{spi}5) & \mathcal{S}^{spi}(\llbracket P \mid Q \rrbracket)\ \rho\ \phi_{\mathcal{S}} & = \ \mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_{\rho} \uplus_{\rho} \{\!|Q|\!\}_{\rho} \uplus_{\rho} \rho \rrbracket)\ \phi_{\mathcal{S}} \\
(\mathcal{S}^{spi}6) & \mathcal{S}^{spi}(\llbracket !P \rrbracket)\ \rho\ \phi_{\mathcal{S}} & = \ \mathcal{F}^{spi}(-1)
\end{array}
$$

where, $\mathcal{F}^{spi}(n) = \mathit{let}\ p_1 = \mathcal{S}^{spi}(\llbracket \prod_{i=1}^{n} P[bnv_i(P)/bnv(P)] \rrbracket)\ \rho\ \phi_{\mathcal{S}}\ \mathit{in}$

$\mathit{let}\ p_2 = \mathcal{S}^{spi}(\llbracket \prod_{i=1}^{n+2} P[bnv_i(P)/bnv(P)] \rrbracket)\ \rho\ \phi_{\mathcal{S}} =\ \mathit{in}$

$\mathit{if}\ p_1 = p_2\ \mathit{then}\ p_1\ \mathit{else}\ \mathcal{F}^{spi}(n+1)$

and, $bnv_i(P) = \{x_i \mid x \in bnv(P)\}$

$(\mathcal{S}^{spi}7)$ $\mathcal{S}^{spi}(\llbracket \mathit{if}\ M = L\ \mathit{then}\ P\ \mathit{else}\ Q \rrbracket)\ \rho\ \phi_{\mathcal{S}} =$

$$
\begin{cases}
\mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_{\rho} \uplus_{\rho} \rho \rrbracket)\ \phi_{\mathcal{S}}, & \text{if } \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, M) = \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, L) \\
\mathcal{R}^{spi}(\llbracket \{\!|Q|\!\}_{\rho} \uplus_{\rho} \rho \rrbracket)\ \phi_{\mathcal{S}}, & \text{otherwise}
\end{cases}
$$

$(\mathcal{S}^{spi}8)$ $\mathcal{S}^{spi}(\llbracket \mathit{let}\ (x_1, \ldots, x_n) = M\ \mathit{in}\ P\ \mathit{else}\ Q \rrbracket)\ \rho\ \phi_{\mathcal{S}} =$

$$
\begin{cases}
\mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_{\rho} \uplus_{\rho} \rho \rrbracket)\ \phi'_{\mathcal{S}}, & \text{if } \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, M) = (t_1, \ldots, t_n) \\
\text{where, } \phi'_{\mathcal{S}} = \phi_{\mathcal{S}}[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n] & \\
\mathcal{R}^{spi}(\llbracket \{\!|Q|\!\}_{\rho} \uplus_{\rho} \rho \rrbracket)\ \phi_{\mathcal{S}}, & \text{otherwise}
\end{cases}
$$

$(\mathcal{S}^{spi}9)$ $\mathcal{S}^{spi}(\llbracket \mathit{case}\ L\ \mathit{of}\ \{x\}_N\ \mathit{in}\ P\ \mathit{else}\ Q \rrbracket)\ \rho\ \phi_{\mathcal{S}} =$

$$
\begin{cases}
\mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_{\rho} \uplus_{\rho} \rho \rrbracket)\ \phi'_{\mathcal{S}}, & \text{if } \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, L) = sec(t, k) \text{ and } \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, N) = k \\
\text{where, } \phi'_{\mathcal{S}} = \phi_{\mathcal{S}}[x \mapsto t] & \\
\mathcal{R}^{spi}(\llbracket \{\!|Q|\!\}_{\rho} \uplus_{\rho} \rho \rrbracket)\ \phi_{\mathcal{S}}, & \text{otherwise}
\end{cases}
$$

$(\mathcal{S}^{spi}10)$ $\mathcal{S}^{spi}(\llbracket \mathit{case}\ L\ \mathit{of}\ \{\!|x|\!\}_N\ \mathit{in}\ P\ \mathit{else}\ Q \rrbracket)\ \rho\ \phi_{\mathcal{S}} =$

$$
\begin{cases}
\mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_{\rho} \uplus_{\rho} \rho \rrbracket)\ \phi'_{\mathcal{S}}, & \text{if } \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, L) = pub(t, k^{+}) \text{ and } \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, N) = k^{-} \\
\text{where, } \phi'_{\mathcal{S}} = \phi_{\mathcal{S}}[x \mapsto t] & \\
\mathcal{R}^{spi}(\llbracket \{\!|Q|\!\}_{\rho} \uplus_{\rho} \rho \rrbracket)\ \phi_{\mathcal{S}}, & \text{otherwise}
\end{cases}
$$

$(\mathcal{S}^{spi}11)$ $\mathcal{S}^{spi}(\llbracket \mathit{case}\ L\ \mathit{of}\ [\!|x|\!]_N\ \mathit{in}\ P\ \mathit{else}\ Q \rrbracket)\ \rho\ \phi_{\mathcal{S}} =$

$$
\begin{cases}
\mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_{\rho} \uplus_{\rho} \rho \rrbracket)\ \phi'_{\mathcal{S}}, & \text{if } \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, L) = sig(t, k^{-}) \text{ and } \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, N) = k^{+} \\
\text{where, } \phi'_{\mathcal{S}} = \phi_{\mathcal{S}}[x \mapsto t] & \\
\mathcal{R}^{spi}(\llbracket \{\!|Q|\!\}_{\rho} \uplus_{\rho} \rho \rrbracket)\ \phi_{\mathcal{S}}, & \text{otherwise}
\end{cases}
$$

$$
(\mathcal{R}^{spi}0) \quad \mathcal{R}^{spi}(\llbracket \rho \rrbracket)\ \phi_{\mathcal{S}} = \biguplus_{P \in \rho} \mathcal{S}^{spi}(\llbracket P \rrbracket)\ (\rho \backslash \{\!|P|\!\}_{\rho})\ \phi_{\mathcal{S}}
$$

Figure 3.11: The denotational semantics of the spi calculus.

$$\forall \phi_{\mathcal{S}}, M : \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, M) = \begin{cases} \phi_{\mathcal{S}}(M), & \text{if } M \in V \\ M, & \text{if } M \in N \\ sec(\varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, M'), \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, N)), & \text{if } M = \{M'\}_N \\ pub(\varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, M'), \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, N)), & \text{if } M = \{\!|M'|\!\}_N \\ sig(\varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, M'), \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, N)), & \text{if } M = [\!\{M'\}\!]_N \\ (\varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, M_1), \ldots, \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, M_n)), & \text{if } M = (M_1, \ldots, M_n) \end{cases}$$

Note that since we only deal with closed terms, the case where $M = x \in V$ and $\phi_{\mathcal{S}}(x) = \perp$ will never be encountered (open terms).

The description of rules ($\mathcal{S}^{spi}1$)–($\mathcal{S}^{spi}11$) is as follows. Rule ($\mathcal{S}^{spi}1$) interprets the meaning of a null process as the empty set mapping, $\emptyset$. Rules ($\mathcal{S}^{spi}2$) and ($\mathcal{S}^{spi}3$) deal with processes guarded with input and output actions, respectively. The rule for output actions, ($\mathcal{S}^{spi}3$), considers communications between the output channel and appropriate input channels guarding processes in ρ. The $\phi_{\mathcal{S}}$ is updated appropriately with semantic elements. Rule ($\mathcal{S}^{spi}4$) uses the *new* mapping to interpret the meaning of a restriction. Rule ($\mathcal{S}^{spi}5$) interprets directly parallel composition by the addition of the parallel subprocesses to ρ.

Finally, rule ($\mathcal{S}^{spi}6$) interprets a replicated process, $!P$, by calculating the higher-order function, $\mathcal{F}^{spi} : \mathbb{N} \rightarrow Spi_{\perp}$, starting from the bottom number, $n = -1$, which computes the bottom semantic element, $\mathcal{S}^{spi}(\![\prod_{i=1}^{-1} P[bnv_i(P)/bnv(P)]]\!)\ \rho\ \phi_{\mathcal{S}} = \{\!|\perp|\!\}$. The value of n is incremented in each iteration until the condition of the *if*-statement holds true, or in other words, the least fixed-point is reached. Due to the fact that the semantic domain, $Spi_{\perp}$, is infinite, this calculation may not terminate within finite limits. The rule also uses the labelling mechanism to rename all the bound variables and names of the spawned processes by subscripting those variables and names with a number signifying each spawned copy. Since this renaming of bound variables and names is, in fact, α-conversion, the resulting process on the right side of the rule is structurally equivalent to a subprocess of the replication on the left side. This preserves the compositionality of the denotational semantics since the α-converted process is still a subprocess of the replication.

The rest of the rules rely on the meaning of terms as held by the $\phi_{\mathcal{S}}$ environment before resolving the analysed process. In rule ($\mathcal{S}^{spi}7$), the meaning of two terms is compared, and depending on the result, one of two processes is chosen and added to ρ. Rules ($\mathcal{S}^{spi}8$)–($\mathcal{S}^{spi}11$) deal with tuple splitting and cryptographic processes. The result of the tuple splitting and cryptographic operations are used to update the $\phi_{\mathcal{S}}$ with the appropriate semantic terms. A residual process, P, signifying the success of the operation is also added to ρ. In case an operation fails, an alternative process, Q, is chosen and added to ρ. We can now state the following theorem.

Theorem 2 (Soundness and Adequacy of $Spi_\perp$ w.r.t. transitions)
The interpretation of processes in $Spi_\perp$ is sound and adequate with respect to transitions in the spi calculus.

Proof. The soundness property relies on the ability of semantic elements p to match transitions in the operational model. Hence:

$P \xrightarrow{\overline{m}\langle N\rangle} Q \;\Rightarrow\; out(m, \varphi_{\mathcal{S}}(N, \phi_{\mathcal{S}}), \mathcal{S}^{spi}(\llbracket Q \rrbracket)\ \rho\ \phi_{\mathcal{S}}) \in \mathcal{S}^{spi}(\llbracket P \rrbracket)\ \rho\ \phi_{\mathcal{S}}$

$P \xrightarrow{(\nu\, n_1,\ldots,\nu\, n_k)\overline{m}\langle N\rangle} Q \;\Rightarrow\; out(m, \lambda n_1 \ldots \lambda n_k.(\varphi_{\mathcal{S}}(N, \phi_{\mathcal{S}}), \mathcal{S}^{spi}(\llbracket Q \rrbracket)\ \rho\ \phi_{\mathcal{S}})) \in \mathcal{S}^{spi}(\llbracket P \rrbracket)\ \rho\ \phi_{\mathcal{S}}$

$P \xrightarrow{m(x)} Q \;\Rightarrow\; in(m, \lambda x.\mathcal{S}^{spi}(\llbracket Q \rrbracket)\ \rho\ \phi_{\mathcal{S}}) \in \mathcal{S}^{spi}(\llbracket P \rrbracket)\ \rho\ \phi_{\mathcal{S}}$

$P \xrightarrow{\tau} Q \;\Rightarrow\; tau(\mathcal{S}^{spi}(\llbracket Q \rrbracket)\ \rho\ \phi_{\mathcal{S}}) \in \mathcal{S}^{spi}(\llbracket P \rrbracket)\ \rho\ \phi_{\mathcal{S}}$

On the other hand, adequacy requires that the semantic transitions must be mapped correctly to the operational model. Hence:

$out(m, t, q) \in \mathcal{S}^{spi}(\llbracket P \rrbracket)\ \rho\ \phi_{\mathcal{S}} \Rightarrow \exists Q : P \xrightarrow{\overline{m}\langle N\rangle} Q \ \wedge\ \varphi_{\mathcal{S}}(N, \phi_{\mathcal{S}}) = t \ \wedge\ \mathcal{S}^{spi}(\llbracket Q \rrbracket)\ \rho\ \phi_{\mathcal{S}} = q$

$out(m, \lambda n_1 \ldots \lambda n_k.(t, q)) \in \mathcal{S}^{spi}(\llbracket P \rrbracket)\ \rho\ \phi_{\mathcal{S}} \Rightarrow \exists Q : P \xrightarrow{(\nu\, n_1,\ldots,\nu\, n_k)\overline{m}\langle N\rangle} Q \ \wedge\ \varphi_{\mathcal{S}}(N, \phi_{\mathcal{S}}) = t \ \wedge\ \mathcal{S}^{spi}(\llbracket Q \rrbracket)\ \rho\ \phi_{\mathcal{S}} = q$

$in(m, \lambda x.q) \in \mathcal{S}^{spi}(\llbracket P \rrbracket)\ \rho\ \phi_{\mathcal{S}} \Rightarrow \exists Q : P \xrightarrow{m(x)} Q \ \wedge\ \mathcal{S}^{spi}(\llbracket Q \rrbracket)\ \rho\ \phi_{\mathcal{S}} = q$

$tau(q) \in \mathcal{S}^{spi}(\llbracket P \rrbracket)\ \rho\ \phi_{\mathcal{S}} \;\Rightarrow\; \exists Q : P \xrightarrow{\tau} Q \ \wedge\ \mathcal{S}^{spi}(\llbracket Q \rrbracket)\ \rho\ \phi_{\mathcal{S}} = q$

The proof of soundness proceeds by induction on the structure of the denotational semantics, whereas the proof of adequacy requires a formal approximation relation, $p \lhd P$, between elements $p \in Spi_\perp$ and processes $P \in \mathcal{P}$. □

3.4 Conclusion

In this section, we have given an overview of the syntax and structural operational semantics of the π-calculus and its cryptographic extension, the spi calculus. The main contribution of this chapter has been in constructing a denotational semantics for the π-calculus based on a domain-theoretic solution of the predomain equations given by Stark in [121]. We have also modified these equations to be able to model processes in the spi calculus by the addition of an extra predomain for complex terms, T, to be able to model ciphertexts, digital signatures and tuples. We also allow multiple-name extrusions to occur in bound output actions.

The standard denotational semantics constitutes a concrete basis for building the abstract interpretation in the next chapter. In particular, the semantic domains will be extended to include elements that express term substitutions occurring in processes as a result of communications (π-calculus and spi calculus) and cryptographic operations (spi calculus).

Chapter 4

Abstract Interpretation

4.1 Introduction

After defining the standard denotational semantics of the π-calculus and the spi calculus in the previous chapter, we use these semantics as the basis for constructing abstract interpretations for the two languages in this chapter. The approach we follow is based on extending the standard semantics of a language to obtain a correct non-standard meaning that captures the property of interest, in this case, *term substitution*. Our definition of the correctness property of the non-standard semantics with respect to the standard semantics is that the standard meaning of a process be extractable from its non-standard meaning.

The next step involves the introduction of a suitable abstraction (approximation) that restricts the semantic domain to a finite size, thereby allowing the termination of any least fixed-point calculations. This abstraction is required to be safe with respect to the concrete non-standard semantics; a term substitution occurring in the concrete semantics will necessarily be captured in an abstract manner by the abstract semantics. However, due to the imprecise nature of abstractions, false positives can appear in the final results. For example, in $\overline{x}\langle s\rangle \mid xx(y) \mid \overline{z}\langle w\rangle \mid zz(u)$, if, $x = xx = t_1$ and $z = zz = t_2$, then we have in the standard semantics that s can only substitute y, and w can only substitute u. However, abstracting the above values, such that $x = xx \in \{t_1, t_2\}$ and $z = zz \in \{t_1, t_2\}$, then s may substitute y or u, and similarly, w may substitute y or u. This is because the abstraction introduces the scenario where $x = zz$ and $z = xx$, which is impossible in the concrete semantics.

Finally, we give specifications of examples of protocols specified in the languages of the π-calculus and the spi calculus and apply the abstract interpretation to these systems to capture the property of term substitutions.

4.2 The π-calculus

As we described in the previous chapter, the main notion of behaviour in the π-calculus is name passing, where processes communicate by exchanging names over channels allowing the overall topology of the network to evolve over time. Analysing this behaviour offers valuable information about the different program properties in general (e.g. control and data flow information), and about security properties in particular (e.g. processes obtaining private information and processes obtaining inauthentic messages).

The standard semantic domain of the previous chapter was defined over the process domain, $Pi_\perp$. The information contained in this domain is not sufficient to reason about name substitutions. Therefore, we extend the domain to include mappings from the set of input parameters of a specification to sets of names representing possible instantiations of those parameters during process communications. However, due to the possibility of processes having infinite traces in $Pi_\perp$ and their ability to create infinite numbers of instances of bound names, an abstraction is required to safely ensure the termination of the semantics. The main idea underlying this abstraction is to place an upper limit on the number of copies of each name that can appear in the abstract meaning of a process. An earlier version of the work presented in this section can be found in [15].

4.2.1 Non-standard Semantics

To trace name substitutions during the evolution of processes in the π-calculus, we define the special environment $\phi_\mathcal{E} : N \rightarrow \wp(N)$, which maps each name of a process to a set of names that may instantiate that name during runtime. The null environment, $\phi_{\mathcal{E}0}$, maps every name to the empty set:

$$\forall x \in N : \phi_{\mathcal{E}0}(x) = \{\}$$

When computing the meaning of a process, P, elements of the set $bn(P)$, representing input parameters, will obtain values in $\phi_\mathcal{E}$ whenever communications take place. For example, if a message, z, is received over a channel, x, in an input action, $x(y).P$, then z will be added to the set of names given by $\phi_\mathcal{E}(y)$. The fact that the semantics is precise, i.e. every instance of a bound name is distinguished from every other instance in the semantics, means that the set $\phi_\mathcal{E}(y)$ will be at most a singleton set per choice of control flow.

As a result of these substitutions, a domain $D_\perp = N \rightarrow \wp(N)$ is formed with the following partial ordering relation, based on subset inclusion:

$$\forall \phi_{\mathcal{E}1}, \phi_{\mathcal{E}2} \in D_\perp : \phi_{\mathcal{E}1} \sqsubseteq_{D_\perp} \phi_{\mathcal{E}2} \;\Leftrightarrow\; \forall x \in N : \phi_{\mathcal{E}1}(x) \subseteq \phi_{\mathcal{E}2}(x)$$

with the bottom element of $D_\perp$ being the null environment, $\phi_{\mathcal{E}0}$. The union of $\phi_\mathcal{E}$ environments, $\cup_\phi$, can be defined in terms of the standard union, $\cup$, as follows:

$$\forall \phi_{\mathcal{E}1}, \phi_{\mathcal{E}2} \in D_\perp, x \in N : (\phi_{\mathcal{E}1} \cup_\phi \phi_{\mathcal{E}2})(x) = \phi_{\mathcal{E}1}(x) \cup \phi_{\mathcal{E}2}(x)$$

The non-standard semantic domain is then the product $Pi_\perp \times D_\perp$ ordered pointwise on its elements, with a bottom element, $(\perp_{Pi_\perp}, \perp_{D_\perp})$. Based on the domain $Pi_\perp \times D_\perp$, the non-standard semantics of the π-calculus can be defined using the function $\mathcal{E}^\pi(\![P]\!)\ \rho\ \phi_\mathcal{E} \in Pi_\perp \times D_\perp$, shown in Figure 4.1.

The semantics utilises a multiset, ρ, to hold all processes in parallel with the analysed process. The contents of this multiset are interpreted in rule ($\mathcal{R}^\pi 0$), which uses the choice mapping, $\uplus$, to group the $p \in Pi_\perp$ elements and the union of environments, $\cup_\phi$, to group the $\phi_\mathcal{E} \in D_\perp$ elements in the resulting semantic pair. The semantics also defines the special function, $\varphi_\mathcal{E} : (N \to \wp(N)) \times \mathcal{N} \to N$, which retrieves the value of a name, x, given earlier substitutions recorded by $\phi_\mathcal{E}$:

$$\forall \phi_\mathcal{E} \in D_\perp, x \in N : \varphi_\mathcal{E}(\phi_\mathcal{E}, x) = \begin{cases} z, & \text{if } \phi_\mathcal{E}(x) = \{z\} \\ x, & \text{otherwise} \end{cases}$$

The rest of the rules ($\mathcal{E}^\pi 1$)–($\mathcal{E}^\pi 8$) are described informally as follows. The interpretation of null processes in rule ($\mathcal{E}^\pi 1$) is given as a pair expressing the empty set map and an unchanged $\phi_\mathcal{E}$ environment. Processes guarded by input and output actions are treated in the following rules, ($\mathcal{E}^\pi 2$) and ($\mathcal{E}^\pi 3$), respectively. Possible communications between output actions and the matching input actions are treated in rule ($\mathcal{E}^\pi 3$), where the value of $\phi_\mathcal{E}$ is updated accordingly. As a result of the substitution in $\phi_\mathcal{E}$, the input parameter is removed from the residue. Along with the fact that bound names are maintained distinct (rule ($\mathcal{E}^\pi 8$)), this results in the possibility of having a single substitution only per choice of control flow. To indicate the presence of a communication, the silent action, *tau*, is added to the process.

Rule ($\mathcal{E}^\pi 4$) deals with the conditional process comparing the values of two names. One process or another is chosen to be added to ρ depending on whether the equality of the two names holds or not. Rule ($\mathcal{E}^\pi 5$) deals with the summation of processes, where elements of the semantic pairs resulting from the non-standard interpretation of the two subprocesses are related pointwise using the choice map, $\uplus$, and the union of environments, $\cup_\phi$. Rule ($\mathcal{E}^\pi 6$) joins two parallel processes to the rest of parallel processes supplied by the ρ multiset. In rule ($\mathcal{E}^\pi 7$), the interpretation of restriction is carried out using the *new* map on the process element of the resulting semantic pair. The $\phi_\mathcal{E}$ element remains unchanged (since internal communications are preserved under restrictions).

$$
\begin{array}{lll}
(\mathcal{E}^\pi 1) & \mathcal{E}^\pi(\llbracket \mathbf{0} \rrbracket)\ \rho\ \phi_\mathcal{E} & = (\emptyset, \phi_\mathcal{E}) \\
(\mathcal{E}^\pi 2) & \mathcal{E}^\pi(\llbracket x(y).P \rrbracket)\ \rho\ \phi_\mathcal{E} & = (\{\!|in(\varphi_\mathcal{E}(\phi_\mathcal{E}, x), \lambda y.p')|\!\}, \phi_\mathcal{E}) \\
 & & \text{where, } (p', \phi'_\mathcal{E}) = \mathcal{R}^\pi(\llbracket \{\!|P|\!\}_\rho \rrbracket)\ \phi_\mathcal{E} \\
(\mathcal{E}^\pi 3) & \mathcal{E}^\pi(\llbracket \overline{x}\langle y\rangle.P \rrbracket)\ \rho\ \phi_\mathcal{E} & = \\
 & (\biguplus_{x'(z).P' \in \rho} \{\!|tau(p')|\!\} \uplus \{\!|out(\varphi_\mathcal{E}(\phi_\mathcal{E}, x), \varphi_\mathcal{E}(\phi_\mathcal{E}, y), p'')|\!\}, \bigcup_{x'(z).P' \in \rho}^{\phi} \phi'_\mathcal{E} \cup_\phi \phi_\mathcal{E}) & \\
 & \text{if, } \varphi_\mathcal{E}(\phi_\mathcal{E}, x) = \varphi_\mathcal{E}(\phi_\mathcal{E}, x') & \\
 & \text{where, } (p', \phi'_\mathcal{E}) = \mathcal{R}^\pi(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \rho[P'/x'(z).P'] \rrbracket)\ \phi_\mathcal{E}[z \mapsto \{\varphi_\mathcal{E}(\phi_\mathcal{E}, y)\}] & \\
 & \text{and, } (p'', \phi''_\mathcal{E}) = \mathcal{R}^\pi(\llbracket \{\!|P|\!\}_\rho \rrbracket)\ \phi_\mathcal{E} & \\
(\mathcal{E}^\pi 4) & \mathcal{E}^\pi(\llbracket \mathit{if}\ x = y\ \mathit{then}\ P\ \mathit{else}\ Q \rrbracket)\ \rho\ \phi_\mathcal{E} = & \\
 & \left\{ \begin{array}{l} \mathcal{R}^\pi(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi_\mathcal{E}, \quad \text{if } \varphi_\mathcal{E}(\phi_\mathcal{E}, x) = \varphi_\mathcal{E}(\phi_\mathcal{E}, y) \\ \mathcal{R}^\pi(\llbracket \{\!|Q|\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi_\mathcal{E}, \quad \text{otherwise} \end{array} \right. & \\
(\mathcal{E}^\pi 5) & \mathcal{E}^\pi(\llbracket P + Q \rrbracket)\ \rho\ \phi_\mathcal{E} & = ((p' \uplus p''), (\phi'_\mathcal{E} \cup_\phi \phi''_\mathcal{E})) \\
 & & \text{where, } (p', \phi'_\mathcal{E}) = \mathcal{R}^\pi(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi_\mathcal{E} \\
 & & \text{and, } (p'', \phi''_\mathcal{E}) = \mathcal{R}^\pi(\llbracket \{\!|Q|\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi_\mathcal{E} \\
(\mathcal{E}^\pi 6) & \mathcal{E}^\pi(\llbracket P \mid Q \rrbracket)\ \rho\ \phi_\mathcal{E} & = \mathcal{R}^\pi(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \{\!|Q|\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi_\mathcal{E} \\
(\mathcal{E}^\pi 7) & \mathcal{E}^\pi(\llbracket (\nu x)P \rrbracket)\ \rho\ \phi_\mathcal{E} & = (new(\lambda x.p'), \phi'_\mathcal{E}) \\
 & & \text{where, } (p', \phi'_\mathcal{E}) = \mathcal{R}^\pi(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi_\mathcal{E} \\
(\mathcal{E}^\pi 8) & \mathcal{E}^\pi(\llbracket !P \rrbracket)\ \rho\ \phi_\mathcal{E} & = \mathcal{F}^\pi(-1) \\
 & \text{where, } \mathcal{F}^\pi(n) = \mathit{let}\ v_1 = \mathcal{E}^\pi(\llbracket \prod_{i=1}^{n} P[bn_i(P)/bn(P)] \rrbracket)\ \rho\ \phi_\mathcal{E}\ \mathit{in} & \\
 & \qquad \mathit{let}\ v_2 = \mathcal{E}^\pi(\llbracket \prod_{i=1}^{n+2} P[bn_i(P)/bn(P)] \rrbracket)\ \rho\ \phi_\mathcal{E}\ \mathit{in} & \\
 & \qquad \mathit{if}\ v_1 = v_2\ \mathit{then}\ v_1\ \mathit{else}\ \mathcal{F}^\pi(n+1) & \\
 & \text{and, } bn_i(P) = \{x_i \mid x \in bn(P)\} & \\
(\mathcal{R}^\pi 0) & \mathcal{R}^\pi(\llbracket \rho \rrbracket)\ \phi_\mathcal{E} & = (\biguplus_{P \in \rho} p', \bigcup_{P \in \rho}^{\phi} \phi'_\mathcal{E}) \\
 & & \text{where, } (p', \phi'_\mathcal{E}) = \mathcal{E}^\pi(\llbracket P \rrbracket)\ (\rho \backslash \{\!|P|\!\}_\rho)\ \phi_\mathcal{E}
\end{array}
$$

Figure 4.1: The non-standard semantics of the π-calculus.

The rule for replication, ($\mathcal{E}^\pi 8$), computes a special function, $\mathcal{F}^\pi : \mathbb{N} \rightarrow Pi_\perp \times D_\perp$, starting at $n = -1$, which contains the bottom element, $(\perp_{Pi_\perp}, \perp_{D_\perp})$, and continues by incrementing n, until it reaches a least fixed-point, at which the meaning $v_1 \in Pi_\perp \times D_\perp$ cannot be increased any further. Due to the infinite nature of the non-standard semantic domain, computing this fixed point is not guaranteed to terminate. Later, we discuss a possible abstraction that maintains the computability of the least fixed-point while sacrificing the precision of the semantics. Also, any spawned copies of $!P$ are α-converted to distinguish between the new copies of bound names. This α-conversion does not affect the compositionality of the semantics since the process on the right side of the rule is still a subprocess of the left side of the rule (up to α-conversion).

The following theorem states that the non-standard semantics of the π-calculus is *correct* with respect to the standard denotational semantics introduced in the previous chapter.

Theorem 3 (Correctness of the Non-Standard Semantics)
$\forall P \in \mathcal{P} : (\mathcal{S}^\pi(\llbracket P \rrbracket)\ \rho\ \phi_\mathcal{S} = p)\ \wedge\ (\mathcal{E}^\pi(\llbracket P \rrbracket)\ \rho\ \phi_\mathcal{E} = (p', \phi'_\mathcal{E}))\ \ \Rightarrow\ \ p = p'$

Proof. The proof is by induction over the structure of the rules of the standard and non-standard semantics. □

Intuitively, the theorem states that for any process, P, it is possible to extract the standard meaning of P, as interpreted by $\mathcal{S}^\pi(\llbracket P \rrbracket)\ \rho\ \phi_\mathcal{S}$, from its non-standard meaning, as interpreted by $\mathcal{E}^\pi(\llbracket P \rrbracket)\ \rho\ \phi_\mathcal{E}$. In other words, the former is equivalent to the first element of the pair generated by the latter.

4.2.2 Abstract Semantics

As we mentioned earlier in the previous section, the non-standard semantics of the π-calculus contains least fixed-point calculations that may not be computable (in the rule for replication). To obtain a computable semantics, whose termination is guaranteed, a safe abstraction is required to remove the source of infinite growth in the semantic domain $Pi_\perp \times D_\perp$. For example, consider the following system:

$$!((\nu z)\overline{x}\langle z\rangle.P \ \mid\ x(y).Q)$$

As a result of the infinite communications within this system, its non-standard semantics will yield (as part of its meaning) an infinite trace of silent actions $tau(tau(tau(\ldots$ as well as a $\phi_\mathcal{E}$ environment that maps every instance y_j to $\{z_j\}$, for $j \in \mathbb{N}$.

To reduce the semantic domain to a finite level, we first assume a finite predomain of tags, *Tag*, ranged over by t, t' etc. Given a process, P, we place distinct tags on all

messages of output actions of P. For example, tagging $!(\nu\, x)\overline{y}\langle x\rangle.\overline{y}\langle y\rangle.y(z).\overline{y}\langle z\rangle$ results in $!(\nu\, x)\overline{y}\langle x^t\rangle.\overline{y}\langle y^{t'}\rangle.y(z).\overline{y}\langle z^{t''}\rangle$. Copies of tags are renamed by subscripting them with the number of their copy. For example, the replication above can spawn two copies:
$!(\nu\, x)\overline{y}\langle x^t\rangle.\overline{y}\langle y^{t'}\rangle.y(z).\overline{y}\langle z^{t''}\rangle \mid$
$(\nu\, x_1)\overline{y}\langle x_1^{t_1}\rangle.\overline{y}\langle y^{t_1'}\rangle.y(z_1).\overline{y}\langle z_1^{t_1''}\rangle \mid (\nu\, x_2)\overline{y}\langle x_2^{t_2}\rangle.\overline{y}\langle y^{t_2'}\rangle.y(z_2).\overline{y}\langle z_2^{t_2''}\rangle$

Furthermore, we define the function, $value\_of(\{t_1,\ldots,t_n\}) = \{x_1,\ldots,x_m\}$, which can be applied to a set of tags, $\{t_1,\ldots,t_n\}$, returning a set of names, $\{x_1,\ldots,x_m\}$, tagged by those tags. Hence, for the above process, we have that:

$$value\_of(\{t, t_1, t_2, t', t_1', t_2', t'', t_1'', t_2''\}) = \{x, x_1, x_2, y, z, z_1, z_2\}$$

Next, we define the following abstraction for names and tags, which places an upper limit, k, on the total number of copies of names and tags that can be captured. In general, selecting k is non-decidable and relies, to a great extent, on user's experience and the specific program being analysed.

Definition 1 *Define the abstraction function,* $\alpha_k : \mathbb{N} \times (N + Tag) \rightarrow (N^\sharp + Tag^\sharp)$*, by:*

$$\forall x \in (N + Tag), i, k \in \mathbb{N} : \alpha_k(x) = \begin{cases} x_k, & \text{if } x = x_i \text{ and } i > k \\ x, & \text{otherwise} \end{cases}$$

Note that $N^\sharp = N \backslash \{x_j \mid j > k\}$ and $Tag^\sharp = Tag \backslash \{t_j \mid j > k\}$. Using abstracted names and tags, we can define the abstract environment, $\phi_{\mathcal{A}} : N^\sharp \rightarrow \wp(Tag^\sharp)$, mapping abstracted input parameters to sets of abstracted tags whose names can possibly replace each input parameter in the semantics. Due to the imprecise nature of the abstract semantics, $\phi_{\mathcal{A}}(x)$ may be larger than a singleton set. This imprecision results from the inability to distinguish between the different copies of input parameters and tags beyond the k^{th} copy.

The value of the integer constraint is dependent on the definition of the property we are interested in and will determine the precision of the results of the abstract interpretation. A *uniform* analysis (i.e. $k = 1$) is incapable of distinguishing between the different copies of replicated processes and it is suitable for properties that remain uniform across the range of copies of names. On the other hand, a flexible *non-uniform* analysis (i.e. $k > 1$) is suitable for properties that require more precision, since the different copies of processes spawned from a replication can be distinguished (by distinguishing their bound names and tags).

The abstract semantic domain, $D_\perp^\sharp = N^\sharp \rightarrow \wp(Tag^\sharp)$, is defined as the domain of $\phi_{\mathcal{A}}$ environments ordered by subset inclusion (with $\cup_\phi$ defined as in the previous section):

$$\forall \phi_{\mathcal{A}1}, \phi_{\mathcal{A}2} \in D_\perp^\sharp : \phi_{\mathcal{A}1} \sqsubseteq_{D_\perp^\sharp} \phi_{\mathcal{A}2} \;\Leftrightarrow\; \forall x \in N^\sharp : \phi_{\mathcal{A}1}(x) \subseteq \phi_{\mathcal{A}2}(x)$$

The bottom element, $\bot_{D^\sharp_\bot}$, is the null environment, $\phi_{\mathcal{A}0}$, mapping every abstract name to the empty set. Using $D^\sharp_\bot$, the abstract semantics of the π-calculus is defined as a function $\mathcal{A}^\pi(\llbracket P \rrbracket)\ \rho\ \phi_\mathcal{A} \in D^\sharp_\bot$, the rules of which are shown in Figure 4.2.

$$
\begin{array}{lll}
(\mathcal{A}^\pi 1) & \mathcal{A}^\pi(\llbracket \mathbf{0} \rrbracket)\ \rho\ \phi_\mathcal{A} & = \ \phi_\mathcal{A} \\
(\mathcal{A}^\pi 2) & \mathcal{A}^\pi(\llbracket x(y).P \rrbracket)\ \rho\ \phi_\mathcal{A} & = \ \phi_\mathcal{A} \\
(\mathcal{A}^\pi 3) & \mathcal{A}^\pi(\llbracket \overline{x}\langle y^t \rangle.P \rrbracket)\ \rho\ \phi_\mathcal{A} & = \ (\bigcup^{\phi}_{x'(z).P' \in \rho} \mathcal{R}^\pi(\llbracket \{\!| P |\!\}_\rho \uplus_\rho \rho[P'/x'(z).P'] \rrbracket)\ \phi'_\mathcal{A})\ \cup_\phi \phi_\mathcal{A} \\
 & \text{where, } \phi'_\mathcal{A} = \phi_\mathcal{A}[\alpha_k(z) \mapsto \ \phi_\mathcal{A}(\alpha_k(z)) \cup \{\alpha_k(t)\}] & \\
 & \text{if, } \varphi_\mathcal{A}(\phi_\mathcal{A}, x) \cap \varphi_\mathcal{A}(\phi_\mathcal{A}, x') \neq \{\} & \\
(\mathcal{A}^\pi 4) & \mathcal{A}^\pi(\llbracket \mathit{if}\ x = y\ \mathit{then}\ P\ \mathit{else}\ Q \rrbracket)\ \rho\ \phi_\mathcal{A} = & \\
 & \left\{ \begin{array}{ll} \mathcal{R}^\pi(\llbracket \{\!| P |\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi_\mathcal{A}, & \text{if, } \varphi_\mathcal{A}(\phi_\mathcal{A}, x) \cap \varphi_\mathcal{A}(\phi_\mathcal{A}, y) \neq \{\} \\ \mathcal{R}^\pi(\llbracket \{\!| Q |\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi_\mathcal{A}, & \text{otherwise} \end{array} \right. \\
(\mathcal{A}^\pi 5) & \mathcal{A}^\pi(\llbracket P + Q \rrbracket)\ \rho\ \phi_\mathcal{A} & = \ \mathcal{R}^\pi(\llbracket \{\!| P |\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi_\mathcal{A}\ \cup_\phi\ \mathcal{R}^\pi(\llbracket \{\!| Q |\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi_\mathcal{A} \\
(\mathcal{A}^\pi 6) & \mathcal{A}^\pi(\llbracket P \mid Q \rrbracket)\ \rho\ \phi_\mathcal{A} & = \ \mathcal{R}^\pi(\llbracket \{\!| P |\!\}_\rho \uplus_\rho \{\!| Q |\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi_\mathcal{A} \\
(\mathcal{A}^\pi 7) & \mathcal{A}^\pi(\llbracket (\nu x)P \rrbracket)\ \rho\ \phi_\mathcal{A} & = \ \mathcal{R}^\pi(\llbracket \{\!| P |\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi_\mathcal{A} \\
(\mathcal{A}^\pi 8) & \mathcal{A}^\pi(\llbracket !P \rrbracket)\ \rho\ \phi_\mathcal{A} & = \ \mathcal{F}^\pi(-1) \\
 & \text{where, } \mathcal{F}^\pi(n) = \mathit{let}\ \phi_1 = \mathcal{A}^\pi(\llbracket \prod_{i=1}^{n} \mathit{ren}(P,i) \rrbracket)\ \rho\ \phi_\mathcal{A}\ \mathit{in} & \\
 & \qquad\qquad \mathit{let}\ \phi_2 = \mathcal{A}^\pi(\llbracket \prod_{i=1}^{n+2} \mathit{ren}(P,i) \rrbracket)\ \rho\ \phi_\mathcal{A}\ \mathit{in} & \\
 & \qquad\qquad \mathit{if}\ \phi_1 = \phi_2\ \mathit{then}\ \phi_1\ \mathit{else}\ \mathcal{F}^\pi(n+1) & \\
 & \text{and, } \forall x \in bn(P), y^t \in n(P) : \mathit{ren}(P,i) = (P[x_i/x])[y^{t_i}/y^t] & \\
(\mathcal{R}^\pi 0) & \mathcal{R}^\pi(\llbracket \rho \rrbracket)\ \phi_\mathcal{A} & = \ \bigcup^{\phi}_{P \in \rho} \mathcal{A}^\pi(\llbracket P \rrbracket)\ (\rho \backslash \{\!| P |\!\}_\rho)\ \phi_\mathcal{A}
\end{array}
$$

Figure 4.2: The abstract semantics of the π-calculus.

The multiset, ρ, holds as usual all the processes composed in parallel with the interpreted process. The function, $\varphi_\mathcal{A} : (N^\sharp \to \wp(\mathit{Tag}^\sharp)) \times \mathcal{N} \to \wp(N^\sharp)$ is defined as follows:

$$\varphi_\mathcal{A}(\phi_\mathcal{A}, x) = \varphi'_\mathcal{A}(\phi_\mathcal{A}, x)_{\{\}}$$

where, $\varphi'_\mathcal{A}(\phi_\mathcal{A}, x)_s =$

$$
\left\{ \begin{array}{ll} \mathit{if}\ \alpha_k(x) \in s\ \mathit{then}\ \{\}\ \mathit{else} \bigcup_{y \in \mathit{value\_of}(\phi_\mathcal{A}(\alpha_k(x)))} \varphi'_\mathcal{A}(\phi_\mathcal{A}, y)_{s \cup \{\alpha_k(x)\}}, & \text{if } \phi_\mathcal{A}(\alpha_k(x)) \neq \{\} \\ \{\alpha_k(x)\}, & \text{otherwise} \end{array} \right.
$$

The function $\varphi'_\mathcal{A}(\phi_\mathcal{A}, x)_s$ terminates only when it evaluates all the abstract input parameters in $\mathit{value\_of}(\phi_\mathcal{A}(\alpha_k(x)))$, whose tags were added to $\phi_\mathcal{A}$ in $(\mathcal{A}^\pi 3)$. Note that this termination is

guaranteed since $\varphi'_{\mathcal{A}}$ never performs a recursive call on an input parameter it has encountered before and the number of input parameters is kept finite by the abstraction, α_k.

The rules of the abstract semantics are described informally as follows. Rules, ($\mathcal{A}^\pi 1$) and ($\mathcal{A}^\pi 2$), for null processes and input actions do not change the value of the $\phi_{\mathcal{A}}$ environment, since no communications take place in these rules. Rule ($\mathcal{A}^\pi 3$) deals with output actions, where the meaning of a process guarded by an output action is given as the union of two $\phi_{\mathcal{A}}$ environments. The first environment reflects all possible communications between the output action and matching input actions in ρ. A communication is fired whenever the sets of values of two channels, as given by $\varphi_{\mathcal{A}}$, have a non-empty intersection. This means that there must have been tags with the same values substituting both channels earlier in the semantics (if the channels were closed under input actions), or that the names of the channels are the same (if the channels were free or restricted names). The effect of the communication is reflected by adding the abstract value of the tag of the message to the value of the substituted input parameter in $\phi_{\mathcal{A}}$. The second environment is an unchanged $\phi_{\mathcal{A}}$ reflecting the no-communications scenario.

The rest of the rules, ($\mathcal{A}^\pi 4$)–($\mathcal{A}^\pi 7$), are straightforward. The rule for replicated processes, ($\mathcal{A}^\pi 8$), attaches subscripts to bound names and tags of the spawned processes according to the number of copy of each process. This is necessary to ensure that these names and tags remain distinct throughout the semantics. The rule uses a least fixed-point calculation for a special function, $\mathcal{F}^\pi : \mathbb{N} \to D^\sharp_\perp$, where the bottom number, $n = -1$, computes the bottom semantic element, $\mathcal{A}^\pi(\![\prod_{i=1}^{-1} ren(P,i)]\!)\ \rho\ \phi_{\mathcal{A}} = \perp_{D^\sharp_\perp}$. It is easy to note that the least fixed point of $\mathcal{F}^\pi$ occurs at the minimum number, n, such that $\mathcal{A}^\pi(\![\prod_{i=1}^{n} ren(P,i)]\!)\ \rho\ \phi_{\mathcal{A}} = \mathcal{A}^\pi(\![\prod_{i=1}^{n+2} ren(P,i)]\!)\ \rho\ \phi_{\mathcal{A}}$. This implies that adding two extra copies of P could not induce any further communications: First, between P and the other processes in parallel, second, between the two added copies of P, and finally, within P itself.

We state, by the following theorem, the fact that the computation of such least fixed points are guaranteed to terminate.

Theorem 4 (Termination of the least fixed-point calculation)
The calculation of rule ($\mathcal{A}^\pi 8$) *terminates.*

Proof. We give a sketch of the proof of the termination property. Two requirements must be satisfied. First, the semantic domain must be finite. This is satisfied by the definition of $D^\sharp_\perp$. The second requirement is to prove that $\mathcal{F}^\pi(n) \sqsubseteq_{D^\sharp_\perp} \mathcal{F}^\pi(n+1)$ (i.e., proving the monotonicity property of $\mathcal{F}^\pi(n)$). This requirement can be restated as $\mathcal{A}^\pi(\![\prod^{n} P]\!)\ \rho\ \phi_{\mathcal{A}} \sqsubseteq_{D^\sharp_\perp}$

$\mathcal{A}^\pi(\![\prod^{n+2} P]\!)\ \rho\ \phi_\mathcal{A}$. To prove this, we simplify the inequality into $\mathcal{A}^\pi(\![Q]\!)\ \rho\ \phi_\mathcal{A} \sqsubseteq \mathcal{A}^\pi(\![Q \mid P \mid P]\!)\ \rho\ \phi_\mathcal{A}$, where $Q = \prod^{n} P$. This is further simplified to become $\mathcal{A}^\pi(\![Q]\!)\ \rho\ \phi_\mathcal{A} \sqsubseteq \mathcal{A}^\pi(\![Q]\!)\ \rho'\ \phi_\mathcal{A}$, where $\rho' = \rho \uplus_\rho \{\!|P|\!\}_\rho \uplus_\rho \{\!|P|\!\}_\rho$. This result can be proven by induction over the rules of $\mathcal{A}^\pi$. In particular, the most interesting rule is that of $(\mathcal{A}^\pi 3)$, where the value of $\phi_\mathcal{A}$ changes. Since $\rho \subseteq \rho'$, then $x'(y).P' \in \rho \Rightarrow x'(y).P' \in \rho'$. This implies that any communications (and the associated substitutions) taking place in $\mathcal{A}^\pi(\![Q]\!)\ \rho\ \phi_\mathcal{A}$ will necessarily take place in $\mathcal{A}^\pi(\![Q]\!)\ \rho'\ \phi_\mathcal{A}$. This eventually leads to the conclusion that $\mathcal{A}^\pi(\![Q]\!)\ \rho\ \phi_\mathcal{A} \sqsubseteq \mathcal{A}^\pi(\![Q]\!)\ \rho'\ \phi_\mathcal{A}$, since the final environment resulting from $\mathcal{A}^\pi(\![Q]\!)\ \rho\ \phi_\mathcal{A}$ will be a subset of the final environment resulting from $\mathcal{A}^\pi(\![Q]\!)\ \rho'\ \phi_\mathcal{A}$ (i.e. the larger system induces more name substitutions). □

In order to be able to prove the safety of the abstract semantics with respect to the non-standard semantics, we need first to prove the safety of the union of environments, $\cup_\phi$.

Lemma 1 (Safety of $\bigcup_\phi$ in the π-calculus)

$\forall i \in \{1 \ldots n\}, n \in \mathbb{N}, \phi_i \in D_\perp, \phi'_i \in D^\sharp_\perp :$

$(\phi = \bigcup_{\phi\ i=1\ldots n} \phi_i) \ \wedge\ (\phi' = \bigcup_{\phi\ i=1\ldots n} \phi'_i) \ \wedge$

$(\exists y \in \mathcal{N} : \varphi_\mathcal{E}(\phi_i, y) \in \phi_i(x) \ \Rightarrow\ \exists t \in \phi'_i(\alpha_k(x)) :\ \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$

$\Rightarrow\quad (\exists y \in \mathcal{N} : \varphi_\mathcal{E}(\phi, y) \in \phi(x) \ \Rightarrow\ \exists t \in \phi'(\alpha_k(x)) :\ \mathit{value\_of}\,(\{t\}) = \{\alpha_k(y)\})$

Proof. Refer to Appendix A.1. □

Hence, the lemma establishes that for any set of concrete environments, $\{\phi_1, \ldots, \phi_n\}$, and their abstract counterparts, $\{\phi'_1, \ldots, \phi'_n\}$, which are related by the abstraction function, α_k, and the tag evaluation function, *value_of*, then the unions, $\phi = \bigcup_{\phi\ i=1\ldots n} \phi_i$ and $\phi' = \bigcup_{\phi\ i=1\ldots n} \phi'_i$, will maintain this relation. Intuitively, the relation states that a name, y, captured by the $\varphi_\mathcal{E}$ function in the concrete semantics, will appear as a tag, t, in the abstract environment, such that the value of t is equivalent to the abstraction, $\alpha_k(y)$.

From this lemma, the safety of the abstract semantics can now be established formally.

Theorem 5 (Safety of the abstract semantics of the π-calculus)

$\forall P, \rho, \phi_\mathcal{E}, \phi_\mathcal{A} :$

$(\mathcal{E}^\pi(\![P]\!)\ \rho\ \phi_\mathcal{E} = (p, \phi'_\mathcal{E})) \ \wedge\ (\mathcal{A}^\pi(\![P]\!)\ \rho\ \phi_\mathcal{A} = \phi'_\mathcal{A}) \wedge$

$(\exists y \in \mathcal{N} : \varphi_\mathcal{E}(\phi_\mathcal{E}, y) \in \phi_\mathcal{E}(x) \ \Rightarrow\ \exists t \in \phi_\mathcal{A}(\alpha_k(x)) : \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$

$\Rightarrow\quad (\exists y \in \mathcal{N} : \varphi_\mathcal{E}(\phi'_\mathcal{E}, y) \in \phi'_\mathcal{E}(x) \ \Rightarrow\ \exists t \in \phi'_\mathcal{A}(\alpha_k(x)) : \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$

Proof. Refer to Appendix A.2. □

The theorem states that values present in the final environment resulting from the concrete non-standard semantics will always be present, as tags of their abstract values, in the environment resulting from the abstract semantics.

4.2.3 The Intruder I

Often when analysing properties of systems, it is the case that the external context is taken into consideration, so as to achieve a more robust and modular analysis. This context is sometimes viewed as the larger and more powerful network, which is an arbitrary collection of intruder processes, I, assumed to be running concurrently with the system under analysis. The capabilities of this intruder include intercepting, reading and modifying any messages that travel over public channels as well as creating fresh messages, sending them to other processes and using them later for further communications. These capabilities vary depending on the network. For example, a local area network may be limited in its ability to communicate with computers external to the network as a result of its isolation from other networks by a firewall. It is also less able to create communication noise as a result of the high quality of service normally expected from local area networks. On the other hand, the Internet has a vast amount of information and is capable of communicating with virtually every computer that has some sort of connection to it (either directly or via a gateway).

For the case of the weakest intruder, its specification is $I \stackrel{\texttt{def}}{=} \mathbf{0}$. However, the model of I adopted in our framework draws on the lines of the most general attacker as given by Dolev and Yao in [47]. In the case of non-cryptographic systems, this model lacks the cryptography aspect and only captures the input/output capabilities of the Dolev-Yao model:

$$I \stackrel{\texttt{def}}{=} (\nu\, i)\ (\bar{i}\langle\kappa_{init}\rangle \mid\ !\, i(\kappa).(\prod_{\forall x,y\in\kappa} \overline{x}\langle y\rangle.\bar{i}\langle\kappa\rangle \mid \prod_{\forall x\in\kappa} x(z).\bar{i}\langle\kappa\cup\{z\}\rangle \mid (\nu\, net)\bar{i}\langle\kappa\cup\{net\}\rangle))$$

In this specification, κ is a set of names representing the knowledge of the intruder, $(\nu\, net)$ allows for the intruder to create fresh data at any time, and i is a channel used for the intruder's internal communications. The initial subprocess, $\bar{i}\langle\kappa_{init}\rangle$, outputs the set of names, κ_{init}, representing an instantiation of the intruder's initial knowledge (in general, $\kappa_{init} = fn(P)$, for the analysed process, P).

The above specification allows the intruder to build its knowledge, κ, by repeatedly inputting over names in its knowledge. The inputted name is then passed as part of the new knowledge to the next instance of the intruder. The intruder can also perform output actions. These are either free output actions sending messages over channels already in κ, or bound output actions that create a copy of the name *net* and send it over the internal

channel i. This allows the intruder to learn *net* without the need to output it first to external processes. The *learning* behaviour is interpreted as the standard union, $\cup$, over κ.

4.2.4 The FTP Server Example

We consider here the example of a simple File Transfer Protocol (FTP) system:

$$\begin{aligned}
ftp \stackrel{\texttt{def}}{=}\ & (!start(x).(\nu\, login)(if\ x = start\ then\ (\nu\, pwd)(Server \mid Client) \\
& \qquad\qquad\qquad\qquad\qquad\qquad else\ ((\nu\, pwd)(Server) \mid I) \\
& \qquad\qquad\qquad\qquad) \\
&)\ \mid\ \overline{start}\langle start^{t1}\rangle.\overline{start}\langle start^{t2}\rangle.\overline{start}\langle start^{t3}\rangle.\overline{start}\langle I_start^{t4}\rangle
\end{aligned}$$

$$\begin{aligned}
Server \stackrel{\texttt{def}}{=}\ & (\nu\, deal)\ login(z).if\ z = pwd\ then \\
& \quad login(data).(\\
& \quad \overline{deal}\langle data^{t5}\rangle\ \mid\ !deal(w).\overline{login}\langle w^{t6}\rangle.login(u).if\ u = logout\ then\ \mathbf{0} \\
& \quad else\ \overline{deal}\langle u^{t7}\rangle\ )\ \ else\ \mathbf{0} \\
Client \stackrel{\texttt{def}}{=}\ & (\nu\, request)\ (\overline{login}\langle pwd^{t8}\rangle.\overline{login}\langle request^{t9}\rangle.login(res).\overline{login}\langle logout^{t10}\rangle)
\end{aligned}$$

Where the tagging scheme, $t1, t2, \ldots$ is chosen such that all tags are distinct from each other (any other tagging scheme maintaining the distinction property could be used instead). The three *start* signals spawn three instances of the *Client*/*Server* system and the fourth *I_start* signal allows the intruder, I, to participate in a session with the server. Each instance of the client/server system shares a one time password, *pwd*, that is not reusable. Additionally, communications between the server and the client or intruder processes are carried out over a secure login channel, *login* (which could be a Secure Socket Layer (SSL) connection). The client process performs a sequence of communications, sending the password and the request for data, and inputting the result from the server. After that, the client process logs out.

The server process accepts a password over the login channel after which it checks whether that password matches the current copy it shares with the client (intruder). If the password is correct, it waits for a data request and then *deals* with that request. For simplicity, the server just routes back the request over the session channel without altering it. The server process then waits for another input, after which it terminates if the new signal is *logout*. If not, it deals again with further requests sending the results over the login channel.

We adopt the following tagging scheme for the specification of I:

$$(\nu i)\ (\bar{i}\langle \kappa_{init}^{t\kappa 1}\rangle \mid\ !\, i(\kappa).(\prod_{\forall x,y\in\kappa} \overline{x}\langle y^{ty}\rangle.\bar{i}\langle \kappa^{t\kappa 2}\rangle \mid \prod_{\forall x\in\kappa} x(zz).\bar{i}\langle \kappa \cup \{zz\}^{t\kappa 3}\rangle \mid (\nu\, net)\bar{i}\langle \kappa \cup \{net\}^{t\kappa 4}\rangle))$$

Assuming the intruder's initial knowledge is set to $\kappa_{init} = \{login, logout\}$, we obtain the following least fixed-point value for $\phi_{\mathcal{A}}$ by applying $\mathcal{A}^{\pi}(\llbracket ftp \rrbracket)\ \{\!|\ |\!\}_{\rho}\ \phi_{\mathcal{A}0}$, where the integer constraint is set to $k = 1$ (uniform analysis):

$$\phi_{\mathcal{A}} = \left[\begin{array}{l} \kappa\ \mapsto \{t6_1, t8_1, t9_1, t10_1, t\kappa1_1, t\kappa2_1, t\kappa3_1, t\kappa4_1\} \\ x_1\ \mapsto \{t1, t2, t3, t4\} \\ z_1\ \mapsto \{t6_1, t8_1, t9_1, t10_1, tlogin_1, tnet_1\} \\ u_1\ \mapsto \{t6_1, t8_1, t9_1, t10_1, tlogin_1, tnet_1\} \\ w_1\ \mapsto \{t5_1\} \\ data_1 \mapsto \{t6_1, t8_1, t9_1, t10_1, tlogin_1, tnet_1\} \\ res_1 \mapsto \{t6_1, t8_1, t9_1, t10_1, tlogin_1, tnet_1\} \end{array}\right]$$

Where the name values for the above input parameters can be converted using $\varphi_{\mathcal{A}}$:

$$\begin{array}{l} \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, \kappa)\ = \{net_1, \underline{pwd_1}, \underline{request_1}, logout, login_1\} \\ \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, x_1)\ = \{start, I\_start\} \\ \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, z_1)\ = \{pwd_1, request_1, logout, login_1, net_1\} \\ \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, u_1)\ = \{pwd_1, request_1, logout, login_1, net_1\} \\ \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, w_1)\ = \{pwd_1, request_1, logout, login_1, net_1\} \\ \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, data_1)\ = \{pwd_1, request_1, logout, login_1, net_1\} \\ \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, res_1)\ = \{pwd_1, request_1, logout, login_1, net_1\} \end{array}$$

We have used underlining to indicate the presence of abnormal information in the intruder's knowledge in the final $\phi_{\mathcal{A}}$ environment. The interpretation detects that the intruder is capable of obtaining the *pwd* and *request* names, which is a dangerous result. This result is a false positive since the uniform interpretation does not distinguish between the different copies of the *login* channels. Therefore, it appears to be possible for the intruder to capture any of the other names outputted over different *login* names. Moreover, it is impossible to distinguish between the information each instance of the client and server processes obtains. For example, the *res* input parameter captures a *request*, but also the *pwd*, *logout*, *login* and *net* messages. As $\phi_{\mathcal{A}}$ is uniform, it is impossible to distinguish between different sessions.

To refine the above results, we increase the value of k from 1 to 4. The intermediate values of $k = 2$ and $k = 3$ will still yield the above false positives for $k = 1$ (since the intruder's session interferes with clients' sessions), therefore, we do not include them here. However, when performing the abstract interpretation for $k = 4$, we obtain the following results:

$$\phi_{\mathcal{A}} = \left[\begin{array}{llll}
\kappa \mapsto \{t\kappa 1_i, t\kappa 2_i, t\kappa 3_i, t\kappa 4_i\} & \text{for } i = 1 \ldots 4 & & \\
x_1 \mapsto \{t1\} & x_2 \mapsto \{t2\} & x_3 \mapsto \{t3\} & x_4 \mapsto \{t4\} \\
z_1 \mapsto \{t8_1\} & z_2 \mapsto \{t8_2\} & z_3 \mapsto \{t8_3\} & z_4 \mapsto \{tnet_i, tlogin_4, \\
 & & & \quad tlogout_4\} \\
u_1 \mapsto \{t10_1\} & u_2 \mapsto \{t10_2\} & u_3 \mapsto \{t10_3\} & u_4 \mapsto \{\} \\
w_1 \mapsto \{t5_1\} & w_2 \mapsto \{t5_2\} & w_3 \mapsto \{t5_3\} & w_4 \mapsto \{\} \\
data_1 \mapsto \{t9_1\} & data_2 \mapsto \{t9_2\} & data_3 \mapsto \{t9_3\} & data_4 \mapsto \{\} \\
res_1 \mapsto \{t6_1\} & res_2 \mapsto \{t6_2\} & res_3 \mapsto \{t6_3\} &
\end{array}\right]$$

The results can be converted back to their name values using the $\varphi_{\mathcal{A}}$ function, as follows:

$$\begin{array}{ll}
\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, \kappa) = \Delta, & \Delta = \{net_i, login_4, logout \mid i = 1, 2, 3, 4\} \\
\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, x_1) = \{start\} & \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, x_2) = \{start\} \\
\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, x_3) = \{start\} & \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, x_4) = \{I_start\} \\
\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, z_1) = \{pwd_1\} & \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, z_2) = \{pwd_2\} \\
\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, z_3) = \{pwd_3\} & \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, z_4) = \Delta \\
\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, u_1) = \{logout\} & \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, u_2) = \{logout\} \\
\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, u_3) = \{logout\} & \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, u_4) = \{\} \\
\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, w_1) = \{request_1\} & \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, w_2) = \{request_2\} \\
\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, w_3) = \{request_3\} & \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, w_4) = \{\} \\
\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, data_1) = \{request_1\} & \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, data_2) = \{request_2\} \\
\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, data_3) = \{request_3\} & \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, data_4) = \{\} \\
\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, res_1) = \{request_1\} & \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, res_2) = \{request_2\} \\
\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, res_3) = \{request_3\} &
\end{array}$$

The results of the non-uniform analysis ($k = 4$) reveal that the intruder is incapable of obtaining information from sessions involving clients. It also shows that the server, in fact, does not process any request from the intruder, since the latter lacks the required password (in this case, pwd_4). Also, a more precise distribution of information between the three copies of the client and server processes is reflected in the results. For example, it is clear that a client can only receive a *request* message as a final result held by the *res* input parameter. Also, each copy of res_i captures the corresponding copy of $request_i$, for $i = 1, 2, 3$.

Another version of the FTP system may contain a faulty client specified as follows:

$$Client \stackrel{\mathtt{def}}{=} (\nu\, request)(\overline{login}\langle pwd^{t8}\rangle.\overline{login}\langle request^{t9}\rangle.login(res).\overline{login}\langle logout^{t10}\rangle.\overline{covert}\langle pwd^{t11}\rangle)$$

The specification of the client reveals, over a *covert* channel, the password it shares with the

server to the intruder. Now, if we perform the abstract interpretation for $k = 1$ setting the knowledge of the intruder to $\kappa_{init} = \{covert, login, logout\}$, we obtain the following results:

$$\phi_{\mathcal{A}} = \left[\begin{array}{l} \kappa \ \mapsto \{t6_1, t8_1, t9_1, t10_1, t\kappa1_1, t\kappa2_1, t\kappa3_1, t\kappa4_1\} \\ x_1 \ \mapsto \{t1, t2, t3, t4\} \\ z_1 \ \mapsto \{t6_1, t8_1, t9_1, t10_1, tlogin_1, tnet_1, tcovert_1\} \\ u_1 \ \mapsto \{t6_1, t8_1, t9_1, t10_1, tlogin_1, tnet_1, tcovert_1\} \\ w_1 \ \mapsto \{t5_1\} \\ data_1 \mapsto \{t6_1, t8_1, t9_1, t10_1, tlogin_1, tnet_1, tcovert_1\} \\ res_1 \mapsto \{t6_1, t8_1, t9_1, t10_1, tlogin_1, tnet_1, tcovert_1\} \end{array}\right]$$

Which when converted using $\varphi_{\mathcal{A}}$ yields the following results:

$$\begin{array}{l} \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, \kappa) \ = \{net_1, \underline{pwd_1}, \underline{request_1}, logout, login_1, covert\} \\ \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, x_1) \ = \{start, I\_start\} \\ \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, z_1) \ = \{pwd_1, request_1, logout, login_1, net_1, covert\} \\ \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, u_1) \ = \{pwd_1, request_1, logout, login_1, net_1, covert\} \\ \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, w_1) \ = \{pwd_1, request_1, logout, login_1, net_1, covert\} \\ \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, data_1) \ = \{pwd_1, request_1, logout, login_1, net_1, covert\} \\ \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, res_1) \ = \{pwd_1, request_1, logout, login_1, net_1, covert\} \end{array}$$

The intruder obtains the *pwd* and *request* names. Refining the interpretation further by setting $k = 4$, reveals that the intruder, in fact, can only capture the password (when the client reveals it over the *covert* channel):

$$\phi_{\mathcal{A}} = \left[\begin{array}{llll} \kappa \ \mapsto \{t\kappa1_i, t\kappa2_i, t\kappa3_i, t\kappa4_i, \underline{t11_1, t11_2, t11_3}\} & \text{for } i = 1 \ldots 4 & & \\ x_1 \ \mapsto \{t1\} & x_2 \ \mapsto \{t2\} & x_3 \ \mapsto \{t3\} & x_4 \ \mapsto \{t4\} \\ z_1 \ \mapsto \{t8_1\} & z_2 \ \mapsto \{t8_2\} & z_3 \ \mapsto \{t8_3\} & \\ & z_4 \ \mapsto \{tnet_i, tlogin_4, tlogout_4, \underline{t11_1, t11_2, t11_3}\} & & \\ u_1 \ \mapsto \{t10_1\} & u_2 \ \mapsto \{t10_2\} & u_3 \ \mapsto \{t10_3\} & u_4 \ \mapsto \{\} \\ w_1 \ \mapsto \{t5_1\} & w_2 \ \mapsto \{t5_2\} & w_3 \ \mapsto \{t5_3\} & w_4 \ \mapsto \{\} \\ data_1 \mapsto \{t9_1\} & data_2 \mapsto \{t9_2\} & data_3 \mapsto \{t9_3\} & data_4 \mapsto \{\} \\ res_1 \mapsto \{t6_1\} & res_2 \mapsto \{t6_2\} & res_3 \mapsto \{t6_3\} & \end{array}\right]$$

With the converted results obtained using $\varphi_{\mathcal{A}}$ as follows:

$\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, \kappa) = \Delta$, $\Delta = \{net_i, login_4, logout, covert, \underline{pwd_j} \mid i = 1,2,3,4 \text{ and } j = 1,2,3\}$

$\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, x_1) = \{start\}$ $\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, x_2) = \{start\}$

$\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, x_3) = \{start\}$ $\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, x_4) = \{I\_start\}$

$\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, z_1) = \{pwd_1\}$ $\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, z_2) = \{pwd_2\}$

$\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, z_3) = \{pwd_3\}$ $\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, z_4) = \Delta$

$\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, u_1) = \{logout\}$ $\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, u_2) = \{logout\}$

$\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, u_3) = \{logout\}$ $\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, u_4) = \{\}$

$\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, w_1) = \{request_1\}$ $\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, w_2) = \{request_2\}$

$\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, w_3) = \{request_3\}$ $\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, w_4) = \{\}$

$\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, data_1) = \{request_1\}$ $\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, data_2) = \{request_2\}$

$\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, data_3) = \{request_3\}$ $\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, data_4) = \{\}$

$\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, res_1) = \{request_1\}$ $\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, res_2) = \{request_2\}$

$\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, res_3) = \{request_3\}$

The results of the non-uniform interpretation reveal that the intruder is incapable of compromising the system, even in the case where it compromises the password of each client. This is due to the fact that these passwords are one-time passwords. The name *request* does not appear in any of the values for κ (the intruder's knowledge). Also, the session between the server and the intruder (for z_4) indicates that the intruder is incapable of processing its data by the server, since it cannot pass beyond the login stage.

4.3 The Spi Calculus

We extend in this section the approach we introduced in Section 4.2 for constructing an abstract interpretation for the spi calculus. The standard semantics is enhanced to be able to capture the property of term substitutions occurring as a result of the message passing and message processing behaviours. Terms can substitute local variables in the residual processes due to input/output as well as the success of the cryptographic operations, like decryption or signature verification. Such substitutions may reveal sensitive information to intruder processes. For example, if the intruder inputs the key to the decryption of a class of ciphertexts, it will be able to decipher any sensitive information encrypted with that key.

4.3.1 Non-standard Semantics

The non-standard semantics of the spi calculus extends the standard denotational semantics introduced in Section 3.3.3, where term substitutions are recorded in a special environment $\phi_{\mathcal{E}} : V \to \wp(T)$ that maps each variable of a closed process to the set of semantic terms that may substitute that variable during the evaluation of the meaning of a process. Since the non-standard semantics is precise (copies of bound names and variables are always distinct), each variable will be mapped to a singleton set at most per choice of control flow, representing the term that substitutes the variable.

A domain, $D_\perp = V \to \wp(T)$, can be constructed, ordered by subset inclusion:

$$\forall \phi_{\mathcal{E}1}, \phi_{\mathcal{E}2} \in D_\perp : \phi_{\mathcal{E}1} \sqsubseteq_{D_\perp} \phi_{\mathcal{E}2} \Leftrightarrow \forall x \in V : \phi_{\mathcal{E}1}(x) \subseteq \phi_{\mathcal{E}2}(x)$$

With the bottom element, $\perp_{D_\perp}$, being the null environment, $\phi_{\mathcal{E}0}$, that maps each variable to the empty set. The union of environments operation, $\cup_\phi$, can also be defined as follows:

$$\forall \phi_{\mathcal{E}1}, \phi_{\mathcal{E}2} \in D_\perp, x \in V : (\phi_{\mathcal{E}1} \cup_\phi \phi_{\mathcal{E}2})(x) = \phi_{\mathcal{E}1}(x) \cup \phi_{\mathcal{E}2}(x)$$

The non-standard semantic domain is formed by pairing $D_\perp$ with the standard semantic domain, $Spi_\perp$, resulting in $Spi_\perp \times D_\perp$. The bottom element of this domain is the pair $(\perp_{Spi_\perp}, \perp_{D_\perp})$ representing the empty set map, $\emptyset$, and the null environment, $\phi_{\mathcal{E}0}$.

The non-standard semantics for the spi calculus is defined by the semantic function, $\mathcal{E}^{spi}(\![P]\!)\ \rho\ \phi_{\mathcal{E}} \in (Spi_\perp \times D_\perp)$, on the structure of P as in Figure 4.3. The ρ multiset holds all the processes in parallel with the process under interpretation. The definition of the $\varphi_{\mathcal{E}} : (V \to \wp(T)) \times Term \to T$ function allows for the meaning of a term to be computed under a particular $\phi_{\mathcal{E}}$ environment:

$$\forall \phi_{\mathcal{E}}, M : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) = \begin{cases} t, & \text{if } M \in V \ \wedge\ \phi_{\mathcal{E}}(M) = \{t\} \\ M, & \text{if } M \in N \\ sec(\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M'), \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, N)), & \text{if } M = \{M'\}_N \\ pub(\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M'), \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, N)), & \text{if } M = \{\!|M'|\!\}_N \\ sig(\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M'), \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, N)), & \text{if } M = [\!\{M'\}\!]_N \\ (\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M_1), \ldots, \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M_n)), & \text{if } M = (M_1, \ldots, M_n) \end{cases}$$

The semantic rules are described as follows. In rule ($\mathcal{E}^{spi}1$), the meaning of a null process is described as the pair $(\emptyset, \phi_{\mathcal{E}})$, where $\phi_{\mathcal{E}}$ is the environment supplied to the rule initially. Rules ($\mathcal{E}^{spi}2$) and ($\mathcal{E}^{spi}3$) deal with the cases of input and output actions, respectively. Communications are dealt with in rule ($\mathcal{E}^{spi}3$) for output actions, therefore, $\phi_{\mathcal{E}}$ remains unchanged in rule ($\mathcal{E}^{spi}2$) for input actions. The rule for output actions requires that terms

$(\mathcal{E}^{spi}1)$ $\mathcal{E}^{spi}(\llbracket \mathbf{0} \rrbracket)\ \rho\ \phi_{\mathcal{E}} = (\emptyset, \phi_{\mathcal{E}})$

$(\mathcal{E}^{spi}2)$ $\mathcal{E}^{spi}(\llbracket M(x).P \rrbracket)\ \rho\ \phi_{\mathcal{E}} = (\{\!|in(\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M), \lambda x.p')|\!\}, \phi_{\mathcal{E}})$

where, $(p', \phi'_{\mathcal{E}}) = \mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_\rho \rrbracket)\ \phi_{\mathcal{E}}$ and, $\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) \in N$

$(\mathcal{E}^{spi}3)$ $\mathcal{E}^{spi}(\llbracket \overline{M}\langle L \rangle.P \rrbracket)\ \rho\ \phi_{\mathcal{E}} =$

$$\Big(\biguplus_{M'(z).P' \in \rho} \{\!|tau(p')|\!\} \ \uplus\ \{\!|out(\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M), \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, L), p'')|\!\}, \bigcup_{\phi}{}_{M'(z).P' \in \rho} \phi'_{\mathcal{E}} \ \cup_\phi\ \phi_{\mathcal{E}} \Big)$$

if, $\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) = \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M') \in N$

where, $(p', \phi'_{\mathcal{E}}) = \mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \rho[P'/M'(z).P'] \rrbracket)\ \phi_{\mathcal{E}}[z \mapsto \{\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, L)\}]$

and, $(p'', \phi''_{\mathcal{E}}) = \mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_\rho \rrbracket)\ \phi_{\mathcal{E}}$

$(\mathcal{E}^{spi}4)$ $\mathcal{E}^{spi}(\llbracket (\nu a)P \rrbracket)\ \rho\ \phi_{\mathcal{E}} = (new(\lambda a.p'), \phi'_{\mathcal{E}})$

where, $(p', \phi'_{\mathcal{E}}) = \mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi_{\mathcal{E}}$

$(\mathcal{E}^{spi}5)$ $\mathcal{E}^{spi}(\llbracket P \mid Q \rrbracket)\ \rho\ \phi_{\mathcal{E}} = \mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \{\!|Q|\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi_{\mathcal{E}}$

$(\mathcal{E}^{spi}6)$ $\mathcal{E}^{spi}(\llbracket !P \rrbracket)\ \rho\ \phi_{\mathcal{E}} = \mathcal{F}^{spi}(-1)$

where, $\mathcal{F}^{spi}(n) = let\ v_1 = \mathcal{E}^{spi}(\llbracket \prod_{i=1}^{n} P[bnv_i(P)/bnv(P)] \rrbracket)\ \rho\ \phi_{\mathcal{E}}\ in$

$let\ v_2 = \mathcal{E}^{spi}(\llbracket \prod_{i=1}^{n+2} P[bnv_i(P)/bnv(P)] \rrbracket)\ \rho\ \phi_{\mathcal{E}}\ in$

$if\ v_1 = v_2\ then\ v_1\ else\ \mathcal{F}^{spi}(n+1)$

and, $bnv_i(P) = \{x_i \mid x \in bnv(P)\}$

$(\mathcal{E}^{spi}7)$ $\mathcal{E}^{spi}(\llbracket if\ M = L\ then\ P\ else\ Q \rrbracket)\ \rho\ \phi_{\mathcal{E}} =$

$$\begin{cases} \mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi_{\mathcal{E}}, & \text{if } \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) = \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, L) \\ \mathcal{R}^{spi}(\llbracket \{\!|Q|\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi_{\mathcal{E}}, & \text{otherwise} \end{cases}$$

Figure 4.3: The non-standard semantics of the spi calculus.

$(\mathcal{E}^{spi}8)$ $\mathcal{E}^{spi}(\llbracket let\ (x_1,\ldots,x_n) = M\ in\ P\ else\ Q \rrbracket)\ \rho\ \phi_{\mathcal{E}} =$

$$\begin{cases} \mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi'_{\mathcal{E}}, & \text{if } \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) = (t_1,\ldots,t_n) \\ \text{where, } \phi'_{\mathcal{E}} = \phi_{\mathcal{E}}[x_1 \mapsto \{t_1\},\ldots,x_n \mapsto \{t_n\}] & \\ \mathcal{R}^{spi}(\llbracket \{\!|Q|\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi_{\mathcal{E}}, & \text{otherwise} \end{cases}$$

$(\mathcal{E}^{spi}9)$ $\mathcal{E}^{spi}(\llbracket case\ L\ of\ \{x\}_N\ in\ P\ else\ Q \rrbracket)\ \rho\ \phi_{\mathcal{E}} =$

$$\begin{cases} \mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi'_{\mathcal{E}}, & \text{if } \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, L) = sec(t,k) \text{ and } \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, N) = k \\ \text{where, } \phi'_{\mathcal{E}} = \phi_{\mathcal{E}}[x \mapsto \{t\}] & \\ \mathcal{R}^{spi}(\llbracket \{\!|Q|\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi_{\mathcal{E}}, & \text{otherwise} \end{cases}$$

$(\mathcal{E}^{spi}10)$ $\mathcal{E}^{spi}(\llbracket case\ L\ of\ \{\!|x|\!\}_N\ in\ P\ else\ Q \rrbracket)\ \rho\ \phi_{\mathcal{E}} =$

$$\begin{cases} \mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi'_{\mathcal{E}}, & \text{if } \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, L) = pub(t, k^+) \text{ and } \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, N) = k^- \\ \text{where, } \phi'_{\mathcal{E}} = \phi_{\mathcal{E}}[x \mapsto \{t\}] & \\ \mathcal{R}^{spi}(\llbracket \{\!|Q|\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi_{\mathcal{E}}, & \text{otherwise} \end{cases}$$

$(\mathcal{E}^{spi}11)$ $\mathcal{E}^{spi}(\llbracket case\ L\ of\ [\![x]\!]_N\ in\ P\ else\ Q \rrbracket)\ \rho\ \phi_{\mathcal{E}} =$

$$\begin{cases} \mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi'_{\mathcal{E}}, & \text{if } \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, L) = sig(t, k^-) \text{ and } \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, N) = k^+ \\ \text{where, } \phi'_{\mathcal{E}} = \phi_{\mathcal{E}}[x \mapsto \{t\}] & \\ \mathcal{R}^{spi}(\llbracket \{\!|Q|\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi_{\mathcal{E}}, & \text{otherwise} \end{cases}$$

$(\mathcal{R}^{spi}0)$ $\mathcal{R}^{spi}(\llbracket \rho \rrbracket)\ \phi_{\mathcal{E}} = (\biguplus_{P\in\rho} p', \bigcup_{P\in\rho}{}_{\phi} \phi'_{\mathcal{E}})$, where, $(p', \phi'_{\mathcal{E}}) = \mathcal{E}^{spi}(\llbracket P \rrbracket)\ (\rho \backslash \{\!|P|\!\}_\rho)\ \phi_{\mathcal{E}}$

Figure 4.3: The non-standard semantics of the spi calculus (continued).

used as channels should evaluate to names, and communications occur whenever an input channel is matched in ρ. The value of $\phi_{\mathcal{E}}$ is updated with the message term substituting the input parameter. Rule ($\mathcal{E}^{spi}4$) interprets the meaning of a restriction using the *new* operation on the first element of the resulting pair, whereas the second element reflects the environment resulting from the residue. This is justified as internal communications are preserved by restriction. Rule ($\mathcal{E}^{spi}5$) adds two parallel processes to the multiset, ρ.

The replication of processes is dealt with in rule ($\mathcal{E}^{spi}6$) by computing a special function, $\mathcal{F}^{spi} : \mathbb{N} \rightarrow Spi_{\perp} \times D_{\perp}$, starting at the bottom number, $n = -1$, and incrementing n until we reach a least fixed-point for $v_1 \in Spi_{\perp} \times D_{\perp}$. Such a computation is not guaranteed to terminate due to the infinite nature of the non-standard semantic domain, $Spi_{\perp} \times D_{\perp}$. Also, α-conversion renames the set of bound names and variables of each process copy, while maintaining the compositionality of the semantics. Rule ($\mathcal{E}^{spi}7$) deals with a conditional process, where the meaning of the overall process is chosen from the two branch processes based on the semantic equality of the compared terms. Pair splitting is dealt with in rule ($\mathcal{E}^{spi}8$) where the $\phi_{\mathcal{E}}$ is updated to hold the result of the substitution of local variables by elements of a tuple. The rest of the rules ($\mathcal{E}^{spi}9$)–($\mathcal{E}^{spi}11$) deal with cryptographic processes performing secret-key decryption, public-key decryption and digital signature verification. The success of these operations will result in the $\phi_{\mathcal{E}}$ being updated and it depends on the meaning of the term being decrypted (verified) and the cryptographic key used. If the operation fails, a different process is chosen and added to ρ, without affecting $\phi_{\mathcal{E}}$.

The correctness requirement for the non-standard semantics of the spi calculus, with respect to its standard semantics, is expressed in the following theorem.

Theorem 6 (Correctness of the Non-Standard Semantics of the Spi Calculus)
$\forall P \in \mathcal{P} : (\mathcal{S}^{spi}(\llbracket P \rrbracket)\ \rho\ \phi_{\mathcal{S}} = p)\ \wedge\ (\mathcal{E}^{spi}(\llbracket P \rrbracket)\ \rho\ \phi_{\mathcal{E}} = (p', \phi'_{\mathcal{E}})) \quad \Rightarrow \quad p = p'$

Proof. The proof is by induction over the standard and non-standard semantics. □

The theorem states that the standard element of the non-standard semantics is equivalent to the value obtained from the standard semantics. In other words, the standard meaning can be extracted from the non-standard meaning.

4.3.2 Abstract Semantics

In this section, we redefine the abstraction used in interpreting processes in the π-calculus (Section 4.2.2) to accommodate the cryptographic behaviour of processes in the spi calculus. In a π-calculus process, terms can only be names; therefore tagging names appearing as messages was sufficient to construct a mapping from input parameters to sets of tags (the

$\phi_{\mathcal{A}}$ environment). In a spi calculus process, terms can be constructed from cryptographic operations, and hence, they are complex data structures that may grow in depth as a result of replicated behaviour. Therefore, different tagging is required in the spi calculus.

We begin by assuming again, a finite predomain of tags, *Tag*, ranged over by $t, \dot{t}, \ddot{t}$, where t is the tag of a generic term, $\dot{t}$ is the tag of a primitive term (name, variable) and $\ddot{t}$ is the tag of a complex term (ciphertext, signature, tuple). Next we tag (sub)terms of the analysed process with unique tags. More precisely, we tag M in the following constructs:

- *let* $(x_1, \ldots, x_n) = (M_1, \ldots, M_n)$ *in* P *else* Q
- *case* $\{M\}_L$ *of* $\{x\}_N$ *in* P *else* Q
- *case* $\{\!|M|\!\}_L$ *of* $\{\!|x|\!\}_N$ *in* P *else* Q
- *case* $[\!\{M\}\!]_L$ *of* $[\!\{x\}\!]_N$ *in* P *else* Q
- $\overline{N}\langle M\rangle.P$.

For example, tagging the term $\{(\{a\}_c, \{b\}_e)\}_d$ yields $\{(\{a^{\dot{t}1}\}_c^{\ddot{t}1}, \{b^{\dot{t}2}\}_e^{\ddot{t}2})^{\ddot{t}3}\}_d^{\ddot{t}4}$. Now, we can define the following functions over tags, terms and processes:

- $value\_of(\{t_1, \ldots, t_n\}) = \{M_1, \ldots, M_n\}$. This function can be applied to a set of tags, $\{t_1, \ldots, t_n\}$, returning the corresponding set of terms, $\{M_1, \ldots, M_n\}$. Hence, $value\_of(\{\dot{t}1, \ddot{4}\}) = \{a^{\dot{t}1},\ \{(\{a^{\dot{t}1}\}_c^{\ddot{t}1}, \{b^{\dot{t}2}\}_e^{\ddot{t}2})^{\ddot{t}3}\}_d^{\ddot{t}4}\ \}$.
- $tags\_of(P) = \{t_1, \ldots, t_n\}$. This function returns the set of tags, $\{t_1, \ldots, t_n\}$, used in a process, P. For example, $tags\_of(\overline{m}\langle a^{\dot{t}1}\rangle.\overline{m}\langle\{(b^{\dot{t}2}, c^{\dot{t}3})^{\ddot{t}1}\}_k^{\ddot{t}2}\rangle.\mathbf{0}) = \{\dot{t}1, \dot{t}2, \dot{t}3, \ddot{t}1, \ddot{t}2\}$.
- $untag(\{M'_1, \ldots, M'_n\}) = \{M_1, \ldots, M_n\}$. When applied to a set of tagged terms, $\{M'_1, \ldots, M'_n\}$, this function removes all associated tags yielding a set of untagged terms, $\{M_1, \ldots, M_n\}$. Hence, $untag(\{a^{\dot{t}5},\ \{(a^{\dot{t}1}, \{b^{\dot{t}2}\}_e^{\ddot{t}2})^{\ddot{t}3}\}_d^{\ddot{t}4}\ \}) = \{a,\ \{(a, \{b\}_e)\}_d\ \}$. The function behaves as *id* if a term, M', has no tags.

We now introduce the $\alpha_{k,k'}$ abstraction function, which keeps to a finite level, the number of copies of bound variables, names and tags captured in the abstract semantics.

Definition 2 *Define the abstraction,* $\alpha_{k,k'} : \mathbb{N} \times \mathbb{N} \times (V + N + Tag) \rightarrow (V^\sharp + N^\sharp + Tag^\sharp)$*:*

$$\forall M \in (V + N + Tag), i, k, k' \in \mathbb{N} : \alpha_{k,k'}(M) = \begin{cases} \dot{t}_k, & \text{if } M = \dot{t}_i \in Tag \text{ and } i > k \\ \ddot{t}_{k'}, & \text{if } M = \ddot{t}_i \in Tag \text{ and } i > k' \\ x_k, & \text{if } M = x_i \in V \text{ and } i > k \\ a_k, & \text{if } M = a_i \in N \text{ and } i > k \\ M, & \text{otherwise} \end{cases}$$

The resulting abstract predomains, $V^\sharp$, $N^\sharp$ and $Tag^\sharp$, can be defined as $V^\sharp = V \backslash \{x_j \mid j > k\}$, $N^\sharp = N \backslash \{a_j \mid j > k\}$ and $Tag^\sharp = Tag \backslash (\{\dot{t}_j \mid j > k\} \cup \{\ddot{t}_i \mid i > k'\})$. Informally, k constrains the number of bound variables and names, and tags of primitive terms, whereas k' constrains the number of tags of complex terms. In effect, constraining the tags of primitive terms implies limiting the copies of bound names and variables carrying the tags, whereas constraining the number of tags of complex terms means limiting the depth of complex data structures.

For example, in the process $!(\nu\, n)\overline{a}\langle n^{\dot{t}}\rangle \mid !a(x)$, it is possible to spawn infinite copies of each replication, $!(\nu\, n)\overline{a}\langle n^{\dot{t}}\rangle \mid !a(x) \mid (\nu\, n_1)\overline{a}\langle n_1^{\dot{t}_1}\rangle \mid a(x_1) \mid (\nu\, n_2)\overline{a}\langle n_2^{\dot{t}_2}\rangle \mid a(x_2) \mid \ldots$. It is clear that $\dot{t}$ is an indicator to the number of copies the new name, n, has after spawning each process. On the other hand, the process $!a(x).\overline{a}\langle\{x\}_k^{\ddot{t}}\rangle \mid \overline{a}\langle b\rangle$, which can be rewritten as $!a(x).\overline{a}\langle\{x\}_k^{\ddot{t}}\rangle \mid a(x_1).\overline{a}\langle\{x_1\}_k^{\ddot{t}_1}\rangle \mid a(x_2).\overline{a}\langle\{x_2\}_k^{\ddot{t}_2}\rangle \mid \overline{a}\langle b\rangle \mid \ldots$ demonstrates the role of $\ddot{t}$ as an indicator to the number of times the ciphertext, $\{x\}_k$, is applied to the name, b.

Using the $\alpha_{k,k'}$ abstraction, we construct the abstract environment $\phi_{\mathcal{A}} : V^\sharp \to \wp(Tag^\sharp)$, which maps each abstract bound variable of the analysed process to a set of tags, representing terms that could substitute that variable during the abstract semantics. An abstract domain $D_\perp^\sharp = V^\sharp \to \wp(Tag^\sharp)$ is formed ordered by subset inclusion:

$$\forall \phi_{\mathcal{A}1}, \phi_{\mathcal{A}2} \in D_\perp^\sharp, x \in V^\sharp : \phi_{\mathcal{A}1} \sqsubseteq_{D_\perp^\sharp} \phi_{\mathcal{A}2} \Leftrightarrow \phi_{\mathcal{A}1}(x) \subseteq \phi_{\mathcal{A}2}(x)$$

The bottom element, $\perp_{D_\perp^\sharp}$, is the null environment, $\phi_{\mathcal{A}0}$, mapping each variable to $\{\}$.

Taking $D_\perp^\sharp$ as the abstract semantic domain, we can define the abstract semantics of the spi calculus by the function $\mathcal{A}^{spi}(\![P]\!)\, \rho\, \phi_{\mathcal{A}} \in D_\perp^\sharp$, shown in Figure 4.4. The semantics utilises the multiset, ρ, to hold all the processes in parallel with the analysed process. The special function, $\varphi_{\mathcal{A}} : (V^\sharp \to \wp(Tag^\sharp)) \times Term \to \wp(Term)$, returns a set of terms corresponding to a term, M, given substitutions captured by $\phi_{\mathcal{A}}$:
$\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, M) = \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, M')_{\{\}}$, where, $M' = M[\alpha_{k,k'}(t)/t][\alpha_{k,k'}(x)/x][\alpha_{k,k'}(n)/n]$ and,
$\varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, M)_s = \mathit{if}\ M \in s\ \mathit{then}\ \{\}\ \mathit{else}$

$$\begin{cases} \bigcup_{L \in value\_of(\phi_{\mathcal{A}}(untag(M)))} \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, L)_{s \cup \{M\}} & \text{if } M \in \mathcal{V} \\ \{M\}, & \text{if, } M \in \mathcal{N} \\ \{\forall N' \in \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, N)_{s\cup\{M\}}, L' \in \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, L)_{s\cup\{M\}} : \ \{N'\}^t_{L'}\ \}, & \text{if, } M = \{N\}^t_L \\ \{\forall N' \in \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, N)_{s\cup\{M\}}, L' \in \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, L)_{s\cup\{M\}} : \ \{\!|N'|\!\}^t_{L'}\ \}, & \text{if, } M = \{\!|N|\!\}^t_L \\ \{\forall N' \in \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, N)_{s\cup\{M\}}, L' \in \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, L)_{s\cup\{M\}} : \ [\![N']\!]^t_{L'}\ \}, & \text{if, } M = [\![N]\!]^t_L \\ \{\forall M'_1 \in \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, M_1)_{s\cup\{M\}}, \ldots, M'_n \in \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, M_n)_{s\cup\{M\}} : \\ \quad (M'_1, \ldots, M'_n)^t\ \}, & \text{if, } M = (M_1, \ldots, M_n)^t \end{cases}$$

$(\mathcal{A}^{spi}1)$ $\mathcal{A}^{spi}(\llbracket \mathbf{0} \rrbracket)\ \rho\ \phi_{\mathcal{A}} = \phi_{\mathcal{A}}$

$(\mathcal{A}^{spi}2)$ $\mathcal{A}^{spi}(\llbracket M(x).P \rrbracket)\ \rho\ \phi_{\mathcal{A}} = \phi_{\mathcal{A}}$

$(\mathcal{A}^{spi}3)$ $\mathcal{A}^{spi}(\llbracket \overline{M}\langle L^t \rangle.P \rrbracket)\ \rho\ \phi_{\mathcal{A}} = (\bigcup_{\substack{\phi \\ M'(z).P' \in \rho}} \phi'_{\mathcal{A}})\ \cup_{\phi}\ \phi_{\mathcal{A}}$

if, $untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, M)) \cap untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, M')) \cap \mathcal{N} \neq \{\}$

where, $\phi'_{\mathcal{A}} = \mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_{\rho} \uplus_{\rho} \rho[P'/M'(z).P'] \rrbracket)\ \phi''_{\mathcal{A}}$

and, $\phi''_{\mathcal{A}} = \phi_{\mathcal{A}}[\alpha_{k,k'}(z) \mapsto\ \phi_{\mathcal{A}}(\alpha_{k,k'}(z)) \cup \{\alpha_{k,k'}(t)\}]$

$(\mathcal{A}^{spi}4)$ $\mathcal{A}^{spi}(\llbracket (\nu a)P \rrbracket)\ \rho\ \phi_{\mathcal{A}} = \mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_{\rho} \uplus_{\rho} \rho \rrbracket)\ \phi_{\mathcal{A}}$

$(\mathcal{A}^{spi}5)$ $\mathcal{A}^{spi}(\llbracket P \mid Q \rrbracket)\ \rho\ \phi_{\mathcal{A}} = \mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_{\rho} \uplus_{\rho} \{\!|Q|\!\}_{\rho} \uplus_{\rho} \rho \rrbracket)\ \phi_{\mathcal{A}}$

$(\mathcal{A}^{spi}6)$ $\mathcal{A}^{spi}(\llbracket !P \rrbracket)\ \rho\ \phi_{\mathcal{A}} = \mathcal{F}^{spi}(-1)$

where, $\mathcal{F}^{spi}(n) = let\ \phi_1 = \mathcal{A}^{spi}(\llbracket \prod_{i=1}^{n} ren(P,i) \rrbracket)\ \rho\ \phi_{\mathcal{A}}\ in$

$let\ \phi_2 = \mathcal{A}^{spi}(\llbracket \prod_{i=1}^{n+2} ren(P,i) \rrbracket)\ \rho\ \phi_{\mathcal{A}}\ in$

$if\ \phi_1 = \phi_2\ then\ \phi_1\ else\ \mathcal{F}^{spi}(n+1)$

and, $\forall x \in bnv(P), t \in tags\_of(P) : ren(P,i) = (P[x_i/x])[t_i/t]$

$(\mathcal{A}^{spi}7)$ $\mathcal{A}^{spi}(\llbracket if\ M = L\ then\ P\ else\ Q \rrbracket)\ \rho\ \phi_{\mathcal{A}} =$

$$\begin{cases} \mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_{\rho} \uplus_{\rho} \rho \rrbracket)\ \phi_{\mathcal{A}}, & \text{if, } untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, M)) \cap untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, L)) \neq \{\} \\ \mathcal{R}^{spi}(\llbracket \{\!|Q|\!\}_{\rho} \uplus_{\rho} \rho \rrbracket)\ \phi_{\mathcal{A}}, & \text{otherwise} \end{cases}$$

Figure 4.4: The abstract semantics of the spi calculus.

$$(\mathcal{A}^{spi}8)\quad \mathcal{A}^{spi}(\llbracket \mathit{let}\ (x_1,\ldots,x_n) = M\ \mathit{in}\ P\ \mathit{else}\ Q\rrbracket)\ \rho\ \phi_{\mathcal{A}} =$$

$$\begin{cases} \bigcup_{\substack{\phi \\ (M_1^{t_1},\ldots,M_n^{t_n})\in\varphi_{\mathcal{A}}(\phi_{\mathcal{A}},M)}} \mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \rho\rrbracket)\ \phi'_{\mathcal{A}}, \\ \text{if } \exists(M_1^{t_1},\ldots,M_n^{t_n}) \in \varphi_{\mathcal{A}}(\phi_{\mathcal{A}},M) \\ \text{where, } \phi'_{\mathcal{A}} = \phi_{\mathcal{A}}[\alpha_{k,k'}(x_1) \mapsto \phi_{\mathcal{A}}(\alpha_{k,k'}(x_1)) \cup \{\alpha_{k,k'}(t_1)\},\ldots, \\ \qquad\qquad \alpha_{k,k'}(x_n) \mapsto \phi_{\mathcal{A}}(\alpha_{k,k'}(x_n)) \cup \{\alpha_{k,k'}(t_n)\}] \\ \mathcal{R}^{spi}(\llbracket \{\!|Q|\!\}_\rho \uplus_\rho \rho\rrbracket)\ \phi_{\mathcal{A}}, \qquad \text{otherwise} \end{cases}$$

$$(\mathcal{A}^{spi}9)\quad \mathcal{A}^{spi}(\llbracket \mathit{case}\ L\ \mathit{of}\ \{x\}_N\ \mathit{in}\ P\ \mathit{else}\ Q\rrbracket)\ \rho\ \phi_{\mathcal{A}} =$$

$$\begin{cases} \bigcup_{\substack{\phi \\ \{M^t\}_n\in\varphi_{\mathcal{A}}(\phi_{\mathcal{A}},L)}} \mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \rho\rrbracket)\ \phi'_{\mathcal{A}}, \quad \text{if, } n \in \varphi_{\mathcal{A}}(\phi_{\mathcal{A}},N) \\ \text{where, } \phi'_{\mathcal{A}} = \phi_{\mathcal{A}}[\alpha_{k,k'}(x) \mapsto \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) \cup \{\alpha_{k,k'}(t)\}] \\ \mathcal{R}^{spi}(\llbracket \{\!|Q|\!\}_\rho \uplus_\rho \rho\rrbracket)\ \phi_{\mathcal{A}}, \qquad \text{otherwise} \end{cases}$$

$$(\mathcal{A}^{spi}10)\quad \mathcal{A}^{spi}(\llbracket \mathit{case}\ L\ \mathit{of}\ \{\!|x|\!\}_N\ \mathit{in}\ P\ \mathit{else}\ Q\rrbracket)\ \rho\ \phi_{\mathcal{A}} =$$

$$\begin{cases} \bigcup_{\substack{\phi \\ \{\!|M^t|\!\}_{n^+}\in\varphi_{\mathcal{A}}(\phi_{\mathcal{A}},L)}} \mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \rho\rrbracket)\ \phi'_{\mathcal{A}}, \quad \text{if, } n^- \in \varphi_{\mathcal{A}}(\phi_{\mathcal{A}},N) \\ \text{where, } \phi'_{\mathcal{A}} = \phi_{\mathcal{A}}[\alpha_{k,k'}(x) \mapsto \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) \cup \{\alpha_{k,k'}(t)\}] \\ \mathcal{R}^{spi}(\llbracket \{\!|Q|\!\}_\rho \uplus_\rho \rho\rrbracket)\ \phi_{\mathcal{A}}, \qquad \text{otherwise} \end{cases}$$

$$(\mathcal{A}^{spi}11)\quad \mathcal{A}^{spi}(\llbracket \mathit{case}\ L\ \mathit{of}\ [\![x]\!]_N\ \mathit{in}\ P\ \mathit{else}\ Q\rrbracket)\ \rho\ \phi_{\mathcal{A}} =$$

$$\begin{cases} \bigcup_{\substack{\phi \\ [\![M^t]\!]_{n^-}\in\varphi_{\mathcal{A}}(\phi_{\mathcal{A}},L)}} \mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \rho\rrbracket)\ \phi'_{\mathcal{A}}, \quad \text{if, } n^+ \in \varphi_{\mathcal{A}}(\phi_{\mathcal{A}},N) \\ \text{where, } \phi'_{\mathcal{A}} = \phi_{\mathcal{A}}[\alpha_{k,k'}(x) \mapsto \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) \cup \{\alpha_{k,k'}(t)\}] \\ \mathcal{R}^{spi}(\llbracket \{\!|Q|\!\}_\rho \uplus_\rho \rho\rrbracket)\ \phi_{\mathcal{A}}, \qquad \text{otherwise} \end{cases}$$

$$(\mathcal{R}^{spi}0)\quad \mathcal{R}^{spi}(\llbracket \rho\rrbracket)\ \phi_{\mathcal{A}} \;=\; \bigcup_{\substack{\phi \\ P\in\rho}} \mathcal{A}^{spi}(\llbracket P\rrbracket)\ (\rho\backslash\{\!|P|\!\}_\rho)\ \phi_{\mathcal{A}}$$

Figure 4.4: The abstract semantics of the spi calculus (continued).

The description of the rules is as follows. Rules ($\mathcal{A}^{spi}1$) and ($\mathcal{A}^{spi}2$) return the $\phi_{\mathcal{A}}$ environment unchanged. Communications are dealt with in rule ($\mathcal{A}^{spi}3$) for output actions, where synchronising output and input channels yield a communication, in which the tag of the message is captured by $\phi_{\mathcal{A}}$. The semantics is imprecise, since $\phi_{\mathcal{A}}$ only captures an abstract tag as a value for an abstract variable. Rules ($\mathcal{A}^{spi}4$) and ($\mathcal{A}^{spi}5$) deal with the cases of restriction and parallel composition directly by placing the subprocesses with the rest in ρ.

The rule for replication, ($\mathcal{A}^{spi}6$), performs a least fixed point calculation using a special function, $\mathcal{F}^{spi} : \mathbb{N} \to D^{\sharp}_{\perp}$. This least fixed point occurs at the minimum number, n, such that $\mathcal{A}^{spi}(\![\prod_{i=1}^{n} ren(P,i)]\!)\ \rho\ \phi_{\mathcal{A}} = \mathcal{A}^{spi}(\![\prod_{i=1}^{n+2} ren(P,i)]\!)\ \rho\ \phi_{\mathcal{A}}$. The termination property of this calculation is stated formally in the following theorem.

Theorem 7 (Termination of the least fixed-point calculation)
The calculation of rule ($\mathcal{A}^{spi}6$) *terminates.*

Proof. To prove the termination property, it is necessary to satisfy two requirements. First, the semantic domain must be finite. This is satisfied by the definition of $D^{\sharp}_{\perp}$. The second requirement is to prove the monotonicity of $\mathcal{A}^{spi}(\![\prod^{n} P]\!)\ \rho\ \phi_{\mathcal{A}}$, i.e. $\mathcal{A}^{spi}(\![\prod^{n} P]\!)\ \rho\ \phi_{\mathcal{A}} \sqsubseteq \mathcal{A}^{spi}(\![\prod^{n+2} P]\!)\ \rho\ \phi_{\mathcal{A}}$. To prove this, we simplify the inequality into $\mathcal{A}^{spi}(\![Q]\!)\ \rho\ \phi_{\mathcal{A}} \sqsubseteq \mathcal{A}^{\pi}(\![Q \mid P]\!)\ \rho\ \phi_{\mathcal{A}}$, where $Q = \prod^{n} P$. This is further simplified to become $\mathcal{A}^{spi}(\![Q]\!)\ \rho\ \phi_{\mathcal{A}} \sqsubseteq \mathcal{A}^{\pi}(\![Q]\!)\ \rho'\ \phi_{\mathcal{A}}$, where $\rho' = \rho \uplus_{\rho} \{\!|P|\!\}_{\rho}$. This can be proven by induction over $\mathcal{A}^{spi}(\![P]\!)\ \rho\ \phi_{\mathcal{A}}$. In particular, the most interesting cases are rules ($\mathcal{A}^{spi}3$) and ($\mathcal{A}^{spi}8$)–($\mathcal{A}^{spi}11$), where $\phi_{\mathcal{A}}$ changes. For example, in rule ($\mathcal{A}^{spi}3$), we have that since $\rho \subseteq \rho'$, then $M'(y).P' \in \rho \Rightarrow M'(y).P' \in \rho'$. From this we can conclude that $\mathcal{A}^{spi}(\![Q]\!)\ \rho\ \phi_{\mathcal{A}} \sqsubseteq \mathcal{A}^{spi}(\![Q]\!)\ \rho'\ \phi_{\mathcal{A}}$, since the environment resulting from $\mathcal{A}^{spi}(\![Q]\!)\ \rho\ \phi_{\mathcal{A}}$ will necessarily be a subset of the environment resulting from $\mathcal{A}^{spi}(\![Q]\!)\ \rho'\ \phi_{\mathcal{A}}$ (i.e. the larger system induces more term substitutions). □

The rule for replication also uses the labelling mechanism to α-convert the set of bound names and variables of each copy of the replication, $!P$, as well as its set of tags. This renaming does not affect the compositionality of the semantics. The rule for conditional processes, ($\mathcal{A}^{spi}7$), relies on the equality of two untagged terms under $\phi_{\mathcal{A}}$. If in the case that the equality does not hold, a different alternative process is chosen. The rule for tuple splitting, ($\mathcal{A}^{spi}8$), attempts to split elements of a set of tuples corresponding to the value of $\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, L)$ of a term, L. The $\phi_{\mathcal{A}}$ environment is updated with the tags of the elements of each tuple. In case no tuples exist in the set, an alternative process is chosen and $\phi_{\mathcal{A}}$ is left unchanged. The rest of the rules, ($\mathcal{A}^{spi}9$)–($\mathcal{A}^{spi}11$), deal with cryptographic processes. Again, a process attempts to decipher (verify) a term, L, closed by $\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, L)$. The tags of the deciphered plaintexts are added to $\phi_{\mathcal{A}}$. Else a different process is chosen without affecting

$\phi_{\mathcal{A}}$. Finally, rule ($\mathcal{R}^{spi}0$) groups all the environments resulting from the interpretation of processes in ρ with the union of environments operation, $\cup_{\phi}$.

We restate here the safety of the $\cup_{\phi}$ operation for the case of the spi calculus (formalised earlier for the case of the π-calculus in Section 4.2.2).

Lemma 2 (Safety of $\bigcup_{\phi}$ in the spi calculus)

$\forall i \in \{1 \ldots n\}, n \in \mathbb{N}, \phi_i \in D_{\perp}, \phi'_i \in D^{\sharp}_{\perp}$:

$(\phi = \bigcup_{\substack{\phi \\ i=1\ldots n}} \phi_i) \ \wedge \ (\phi' = \bigcup_{\substack{\phi \\ i=1\ldots n}} \phi'_i) \ \wedge$

$(\exists M \in \mathit{Term} : \varphi_{\mathcal{E}}(\phi_i, M) \in \phi_i(x) \ \Rightarrow \ \exists t \in \phi'_i(\alpha_{k,k'}(x)) : \mathit{value\_of}(\{t\}) = \{M'\} \ \wedge$

$\mathit{untag}(M') = (\forall x \in \mathit{bnv}(M) : M[\alpha_{k,k'}(x)/x]))$

$\Rightarrow \quad (\exists M \in \mathit{Term} : \varphi_{\mathcal{E}}(\phi, M) \in \phi(x) \ \Rightarrow \ \exists t \in \phi'(\alpha_{k,k'}(x)) : \mathit{value\_of}(\{t\}) = \{M'\} \ \wedge$

$\mathit{untag}(M') = (\forall x \in \mathit{bnv}(M) : M[\alpha_{k,k'}(x)/x]))$

Proof. Refer to Appendix A.3. □

From this result, we can state the safety of the abstract semantics by the following theorem.

Theorem 8 (Safety of the abstract semantics for the spi calculus)

$\forall P, \rho, \phi_{\mathcal{E}}, \phi_{\mathcal{A}}$:

$(\mathcal{E}^{spi}(\llbracket P \rrbracket) \ \rho \ \phi_{\mathcal{E}} = (p, \phi'_{\mathcal{E}})) \ \wedge \ (\mathcal{A}^{spi}(\llbracket P \rrbracket) \ \rho \ \phi_{\mathcal{A}} = \phi'_{\mathcal{A}}) \ \wedge$

$(\exists M \in \mathit{Term} : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) \in \phi_{\mathcal{E}}(x) \ \Rightarrow \ \exists t \in \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) : \mathit{value\_of}(\{t\}) = \{M'\} \ \wedge$

$\mathit{untag}(M') = (\forall y \in \mathit{bnv}(M) : M[\alpha_{k,k'}(y)/y]))$

$\Rightarrow \quad (\exists M \in \mathit{Term} : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, M) \in \phi'_{\mathcal{E}}(x) \ \Rightarrow \ \exists t \in \phi'_{\mathcal{A}}(\alpha_{k,k'}(x)) : \mathit{value\_of}(\{t\}) = \{M'\} \ \wedge$

$\mathit{untag}(M') = (\forall y \in \mathit{bnv}(M) : M[\alpha_{k,k'}(y)/y]))$

Proof. Refer to Appendix A.4. □

The theorem states that for any term, M, captured in the non-standard semantics by including its $\varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, M)$ value in the value of a variable, $\phi'_{\mathcal{E}}(x)$, then that will correspond to capturing a tag, t, in the abstract semantics, by $\phi'_{\mathcal{A}}(\alpha_{k,k'}(x))$. The appropriateness of t is expressed by the ability to obtain an abstract form, $\forall x \in \mathit{bnv}(M) : M[\alpha_{k,k'}(x)/x]$, of the concrete term, M, by evaluating t using *value_of* and untagging the resulting term, M', using *untag*. More concisely, every concrete term, M, captured in the non-standard semantics is also captured in the form of the corresponding abstract tag, t, in the abstract semantics.

4.3.3 The Intruder I

The specification of the intruder in the spi calculus is inspired by the model presented by Dolev and Yao [47]. The model describes the general guidelines along which the most general attacker in cryptographic protocols can be specified. This model was shown by [40] to be sufficient to subsume any other adversary and the specification is dependent on the language of choice. In this section, we specify the most general attacking process in the spi calculus. Informally, any such specification should adhere to the following criteria:

- The attacker can read, learn, modify and block any messages passed over the network's public channels, as well as create fresh messages. It can also send the messages it has in its knowledge to other processes.
- The attacker can compose tuples from learnt messages and can decompose learnt tuples to their basic elements.
- The attacker can apply cryptographic operations, such as encryption, decryption etc. to any of the messages it has in its knowledge using any of the keys it knows about. The cryptographic model of the spi calculus coincides with Dolev-Yao's cryptographic abstraction as both assume perfect cryptography.

The above features can be stated more formally in the spi calculus by the specification of Figure 4.5. The specification contains the subprocess $\overline{i}\langle\kappa_{init}\rangle$, which initialises the knowledge of the intruder by setting $\kappa_{init} = \{M_n, \ldots, M_0\}$, where for the analysed process P running in parallel with I, we have that $fn(P) = \{M_n, \ldots, M_0\}$. Moreover, we refer to the set of names obtained by κ during the interpretation as $\phi_{\mathcal{A}}(\kappa)$. The knowledge of the intruder, κ, is increased due to the message-passing behaviour whenever input actions occur or fresh data are created as part of bound output actions. κ also increases due to the message-processing behaviour whenever decryption, signature verification or tuple-splitting operations succeed. In both cases, standard union is performed between κ and the extra term.

Apart from the initialisation process $\overline{i}\langle\kappa_{init}\rangle$, the rest of the specification consists of a replication of processes each of which is guarded by an input action, $i(\kappa)$, over the special channel i. The input parameter κ is instantiated with sets of terms. This is necessary to be able to express the fact that I can learn from its own behaviour. For example, in order for κ to obtain the new name *net* without necessarily outputting *net* to external processes, I sends *net* over channel i. Similarly, in order for κ to learn all the terms it has encrypted, signed etc., it needs to send them again over channel i. On the other hand, the main body of the process consists of the parallel composition of all the possible input/output actions

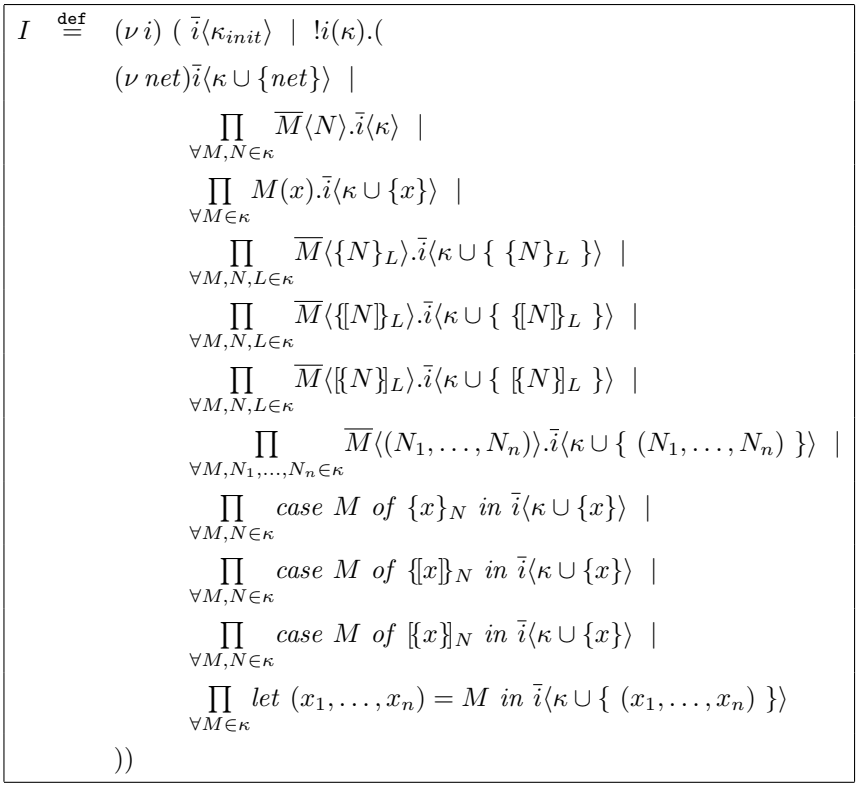

Figure 4.5: Specification of the Dolev-Yao attacker in the spi calculus.

and cryptographic operations quantified over all the terms currently in κ.

4.3.4 The Needham-Schroeder Public-Key Protocol Example

The Needham-Schroeder public key authentication protocol [100] aims at establishing authentication between two entities, an initiator, *Init*, and a responder, *Resp*. The protocol can be described by the following reduced sequence of messages using nonces N, N' [87], which assume that both agents have each other's public keys K^+_{Init} and K^+_{Resp}, beforehand:

Message 1	$Init \rightarrow Resp:$	$Init, Resp, \{N, Init\}_{K^+_{Resp}}$	on c_{Resp}
Message 2	$Resp \rightarrow Init:$	$Resp, Init, \{N, N'\}_{K^+_{Init}}$	on c_{Init}
Message 3	$Init \rightarrow Resp:$	$Init, Resp, \{N'\}_{K^+_{Resp}}$	on c_{Resp}

After which *Init* and *Resp* have authenticated each other's identities. We also include Messages 4 and 5 to indicate the mutual trust established after the protocol:

Message 4	$Resp \rightarrow Init:$	$\{M', N\}_{N'}$	on c_{Init}
Message 5	$Init \rightarrow Resp:$	$\{M, N'\}_N$	on c_{Resp}

Where M' and M are messages created by *Resp* and *Init* respectively. Here, *Init* fully trusts that M' is from *Resp* and similarly, *Resp* fully trusts that M is from *Init*. Nonces N' and N can be used as keys since these are names. This is done in order to indicate the importance of these nonces for the security of messages following the completion of the protocol.

The specification of the Needham-Schroeder protocol in the spi calculus is given in Figure 4.6, where the continuation processes of *Init* and *Resp* are denoted by F and F', respectively (F and F' are arbitrary dummy values and do not play any role in the analysis). In this specification, we have used non-recursive definitions $Init(X, Y)$ and $Resp(X, Y)$ to describe the behaviours of the protocol initiator and responder participants. Here, X is an agent variable representing the identity of the initiator and Y is an agent variable representing the identity of the responder. Both X and Y may be instantiated by agent names $A, B, C \ldots$. Hence, $Init\langle A, B\rangle \equiv Init[A/X, B/Y]$ indicates that the initiator is agent A and it is initiating the protocol to agent B and $Resp\langle A, B\rangle \equiv Resp[A/X, B/Y]$ to indicate that the responder is agent B and it is expected to respond to agent A. Note that the specification of the responder performs a matching check between variable u'_{XY} and the supplied parameter X to determine whether or not it is the expected initiator. If not, the responder halts, else it continues with the protocol. Variable u'_{XY} will be instantiated to the name of the initiator as included in the Message 1.

It is important to distinguish at this point between the concept of a *role* and that of an *agent*. *Init* and *Resp* are roles that can be played by the same agent or by different agents. In general, we assume from now on the presence of two honest agents, A and B. In addition to these agents, the intruder I exists and it is specified as in Section 4.3.3 by the Dolev-Yao model. Agents A and B are *honest* in the sense that they can assume no other specification apart from the $Init(X, Y)$ and $Resp(X, Y)$ processes. For example, agent A can act as *Init* in one session and as *Resp* in the next. As a result, it is necessary to include all the possible combinations involving two agents A and B as is done in the specification of the *Protocol* process. Additionally, when A (or B) acts as the initiator or the responder of the protocol (playing the *Init* or *Resp* role), it has the option of communicating with the intruder I. Therefore, the specification must allow for this possibility equally as well by including the options $Init\langle A, I\rangle$, $Init\langle B, I\rangle$, $Resp\langle I, A\rangle$ and $Resp\langle I, B\rangle$.

Assuming the intruder I has initially the knowledge $\kappa_{init} = \{A, B, I, K_I^+, K_I^-, K_A^+, K_B^+, c_A, c_B, c_I, \}$, we perform the abstract interpretation by applying $\mathcal{A}^{spi}([\![Protocol]\!])\ \{\!|\ |\!\}_\rho\ \phi_{\mathcal{A}0}$ for $\alpha_{1,1}$ (uniform analysis). The least fixed point values for $untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, x))$ for some of the variables, x, are shown in Figure 4.7. Since the definitions of security properties in the next chapter rely on names only, we only show these. Also, underlining is used to draw the

$$
\begin{array}{lll}
Init(X,Y) & \stackrel{\text{def}}{=} & (\nu\, N_{XY})(\nu\, M_{XY})\ \ (\\
 & & \overline{c_Y}\langle X, Y, \{\!|N_{XY}, X|\!\}_{K_Y^+}\rangle.c_X(x_{XY}). \\
 & & let\ (u_{XY}, y_{XY}, z_{XY}) = x_{XY}\ in \\
 & & case\ z_{XY}\ of\ \{\!|w_{XY}|\!\}_{K_X^-}\ in \\
 & & let\ (r_{XY}, v_{XY}) = w_{XY}\ in \\
 & & if\ (r_{XY} = N_{XY})\ then \\
 & & \overline{c_Y}\langle X, Y, \{\!|v_{XY}|\!\}_{K_Y^+}\rangle.c_X(msg1''_{XY}). \\
 & & case\ msg1''_{XY}\ of\ \{msg1'_{XY}\}_{v_{XY}}\ in \\
 & & let\ (msg1_{XY}, t_{XY}) = msg1'_{XY}\ in \\
 & & if\ (t_{XY} = N_{XY})\ then \\
 & & \overline{c_Y}\langle\{M_{XY}, v_{XY}\}_{N_{XY}}\rangle.F(msg1_{XY})) \\
 & & \\
Resp(X,Y) & \stackrel{\text{def}}{=} & (\nu\, N'_{XY})(\nu\, M'_{XY})\ \ (\\
 & & c_Y(x'_{XY}).let\ (u'_{XY}, y'_{XY}, z'_{XY}) = x'_{XY}\ in \\
 & & if\ (u'_{XY} = X)\ then \\
 & & case\ z'_{XY}\ of\ \{\!|w'_{XY}|\!\}_{K_Y^-}\ in \\
 & & let\ (r'_{XY}, v'_{XY}) = w'_{XY}\ in \\
 & & if\ (v'_{XY} = X)\ then \\
 & & \overline{c_X}\langle Y, X, \{\!|r'_{XY}, N'_{XY}|\!\}_{K_X^+}\rangle.c_Y(t'_{XY}). \\
 & & let\ (o'_{XY}, f'_{XY}, e'_{XY}) = t'_{XY}\ in \\
 & & case\ e'_{XY}\ of\ \{\!|h'_{XY}|\!\}_{K_Y^-}\ in \\
 & & if\ (h'_{XY} = N'_{XY})\ then \\
 & & \overline{c_X}\langle\{M'_{XY}, r'_{XY}\}_{N'_{XY}}\rangle.c_Y(msg2''_{XY}). \\
 & & case\ msg2''_{XY}\ of\ \{msg2'_{XY}\}_{r'_{XY}}\ in \\
 & & let\ (msg2_{XY}, g'_{XY}) = msg2'_{XY}\ in \\
 & & if\ (g'_{XY} = N'_{XY})\ then\ F'(msg2_{XY})) \\
 & & \\
Protocol & \stackrel{\text{def}}{=} & (\nu\, K_A^-)\ !(Init\langle A, B\rangle\ \mid\ Init\langle A, I\rangle\ \mid\ Resp\langle B, A\rangle\ \mid\ Resp\langle I, A\rangle)\ \mid \\
 & & (\nu\, K_B^-)\ !(Init\langle B, A\rangle\ \mid\ Init\langle B, I\rangle\ \mid\ Resp\langle A, B\rangle\ \mid\ Resp\langle I, B\rangle)\ \mid \\
 & & (\nu\, K_I^-)(I)
\end{array}
$$

Figure 4.6: Specification of the Needham-Schroeder protocol.

attention of the reader to interesting values.

$$
\begin{aligned}
&\kappa \mapsto \{A, B, I, K_I^+, K_I^-, K_A^+, K_B^+, c_A, c_B, c_I,\\
&\qquad N_{AI1}, N_{BI1}, N'_{IA1}, N'_{IB1}, \underline{N'_{AB1}, N'_{BA1}},\\
&\qquad M_{AI1}, M_{BI1}, M'_{IA1}, M'_{IB1}, \underline{M'_{AB1}, M'_{BA1}}, net_1\}
\end{aligned}
$$

$$
\begin{array}{lll}
msg1_{AB1} \mapsto \{M'_{AB1}\} & r_{AB1} \mapsto \{N_{AB1}\} \cup \Delta & v_{AB1} \mapsto \{N'_{AB1}\} \cup \Delta \\
msg1_{BA1} \mapsto \{M'_{BA1}\} & r_{BA1} \mapsto \{N_{BA1}\} \cup \Delta & v_{BA1} \mapsto \{N'_{BA1}\} \cup \Delta \\
msg1_{AI1} \mapsto \{net_1\} & r_{AI1} \mapsto \{N_{AI1}\} \cup \Delta & v_{AI1} \mapsto \{net_1\} \cup \Delta \\
msg1_{BI1} \mapsto \{net_1\} & r_{BI1} \mapsto \{N_{BI1}\} \cup \Delta & v_{BI1} \mapsto \{net_1\} \cup \Delta \\
msg2_{AB1} \mapsto \{M_{AB1}\} \cup \underline{\Delta} & h'_{AB1} \mapsto \{N'_{AB1}\} \cup \Delta & r'_{AB1} \mapsto \{N_{AB1}\} \cup \Delta \\
msg2_{BA1} \mapsto \{M_{BA1}\} \cup \underline{\Delta} & h'_{BA1} \mapsto \{N'_{BA1}\} \cup \Delta & r'_{IB1} \mapsto \{net_1\} \cup \Delta \\
msg2_{IB1} \mapsto \{net_1\} & h'_{IA1} \mapsto \{N'_{IA1}\} \cup \Delta & r'_{BA1} \mapsto \{N_{BA1}\} \cup \Delta \\
msg2_{IA1} \mapsto \{net_1\} & h'_{IB1} \mapsto \{N'_{IB1}\} \cup \Delta & r'_{IA1} \mapsto \{net_1\} \cup \Delta
\end{array}
$$

where, $\Delta = untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, \kappa))$

Figure 4.7: Results of analysing the Needham-Schroeder protocol.

In these results, we find that nonce N'_{AB1} created by B in response to A and nonce N'_{BA1} created by A in response to B appear in the knowledge of the intruder κ along with the corresponding messages M'_{AB1} and M'_{BA1}. Although we only show the name subset of the final results, the intruder is also capable of learning any complex terms that can be constructed from this subset. Also, we find that both A and B when acting as responders in communication sessions involving themselves only, may accept messages and nonces from I as well. These interfering messages and nonces can be any value captured by the intruder.

The anomalous results correspond to the famous *man-in-the-middle* attack, first published and fixed by Lowe [87]. In this attack, the intruder I is capable of masquerading as the initiator to the responder in communication sessions involving honest agents A and B only. This will eventually cause the responder to reveal secret messages to I and will cause I to undermine the authenticity requirements of the responder.

The important case where the attack occurs is when the initiator of the protocol, which is an honest agent A, attempts to initiate a session with the intruder I, which then manipulates this session to initiate another session with the other honest agent B while impersonating A. At the end of the protocol, I convinces B that it is communicating with A. The following sequence of messages describes a particular instance of the attack. Two sessions a and b are

running in parallel. In a, A is the initiator and I is the responder, and in b, $I(A)$ is the initiator and B is the responder ($I(A)$ denotes I masquerading as A):

Message 1a	$A \rightarrow I:$	$A, I, \{N_1, A\}_{K_I^+}$	on c_I
Message 1b	$I(A) \rightarrow B:$	$A, B, \{N_1, A\}_{K_B^+}$	on c_B
Message 2b	$B \rightarrow I(A):$	$B, A, \{N_1, N_2\}_{K_A^+}$	on c_A
Message 2a	$I \rightarrow A:$	$I, A, \{N_1, N_2\}_{K_A^+}$	on c_A
Message 3a	$A \rightarrow I:$	$A, I, \{N_2\}_{K_I^+}$	on c_I
Message 3b	$I(A) \rightarrow B:$	$A, B, \{N_2\}_{K_B^+}$	on c_B
Message 4b	$B \rightarrow I(A):$	$\{M, N_1\}_{N_2}$	on c_A
Message 5b	$I(A) \rightarrow B:$	$\{M', N_2\}_{N_1}$	on c_B

The attack occurs due to the fact that the two nonces encrypted and returned by B in Message 2b bear no indication as to the identity of the initiator that B expects from the previous message 1b. Hence, the intruder is capable of using these nonces in the context of its communication with agent A. In the next chapter, we formalise the secrecy and authenticity properties of the protocol.

4.3.5 The SPLICE/AS Protocol Example

The SPLICE/AS protocol was first suggested by [134] as a public-key protocol that establishes authentication between two agents. The protocol was found flawed in [80], and two attacks were published that allowed the intruder to impersonate initiators and responders.

Here, we consider the modified version of the protocol as suggested in [80], where we have removed the messages dealing with the distribution of the public keys as in [42], and assumed that both the initiator and the responder have obtained each other's public keys in a secure manner. Then the resulting sequence of messages describes the protocol, where N is a nonce (we have further omitted timestamps):

Message 1	$Init \rightarrow Resp:$	$Init, Resp, \{Init, \{N\}_{K_{Resp}^+}\}_{K_{Init}^-}$	on c_{Resp}
Message 2	$Resp \rightarrow Init:$	$Resp, Init, \{Resp, N\}_{K_{Init}^+}$	on c_{Init}

These messages establish authentication between an initiator and a responder whenever the responder verifies successfully the digital signature created by the initiator in the first message and the initiator receives back its nonce from the responder in the second message. One may include Messages 3 and 4 to indicate that both the initiator and the responder are confident enough to exchange secret messages:

Message 3	$Init \rightarrow Resp:$	$\{M\}_N$	on c_{Resp}
Message 4	$Resp \rightarrow Init:$	$\{M'\}_N$	on c_{Init}

Where we have assumed that nonce N is used as a shared session key. Alternatively, a separate session key K could be created by the initiator and sent to the responder.

The specification of the modified SPLICE/AS protocol in the spi calculus is given in Figure 4.8. The continuation processes of *Init* and *Resp* are denoted by F and F', respectively. Here we have used non-recursive definitions to arrive at a simple specification.

$$
\begin{array}{lll}
Init(X,Y) & \stackrel{\text{def}}{=} & (\nu\, N_{XY})(\nu\, M_{XY})\ \ (\\
 & & \overline{c_Y}\langle X, Y, \{\!|X, \{\!|N_{XY}|\!\}_{K_Y^+}|\!\}_{K_X^-}\rangle. \\
 & & c_X(x_{XY}).let\ (u_{XY}, y_{XY}, z_{XY}) = x_{XY}\ in \\
 & & case\ z_{XY}\ of\ \{\!|w_{XY}|\!\}_{K_X^-}\ in \\
 & & let\ (r_{XY}, v_{XY}) = w_{XY}\ in \\
 & & if\ v_{XY} = N_{XY}\ then \\
 & & if\ r_{XY} = Y\ then \\
 & & \overline{c_Y}\langle\{M_{XY}\}_{v_{XY}}\rangle.c_X(msg1'_{XY}). \\
 & & case\ msg1'_{XY}\ of\ \{msg1_{XY}\}_{v_{XY}}\ in \\
 & & F(msg1_{XY}) \\
 & & \\
Resp(X,Y) & \stackrel{\text{def}}{=} & (\nu\, M'_{XY})\ \ (\\
 & & c_Y(x'_{XY}).let\ (u'_{XY}, y'_{XY}, z'_{XY}) = x'_{XY}\ in \\
 & & if\ u'_{XY} = X\ then \\
 & & case\ z'_{XY}\ of\ \{\!|w'_{XY}|\!\}_{K_X^+}\ in \\
 & & let\ (r'_{XY}, s'_{XY}) = w'_{XY}\ in \\
 & & case\ s'_{XY}\ of\ \{\!|v'_{XY}|\!\}_{K_Y^-}\ in \\
 & & \overline{c_X}\langle Y, X, \{\!|Y, v'_{XY}|\!\}_{K_X^+}\rangle.c_Y(msg2'_{XY}). \\
 & & case\ msg2'_{XY}\ of\ \{msg2_{XY}\}_{v'_{XY}}\ in \\
 & & \overline{c_X}\langle\{M'_{XY}\}_{v'_{XY}}.F(msg2_{XY}) \\
 & & \\
Protocol & \stackrel{\text{def}}{=} & (\nu\, K_A^-)\ !(Init\langle A,B\rangle \mid Init\langle A,I\rangle \mid Resp\langle B,A\rangle \mid Resp\langle I,A\rangle) \mid \\
 & & (\nu\, K_B^-)\ !(Init\langle B,A\rangle \mid Init\langle B,I\rangle \mid Resp\langle A,B\rangle \mid Resp\langle I,B\rangle) \mid \\
 & & (\nu\, K_I^-)(I)
\end{array}
$$

Figure 4.8: Specification of the SPLICE/AS protocol.

Assuming $\kappa_{init} = \{A, B, I, K_I^+, K_I^-, K_A^+, K_B^+, c_A, c_B, c_I\}$, we arrive at the least fixed point results of Figure 4.9 by applying the abstract interpretation $\mathcal{A}^{spi}(\llbracket Protocol \rrbracket)\ \{\!|\ |\!\}_\rho\ \phi_{\mathcal{A}0}$ with $\alpha_{1,1}$ (uniform analysis). Again, the results show the values for $untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, x))$ for some of the variables, x. The intruder is also capable of constructing further complex terms from the name subset shown in Figure 4.9.

$$\kappa \mapsto \{A, B, I, K_I^+, K_I^-, K_A^+, K_B^+, c_A, c_B, c_I,$$
$$N_{AI1}, N_{BI1}, \underline{N_{AB1}, N_{BA1}}, M_{AI1}, M_{BI1}, \underline{M_{AB1}, M_{BA1}}, M'_{IA1}, M'_{IB1}, net_1\}$$

$msg1_{AB1} \mapsto \{M'_{AB1}\} \cup \underline{untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, \kappa))}$	$v_{AB1} \mapsto \{N_{AB1}\} \cup untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, \kappa))$
$msg1_{BA1} \mapsto \{M'_{BA1}\} \cup \underline{untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, \kappa))}$	$v_{BA1} \mapsto \{N_{BA1}\} \cup untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, \kappa))$
$msg1_{AI1} \mapsto \{net_1\}$	$v_{AI1} \mapsto \{N_{AI1}\} \cup untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, \kappa))$
$msg1_{BI1} \mapsto \{net_1\}$	$v_{BI1} \mapsto \{N_{BI1}\} \cup untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, \kappa))$
$msg2_{AB1} \mapsto \{M_{AB1}\}$	$v'_{AB1} \mapsto \{N_{AB1}\}$
$msg2_{BA1} \mapsto \{M_{BA1}\}$	$v'_{BA1} \mapsto \{N_{BA1}\}$
$msg2_{IA1} \mapsto \{net_1\}$	$v'_{IB1} \mapsto \{net_1\}$
$msg2_{IB1} \mapsto \{net_1\}$	$v'_{IA1} \mapsto \{net_1\}$

Figure 4.9: Results of analysing the SPLICE/AS protocol.

Examining the results of the abstract interpretation, we find a few irregularities. The intruder was successful in capturing nonce N_{AB1} created by A as initiator to B and nonce N_{BA1} created by B as initiator to A. Messages M_{AB1} and M_{BA1} created in sessions between A and B were also captured. Additionally, we find that initiators, in communication sessions involving A and B only, have captured messages from, I, that can be any of I's names.

In the next chapter on security analysis, we explain these anomalous results in the light of an impersonation attack that is carried out by I on initiators in communication sessions involving agents A and B. Assuming that agent A is acting as the initiator and agent B as the responder ($I(B)$ denotes I masquerading as B), the following steps describe the attack:

Message 1a	$A \rightarrow I(B):$	$A, B, \{A, \{N\}_{K_B^+}\}_{K_A^-}$	on c_B
Message 1b	$I \rightarrow B:$	$I, B, \{I, \{N\}_{K_B^+}\}_{K_I^-}$	on c_B
Message 2b	$B \rightarrow I:$	$B, I, \{B, N\}_{K_I^+}$	on c_I
Message 2a	$I(B) \rightarrow A:$	$B, A, \{B, N\}_{K_A^+}$	on c_A
Message 3a	$A \rightarrow I(B):$	$\{M\}_N$	on c_B
Message 4a	$I(B) \rightarrow A:$	$\{M'\}_N$	on c_A

The main problem here is in the initiator's message (Message 1a), which lacks any indication, inside $\{N\}_{K_B^+}$, to the initiator's identity (A in this case). Further formalisation of the security breaches of this attack is given in the next chapter on security properties.

4.3.6 The Otway-Rees Protocol Example

The Otway-Rees protocol was introduced in [104] with the purpose of creating session keys without the use of timestamps. We review here a modified version of the protocol [35]:

Message 1	$Init \rightarrow Resp: \quad N_1, Init, Resp, \{N_1, Init, Resp\}_{K_{InitS}}$	on c_{Resp}
Message 2	$Resp \rightarrow S: \quad N_1, Init, Resp, \{N_1, Init, Resp\}_{K_{InitS}},$ $N_2, \{N_1, Init, Resp\}_{K_{RespS}}$	on c_S
Message 3	$S \rightarrow Resp: \quad N_1, \{K, N_1\}_{K_{InitS}}, \{K, N_2\}_{K_{RespS}}$	on c_{Resp}
Message 4	$Resp \rightarrow Init: \quad N_1, \{K, N_1\}_{K_{InitS}}$	on c_{Init}

Where the initiator and the responder establish a shared session key K with the aid of a server S with whom they share the long-term secret keys K_{InitS} and K_{RespS}, respectively. N_1 is a nonce created by the initiator and N_2 is a nonce created by the responder. We also add the follow-up messages to indicate that both entities use K as the session key:

Message 5	$Init \rightarrow Resp: \quad \{M\}_K$	on c_{Resp}
Message 6	$Resp \rightarrow Init: \quad \{M'\}_K$	on c_{Init}

The specification of the Otway-Rees protocol in the spi calculus is shown in Figure 4.10. F and F' are continuation processes of the initiator and the responder processes, respectively. The non-recursive definitions of *Init* and *Resp* simplify the specification and can be instantiated for any of the honest agents A and B. The initiator, in addition, requires an extra argument denoting the responder. The server is assumed to be trusted and keeps a secure database of the long-term secret keys it shares with different agents and the intruder.

Given the initial knowledge of the intruder is set to $\kappa_{init} = \{A, B, I, c_A, c_B, c_S, K_{IS}\}$, where K_{IS} is the long-term key shared between I and S, applying the abstract interpretation $\mathcal{A}^{spi}([\![Protocol]\!])\ \{\!|\ |\!\}_\rho\ \phi_{\mathcal{A}0}$ for $\alpha_{1,1}$ (uniform analysis) reveals the name subset results of Figure 4.11, where we have applied $untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, x))$ for some variables, x. In these results, we notice that the intruder was capable of capturing additional messages M_{AB1}, M_{BA1} created by A and B acting as initiators, and messages M'_{AB1}, M'_{BA1} created by B and A acting as responders. All these messages were created in protocol sessions involving A and B only. Also, we find that any of the names in the knowledge of the intruder may appear as values for the $msg1$ and $msg2$ input parameters in sessions involving A and B only. This implies

$$
\begin{array}{lcl}
\mathit{Init}(X,Y) & \stackrel{\mathtt{def}}{=} & (\nu\, N_{XY})(\nu\, M_{XY})\ \ (\\
 & & \overline{c_Y}\langle N_{XY}, X, Y, \{N_{XY}, X, Y\}_{K_{XS}}\rangle. \\
 & & c_X(x_{XY}).\mathit{let}\ (y_{XY}, z_{XY}) = x_{XY}\ \mathit{in} \\
 & & \mathit{case}\ z_{XY}\ \mathit{of}\ \{w_{XY}\}_{K_{XS}}\ \mathit{in} \\
 & & \mathit{let}\ (v_{XY}, u_{XY}) = w_{XY}\ \mathit{in} \\
 & & \mathit{if}\ u_{XY} = N_{XY}\ \mathit{then}\ \overline{c_Y}\langle\{M_{XY}\}_{v_{XY}}\rangle.c_X(msg2'_{XY}). \\
 & & \mathit{case}\ msg2'_{XY}\ \mathit{of}\ \{msg2_{XY}\}_{v_{XY}}\ \mathit{in}\ F(msg2_{XY})) \\
 & & \\
\mathit{Resp}(X,Y) & \stackrel{\mathtt{def}}{=} & (\nu\, N'_{XY})(\nu\, M'_{XY})\ \ (\\
 & & c_Y(h'_{XY}).\mathit{let}\ (w'_{XY}, g'_{XY}, z'_{XY}, u'_{XY}) = h'_{XY}\ \mathit{in} \\
 & & \mathit{if}\ g'_{XY} = X\ \mathit{then}\ \overline{c_S}\langle h'_{XY}, N'_{XY}, \{w'_{XY}, X, z'_{XY}\}_{K_{YS}}\rangle. \\
 & & c_Y(v'_{XY}).\mathit{let}\ (t'_{XY}, r'_{XY}, s'_{XY}) = v'_{XY}\ \mathit{in} \\
 & & \mathit{case}\ s'_{XY}\ \mathit{of}\ \{o'_{XY}\}_{K_{YS}}\ \mathit{in} \\
 & & \mathit{let}\ (f'_{XY}, e'_{XY}) = o'_{XY}\ \mathit{in} \\
 & & \mathit{if}\ e'_{XY} = N'_{XY}\ \mathit{then}\ \overline{c_X}\langle t'_{XY}, r'_{XY}\rangle.c_Y(msg1'_{XY}). \\
 & & \mathit{case}\ msg1'_{XY}\ \mathit{of}\ \{msg1_{XY}\}_{f'_{XY}}\ \mathit{in} \\
 & & \overline{c_X}\langle\{M'_{XY}\}_{f'_{XY}}\rangle.F'(msg1_{XY})) \\
 & & \\
S & \stackrel{\mathtt{def}}{=} & c_S(x_S).\mathit{let}\ (u_S, x, y, z_S, v_S, r_S) = x_S\ \mathit{in} \\
 & & (\nu\, K_{xy})\ \ \overline{c_y}\langle u_S, \{K_{xy}, u_S\}_{K_{xS}}, \{K_{xy}, v_S\}_{K_{yS}}\rangle \\
 & & \\
\mathit{Protocol} & \stackrel{\mathtt{def}}{=} & (\nu\, K_{AS})\ !(\mathit{Init}\langle A, B\rangle\ \mid\ \mathit{Init}\langle A, I\rangle\ \mid \\
 & & \qquad\qquad \mathit{Resp}\langle B, A\rangle\ \mid\ \mathit{Resp}\langle I, A\rangle\ \mid\ S)\ \ \mid \\
 & & (\nu\, K_{BS})\ !(\mathit{Init}\langle B, A\rangle\ \mid\ \mathit{Init}\langle B, I\rangle\ \mid \\
 & & \qquad\qquad \mathit{Resp}\langle A, B\rangle\ \mid\ \mathit{Resp}\langle I, B\rangle\ \mid\ S)\ \ \mid \\
 & & (\nu\, K_{IS})(I\ \mid\ !S)
\end{array}
$$

Figure 4.10: The specification of the Otway-Rees protocol.

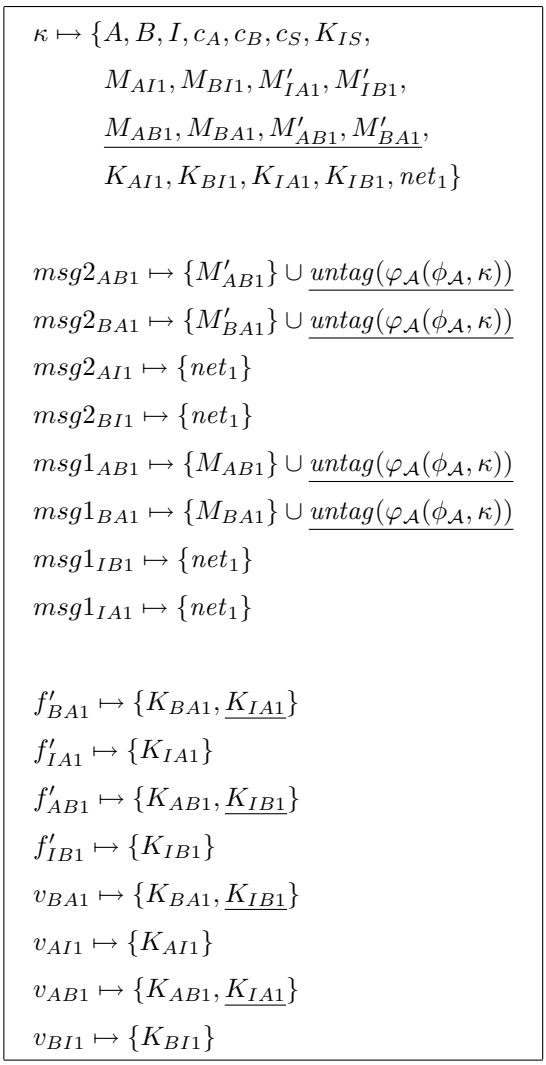

Figure 4.11: Results of analysing the Otway-Rees protocol.

that the intruder was successful in passing its knowledge to agents A and B in those sessions. On the other hand, the f'_{AB1}, f'_{BA1}, v_{AB1} and v_{BA1} input parameters will be instantiated with session keys received by B and A. These include the additional keys K_{IA1} and K_{IB1}.

The anomalous results above are due to the presence of two kinds of impersonation attacks carried out by the intruder, I. First, we discuss the attack by [106]:

Message 1a	$A \rightarrow I(B):\ N_1, A, B, \{N_1, A, B\}_{K_{AS}}$	on c_B
Message 1b	$I \rightarrow A:\ N'_1, I, A, \{N'_1, I, A\}_{K_{IS}}$	on c_A
Message 2b	$A \rightarrow I(S):\ N'_1, I, A, \{N'_1, I, A\}_{K_{IS}}, N'_2, \{N'_1, I, A\}_{K_{AS}}$	on c_S
Message 2c	$I(A) \rightarrow S:\ N'_1, I, A, \{N'_1, I, A\}_{K_{IS}}, N_1, \{N'_1, I, A\}_{K_{AS}}$	on c_S
Message 3b	$S \rightarrow I(A):\ N'_1, \{K, N'_1\}_{K_{IS}}, \{K, N_1\}_{K_{AS}}$	on c_A

Message 4a	$I(B) \rightarrow A: \ N_1, \{K, N_1\}_{K_{AS}}$	on c_A
Message 5a	$A \rightarrow I(B): \ \{M\}_K$	on c_B
Message 6a	$I(B) \rightarrow A: \ \{M'\}_K$	on c_A

Where $I(X)$ means I masquerading as X. The intruder I manages in this attack to completely isolate the initiator A by accessing and tampering with messages sent to and from A and with the aid of a parallel session that it initiates with A. I is capable of using S to create a bogus key K, which is then passed back to A for use with B (which plays no role and its part is masqueraded by I). The success of I in carrying out this attack is helped by nonce N_2' being sent without encryption in Message 2b, which I then replaces with the nonce created in the other session where A initiates communication with B.

The second attack we present here is a slightly modified version of the attack first published by Boyd and Mao in [28]. In their version, an assumption was made that the server should not be sensitive to the freshness of the nonces it receives (in other words, it cannot remember their history). This assumption can be relaxed in our version of the attack, which is described by the following sequence of messages:

Message 1a	$I(A) \rightarrow B: \ N_1, A, B, \{N_1, A, B\}_{K_{IS}}$	on c_B
Message 1b	$I \rightarrow B: \ N_1', I, B, \{N_1', I, B\}_{K_{IS}}$	on c_B
Message 2a	$B \rightarrow I(S): \ N_1, A, B, \{N_1, A, B\}_{K_{IS}}, N_2, \{N_1, A, B\}_{K_{BS}}$	on c_S
Message 2b	$B \rightarrow I(S): \ N_1', I, B, \{N_1', I, B\}_{K_{IS}}, N_2', \{N_1', I, B\}_{K_{BS}}$	on c_S
Message 2c	$I(B) \rightarrow S: \ N_1', I, B, \{N_1', I, B\}_{K_{IS}}, N_2, \{N_1', I, B\}_{K_{BS}}$	on c_S
Message 3b	$S \rightarrow I(B): \ N_1', \{K, N_1'\}_{K_{IS}}, \{K, N_2\}_{K_{BS}}$	on c_B
Message 3c	$I(S) \rightarrow B: \ N_1, \{K, N_1\}_{K_{IS}}, \{K, N_2\}_{K_{BS}}$	on c_B
Message 4b	$B \rightarrow I(A): \ N_1, \{K, N_1\}_{K_{IS}}$	on c_A
Message 5b	$I(A) \rightarrow B: \ \{M\}_K$	on c_B
Message 5b	$B \rightarrow I(A): \ \{M'\}_K$	on c_A

After which the responder B believes that K is a key shared with A, since it came encrypted along with the nonce N_2 that B created in the session responding to A (more precisely, $I(A)$). The reality, however, is that K is shared with I. Again, the key point in the success of I in playing the role of A lies in the sending of nonces N_2 and N_2' above without encryption in Messages 2a and 2b. This means that I can utilise this vulnerability to adjust the contents of those messages. Notice here that the server could not have encountered N_1' before, and so no assumption is made about its lack of memory as in [28]. In the next chapter, we formalise the secrecy and authenticity properties of the protocol.

4.3.7 The Kerberos Protocol Example

The Kerberos protocol was first introduced as part of project Athena [92] to offer a complete authentication solution. However, the intricacy of the implementation of the protocol rendered it far from being ideal in the real world. Here, we consider a simplified version of the Kerberos protocol that uses nonces N_1, N_2 instead of timestamps. The effect of using nonces is to prevent replay attacks that normally exploit the validity of timestamps. The protocol is described by the following sequence of messages [30]:

Message 1	$Init \rightarrow S:$	$Init, Resp$	on c_S
Message 2	$S \rightarrow Init:$	$\{N_1, K, Resp, \{N_1, K, Init\}_{K_{RespS}}\}_{K_{InitS}}$	on c_{Init}
Message 3	$Init \rightarrow Resp:$	$\{N_1, K, Init\}_{K_{RespS}}, \{Init, N_2\}_K$	on c_{Resp}
Message 4	$Resp \rightarrow Init:$	$\{N_2\}_K$	on c_{Init}

The aim of the protocol is to establish a session key K between two principals, the initiator, *Init*, and the responder, *Resp*, using a trusted server, S, with whom they share the long-term secret keys, K_{InitS} and K_{RespS}, respectively. The follow-up messages are included as usual:

Message 5	$Init \rightarrow Resp:$	$\{M\}_K$	on c_{Resp}
Message 6	$Resp \rightarrow Init:$	$\{M'\}_K$	on c_{Init}

The specifications of the protocol is given in Figure 4.12, where F and F' represent the dummy residual processes of *Init* and *Init*, respectively, which do not play any role in the protocol analysis.

The non-recursive definitions of *Init* and *Resp* can be instantiated for any of two honest agents A and B. Also, the server S is assumed to be trusted and keeps securely a database of long-term secret keys shared with agents A and B as well as the intruder I. By setting the knowledge of the intruder to $\kappa_{init} = \{A, B, I, c_A, c_B, c_S, K_{IS}\}$ and applying the abstract interpretation $\mathcal{A}^{spi}(\llbracket Protocol \rrbracket)\ \{\!|\ |\!\}_\rho\ \phi_{\mathcal{A}0}$ for $\alpha_{1,1}$ (uniform analysis), we obtain the name subset results shown in Figure 4.13, after converting these by applying $untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, x))$, for each variable, x. The results do not reveal any anomalies. The intruder is incapable of obtaining any knowledge beyond the sessions that it takes part in. The distribution of messages, keys and nonces to variables is as expected. In the next chapter, we discuss the secrecy and authenticity properties of this version of the Kerberos protocol.

$$
\begin{array}{lcl}
\mathit{Init}(X,Y) & \stackrel{\text{def}}{=} & (\nu\, N_{XY})(\nu\, M_{XY})\ \ (\\
 & & \overline{c_S}\langle X,Y\rangle.c_X(x_{XY}).\mathit{case}\ x_{XY}\ \mathit{of}\ \{y_{XY}\}_{K_{XS}}\ \mathit{in} \\
 & & \mathit{let}\ (v_{XY},u_{XY},w_{XY},z_{XY}) = y_{XY}\ \mathit{in} \\
 & & \overline{c_Y}\langle z_{XY},\{X,N_{XY}\}_{u_{XY}}\rangle.c_X(r_{XY}). \\
 & & \mathit{case}\ r_{XY}\ \mathit{of}\ \{t_{XY}\}_{u_{XY}}\ \mathit{in} \\
 & & \mathit{if}\ t_{XY} = N_{XY}\ \mathit{then}\ \overline{c_Y}\langle\{M_{XY}\}_{u_{XY}}\rangle.c_X(msg2'_{XY}). \\
 & & \mathit{case}\ msg2'_{XY}\ \mathit{of}\ \{msg2_{XY}\}_{u_{XY}}\ \mathit{in}\ F(msg2_{XY}) \\
 & & \\
\mathit{Resp}(X,Y) & \stackrel{\text{def}}{=} & (\nu\, M'_{XY})\ \ (\\
 & & c_Y(x'_{XY}).\mathit{let}\ (y'_{XY},z'_{XY}) = x'_{XY}\ \mathit{in} \\
 & & \mathit{case}\ y'_{XY}\ \mathit{of}\ \{o'_{XY}\}_{K_{YS}}\ \mathit{in} \\
 & & \mathit{let}\ (u'_{XY},w'_{XY},v'_{XY}) = o'_{XY}\ \mathit{in} \\
 & & \mathit{if}\ v'_{XY} = X\ \mathit{then}\ \mathit{case}\ z'_{XY}\ \mathit{of}\ \{f'_{XY}\}_{w'_{XY}}\ \mathit{in} \\
 & & \mathit{let}\ (r'_{XY},t'_{XY}) = f'_{XY}\ \mathit{in} \\
 & & \overline{c_X}\langle\{t'_{XY}\}_{w'_{XY}}\rangle.c_Y(msg1'_{XY}).\overline{c_X}\langle\{M'_{XY}\}_{w'_{XY}}\rangle. \\
 & & \mathit{case}\ msg1'_{XY}\ \mathit{of}\ \{msg1_{XY}\}_{w'_{XY}}\ \mathit{in}\ F'(msg1_{XY})) \\
 & & \\
S & \stackrel{\text{def}}{=} & c_S(x_S).\mathit{let}\ (x,y) = x_S\ \mathit{in} \\
 & & (\nu\, K_{xy})(\nu\, N)\overline{c_x}\langle\{N,K_{xy},y,\{N,K_{xy},x\}_{K_{yS}}\}_{K_{xS}}\rangle \\
 & & \\
\mathit{Protocol} & \stackrel{\text{def}}{=} & (\nu\, K_{AS})\ !(\mathit{Init}\langle A,B\rangle\ \mid\ \mathit{Init}\langle A,I\rangle\ \mid \\
 & & \qquad\qquad \mathit{Resp}\langle B,A\rangle\ \mid\ \mathit{Resp}\langle I,A\rangle\ \mid\ S)\ \mid \\
 & & (\nu\, K_{BS})\ !(\mathit{Init}\langle B,A\rangle\ \mid\ \mathit{Init}\langle B,I\rangle\ \mid \\
 & & \qquad\qquad \mathit{Resp}\langle A,B\rangle\ \mid\ \mathit{Resp}\langle I,B\rangle\ \mid\ S)\ \mid \\
 & & (\nu\, K_{IS})(I\ \mid\ !S)
\end{array}
$$

Figure 4.12: The specification of the Kerberos protocol.

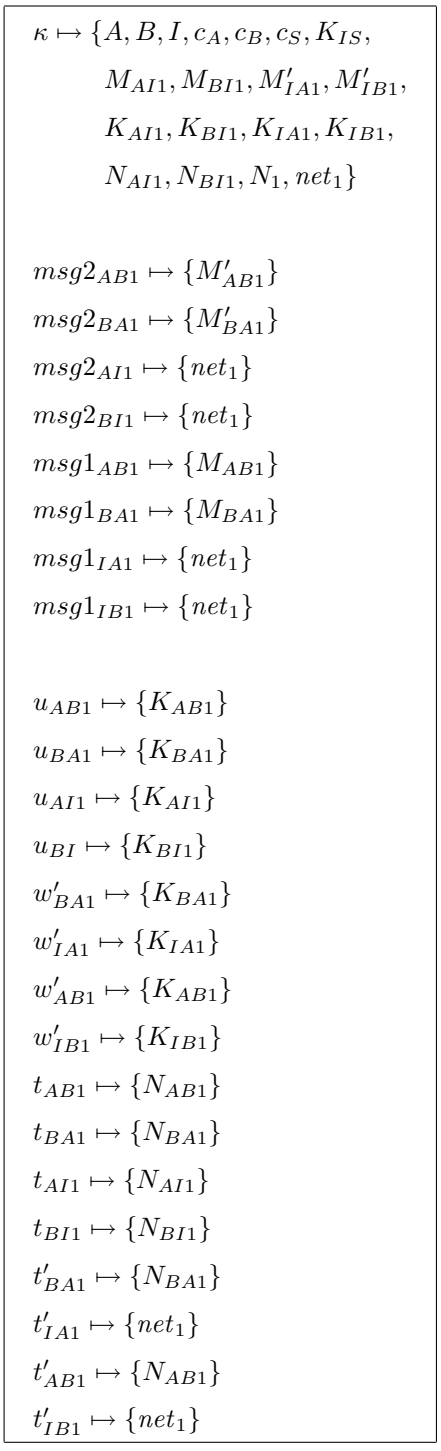

Figure 4.13: The results of analysing the Kerberos protocol.

4.3.8 The Yahalom Protocol Example

The Yahalom protocol first appeared in [35] with the usual set up of agents as before. Two participants, the initiator, *Init*, and the responder, *Resp*, establish a session key, K, with the aid of a trusted server, S, with whom they share the long-term secret keys, K_{InitS} and K_{RespS}, as indicated by the following sequence of messages:

Message 1	$Init \rightarrow Resp:$	$Init, N_1$	on c_{Resp}
Message 2	$Resp \rightarrow S:$	$Resp, \{Init, N_1, N_2\}_{K_{RespS}}$	on c_S
Message 3	$S \rightarrow Init:$	$\{Resp, K, N_1, N_2\}_{K_{InitS}}, \{Init, K\}_{K_{RespS}}$	on c_{Init}
Message 4	$Init \rightarrow Resp:$	$\{Init, K\}_{K_{RespS}}, \{N_2\}_K$	on c_{Resp}

Where N_1 and N_2 are nonces, created by the initiator and the responder, respectively. We may also include the follow-up messages as usual, whereby the initiator and the responder exchange a pair of (secret) messages encrypted with the session key created by the server in Message 3:

Message 5	$Resp \rightarrow Init:$	$\{M'\}_K$	on c_{Init}
Message 6	$Init \rightarrow Resp:$	$\{M\}_K$	on c_{Resp}

The specifications of the Yahalom protocol in the spi calculus is given in Figure 4.14. Here, F and F' are the continuations of the *Init* and *Resp* processes, respectively. S is a trusted server that keeps securely a database of long-term secret keys it shares with agents A and B as well as the intruder I. The *Protocol* process represents all the possible communications involving the two agents, A and B, as well as the intruder, I.

Assuming that the initial knowledge of the intruder is $\kappa_{init} = \{A, B, I, c_A, c_B, c_S, K_{IS}\}$, where K_{IS} is the key shared between the intruder and the server, then by applying the abstract interpretation $\mathcal{A}^{spi}(\llbracket Protocol \rrbracket)\ \{\!|\ |\!\}_\rho\ \phi_{\mathcal{A}0}$ for $\alpha_{1,1}$ (uniform analysis), we obtain a least fixed point value for $\phi_{\mathcal{A}}$, which after conversion by $untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, x))$, reveals the name subset results shown in Figure 4.15 for some variables, x. The result shown for κ reflects only the name part of the final value. More complex messages can also be composed by the intruder from the captured names.

These results appear to have correct distribution of session keys, messages and nonces to variables. Therefore, they do not reveal any anomalies. Further discussion of the secrecy and authenticity properties of the Yahalom protocol is included in the next chapter on security analysis.

$$
\begin{array}{lcl}
\mathit{Init}(X,Y) & \stackrel{\text{def}}{=} & (\nu\, N_{XY})(\nu\, M_{XY})\ \ (\\
 & & \overline{c_Y}\langle X, N_{XY}\rangle. \\
 & & c_X(x_{XY}).\mathit{let}\ (y_{XY}, z_{XY}) = x_{XY}\ \mathit{in} \\
 & & \mathit{case}\ y_{XY}\ \mathit{of}\ \{o_{XY}\}_{K_{XS}}\ \mathit{in} \\
 & & \mathit{let}\ (u_{XY}, v_{XY}, w_{XY}, r_{XY}) = o_{XY}\ \mathit{in} \\
 & & \mathit{if}\ w_{XY} = N_{XY}\ \mathit{then} \\
 & & \overline{c_Y}\langle z_{XY}, \{r_{XY}\}_{v_{XY}}\rangle.c_X(msg2'_{XY}). \\
 & & \overline{c_Y}\langle \{M_{XY}\}_{v_{XY}}\rangle. \\
 & & \mathit{case}\ msg2'_{XY}\ \mathit{of}\ \{msg2_{XY}\}_{v_{XY}}\ \mathit{in}\ F(msg2_{XY})) \\
 & & \\
\mathit{Resp}(X,Y) & \stackrel{\text{def}}{=} & (\nu\, N'_{XY})(\nu\, M'_{XY})\ \ (\\
 & & c_Y(z'_{XY}).\mathit{let}\ (x'_{XY}, y'_{XY}) = z'_{XY}\ \mathit{in} \\
 & & \mathit{if}\ x'_{XY} = X\ \mathit{then} \\
 & & \overline{c_S}\langle Y, \{X, y'_{XY}, N'_{XY}\}_{K_{YS}}\rangle. \\
 & & c_Y(u'_{XY}).\mathit{let}\ (s'_{XY}, t'_{XY}) = u'_{XY}\ \mathit{in} \\
 & & \mathit{case}\ s'_{XY}\ \mathit{of}\ \{w'_{XY}\}_{K_{YS}}\ \mathit{in} \\
 & & \mathit{let}\ (v'_{XY}, r'_{XY}) = w'_{XY}\ \mathit{in} \\
 & & \mathit{case}\ t'_{XY}\ \mathit{of}\ \{f'_{XY}\}_{r'_{XY}}\ \mathit{in} \\
 & & \mathit{if}\ f'_{XY} = N'_{XY}\ \mathit{then} \\
 & & \overline{c_X}\langle \{M'_{XY}\}_{r'_{XY}}\rangle.c_Y(msg1'_{XY}). \\
 & & \mathit{case}\ msg1'_{XY}\ \mathit{of}\ \{msg1_{XY}\}_{r'_{XY}}\ \mathit{in}\ F'(msg1_{XY})) \\
 & & \\
S & \stackrel{\text{def}}{=} & c_S(x_S).\mathit{let}\ (y, z_S) = x_S\ \mathit{in} \\
 & & \mathit{case}\ z_S\ \mathit{of}\ \{w_S\}_{K_{yS}}\ \mathit{in} \\
 & & \mathit{let}\ (x, r_S, t_S) = w_S\ \mathit{in} \\
 & & (\nu\, K_{xy})\ \ \overline{c_x}\langle \{y, K_{xy}, r_S, t_S\}_{K_{xS}}, \{x, K_{xy}\}_{K_{yS}}\rangle \\
 & & \\
\mathit{Protocol} & \stackrel{\text{def}}{=} & (\nu\, K_{AS})\ !(\mathit{Init}\langle A, B\rangle\ \mid\ \mathit{Init}\langle A, I\rangle\ \mid \\
 & & \qquad \mathit{Resp}\langle B, A\rangle\ \mid\ \mathit{Resp}\langle I, A\rangle\ \mid\ S)\ \ \mid \\
 & & (\nu\, K_{BS})\ !(\mathit{Init}\langle B, A\rangle\ \mid\ \mathit{Init}\langle B, I\rangle\ \mid \\
 & & \qquad \mathit{Resp}\langle A, B\rangle\ \mid\ \mathit{Resp}\langle I, B\rangle\ \mid\ S)\ \ \mid \\
 & & (\nu\, K_{IS})(I\ \mid\ !S)
\end{array}
$$

Figure 4.14: The specification of the Yahalom protocol.

$$\kappa \mapsto \{A, B, I, c_A, c_B, c_S, K_{IS}, M_{AI1}, M_{BI1}, M'_{IA1}, M'_{IB1},$$
$$K_{AI1}, K_{BI1}, K_{IA1}, K_{IB1}, N_{AI1}, N_{BI1}, N'_{IA1}, N'_{IB1}, net_1\}$$

$msg2_{AB1} \mapsto \{M'_{AB1}\}$	$v_{AB1} \mapsto \{K_{AB1}\}$	$w_{AB1} \mapsto \{N_{AB1}\}$
$msg2_{BA1} \mapsto \{M'_{BA1}\}$	$v_{BA1} \mapsto \{K_{BA1}\}$	$w_{BA1} \mapsto \{N_{BA1}\}$
$msg2_{AI1} \mapsto \{net_1\}$	$v_{AI1} \mapsto \{K_{AI1}\}$	$w_{AI1} \mapsto \{N_{AI1}\}$
$msg2_{BI1} \mapsto \{net_1\}$	$v_{BI1} \mapsto \{K_{BI1}\}$	$w_{BI1} \mapsto \{N_{BI1}\}$
$msg1_{AB1} \mapsto \{M_{AB1}\}$	$r'_{BA1} \mapsto \{K_{BA1}\}$	$f'_{BA1} \mapsto \{N'_{BA1}\}$
$msg1_{BA1} \mapsto \{M_{BA1}\}$	$r'_{IA1} \mapsto \{K_{IA1}\}$	$f'_{IA1} \mapsto \{N'_{IA1}\}$
$msg1_{IA1} \mapsto \{net_1\}$	$r'_{AB1} \mapsto \{K_{AB1}\}$	$f'_{AB1} \mapsto \{N'_{AB1}\}$
$msg1_{IB1} \mapsto \{net_1\}$	$r'_{IB1} \mapsto \{K_{IB1}\}$	$f'_{IB1} \mapsto \{N'_{IB1}\}$

Figure 4.15: The results of analysing the Yahalom protocol.

4.3.9 The Woo-Lam One-Way Authentication Protocol Example

A one-way authentication protocol was proposed in [133] using symmetric-key cryptography and nonces. The protocol aims at authenticating an initiator *Init* to a responder *Resp*, but not vice versa, using a trusted server S. This is achieved by the following modified sequence of messages, where N is a nonce, and we have included a session key K suggested by the initiator, *Init* for further communications with the responder, *Resp*:

Message 1	$Init \rightarrow Resp:$	$Init$	on c_{Resp}
Message 2	$Resp \rightarrow Init:$	N	on c_{Init}
Message 3	$Init \rightarrow Resp:$	$\{N, K\}_{K_{InitS}}$	on c_{Resp}
Message 4	$Resp \rightarrow S:$	$Resp, \{Init, \{N, K\}_{K_{InitS}}\}_{K_{RespS}}$	on c_S
Message 5	$S \rightarrow Resp:$	$\{N, K\}_{K_{RespS}}$	on c_{Resp}

After which the responder *Resp* is assured of the presence of the initiator *Init* but *Init* has no guarantees as to *Resp's* identity. The follow-up message indicates that *Resp* now trusts *Init*:

Message 6	$Resp \rightarrow Init:$	$\{M\}_K$	on c_{Init}

The above sequence of messages is specified in the spi calculus as in Figure 4.16, where F and F' are the continuation processes of the initiator and the responder, respectively. Setting the initial knowledge of the intruder to $\kappa_{init} = \{A, B, I, c_A, c_B, c_S, K_{IS}\}$, where K_{IS}

$$
\begin{array}{lll}
\mathit{Init}(X,Y) & \stackrel{\text{def}}{=} & (\nu\, K_{XY})\ \ (\\
 & & \overline{c_Y}\langle X\rangle.c_X(x_{XY}).\overline{c_Y}\langle\{x_{XY},K_{XY}\}_{K_{XS}}\rangle. \\
 & & c_X(msg1'_{XY}).\mathit{case}\ msg1'_{XY}\ \mathit{of}\ \{msg1_{XY}\}_{K_{XY}}\ \mathit{in} \\
 & & F(msg1_{XY})) \\
 & & \\
\mathit{Resp}(X,Y) & \stackrel{\text{def}}{=} & (\nu\, N'_{XY})(\nu\, M'_{XY})\ \ (\\
 & & c_Y(z'_{XY}).\mathit{if}\ z'_{XY}=X\ \mathit{then} \\
 & & \overline{c_X}\langle N'_{XY}\rangle. \\
 & & c_Y(x'_{XY}).\overline{c_S}\langle Y,\{X,x'_{XY}\}_{K_{YS}}\rangle \\
 & & c_Y(u'_{XY}).\mathit{case}\ u'_{XY}\ \mathit{of}\ \{w'_{XY}\}_{K_{YS}}\ \mathit{in} \\
 & & \mathit{let}\ (r'_{XY},s'_{XY})=w'_{XY}\ \mathit{in} \\
 & & \mathit{if}\ r'_{XY}=N'_{XY}\ \mathit{then} \\
 & & \overline{c_X}\langle\{M'_{XY}\}_{s'_{XY}}\rangle.F' \\
 & & \\
S & \stackrel{\text{def}}{=} & c_S(x_S).\mathit{let}\ (y,z_S)=x_S\ \mathit{in} \\
 & & \mathit{case}\ z_S\ \mathit{of}\ \{t_S\}_{K_{yS}}\ \mathit{in} \\
 & & \mathit{let}\ (x,u_S)=t_S\ \mathit{in} \\
 & & \mathit{case}\ u_S\ \mathit{of}\ \{r_{xy}\}_{K_{xS}}\ \mathit{in} \\
 & & \mathit{let}\ (v_{xy},w_{xy})=r_{xy}\ \mathit{in} \\
 & & \overline{c_y}\langle\{v_{xy},w_{xy}\}_{K_{yS}}\rangle \\
 & & \\
\mathit{Protocol} & \stackrel{\text{def}}{=} & (\nu\, K_{AS})\ !(\mathit{Init}\langle A,B\rangle\ \mid\ \mathit{Init}\langle A,I\rangle\ \mid \\
 & & \qquad\qquad \mathit{Resp}\langle B,A\rangle\ \mid\ \mathit{Resp}\langle I,A\rangle\ \mid\ S)\ \ \mid \\
 & & (\nu\, K_{BS})\ !(\mathit{Init}\langle B,A\rangle\ \mid\ \mathit{Init}\langle B,I\rangle\ \mid \\
 & & \qquad\qquad \mathit{Resp}\langle A,B\rangle\ \mid\ \mathit{Resp}\langle I,B\rangle\ \mid\ S)\ \ \mid \\
 & & (\nu\, K_{IS})(I\ \mid\ !S)
\end{array}
$$

Figure 4.16: The specification of the Woo-Lam protocol.

is the key shared between the intruder, I, and the server, S, and applying the abstract interpretation $\mathcal{A}^{spi}(\llbracket Protocol \rrbracket)\ \{\!|\ |\!\}_\rho\ \phi_{\mathcal{A}0}$ for $\alpha_{1,1}$ (uniform analysis), we obtain the results shown in Figure 4.17, where we have converted the final values of variables shown below, by applying $untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, x))$ to each variable, x.

$$\kappa \mapsto \{A, B, I, c_A, c_B, c_S, K_{IS},$$
$$N'_{AB0}, N'_{BA0}, N'_{IA0}, N'_{IB0}, M'_{IA0}, M'_{IB0}, \underline{M'_{AB0}, M'_{BA0}}, net_0\}$$

$msg1_{AB} \mapsto \{M'_{AB0}\}$	$s'_{BA} \mapsto \{K_{BA0}\} \cup \{\underline{untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, \kappa))}\}$
$msg1_{BA} \mapsto \{M'_{BA0}\}$	$s'_{IA} \mapsto \{net_0\}$
$msg1_{AI} \mapsto \{net_0\}$	$s'_{AB} \mapsto \{K_{AB0}\} \cup \{\underline{untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, \kappa))}\}$
$msg1_{BI} \mapsto \{net_0\}$	$s'_{IB} \mapsto \{net_0\}$
$x_{AB} \mapsto \{N'_{AB0}\} \cup untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, \kappa))$	$r'_{BA} \mapsto \{N'_{BA0}\}$
$x_{BA} \mapsto \{N'_{BA0}\} \cup untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, \kappa))$	$r'_{IA} \mapsto \{N'_{IA0}\}$
$x_{AI} \mapsto \{net_0\} \cup untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, \kappa))$	$r'_{AB} \mapsto \{N'_{AB0}\}$
$x_{BI} \mapsto \{net_0\} \cup untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, \kappa))$	$r'_{IB} \mapsto \{N'_{IB0}\}$

Figure 4.17: The results of analysing the Woo-Lam protocol.

The results of the analysis reveal some irregularities. We find that the input parameters s'_{AB} and s'_{BA} can obtain any of the names contained in the intruder's knowledge. The value of the intruder's knowledge κ also shows the additional message M'_{AB1} created by B and M'_{BA1} created by A. Both these messages were created in communications involving A and B only. Note that we do not consider the possibility of the intruder using any of the names of its knowledge to replace the nonce sent by the responder to the initiator in Message 2 as anomalous, since this message travels in the clear. Replacing would cause the responder not to accept the protocol.

In the next chapter, we explain an impersonation attack that causes the above anomalous results and discuss the resulting secrecy and authenticity implications. The intruder I makes use of a session it initiates with agent B to initiate another session with B but masquerading in the second session as agent A. The attack was first reported by [7] and a solution was suggested to avoid it. The following sequence of messages describes the scenario in which the attack occurs while agent B acting as the responder, where $I(A)$ means I masquerading as agent A:

Message 1a	$I(A) \rightarrow B:$	A	on c_B
Message 1b	$I \rightarrow B:$	I	on c_B
Message 2a	$B \rightarrow I(A):$	N	on c_A
Message 2b	$B \rightarrow I:$	N'	on c_I
Message 3a	$I(A) \rightarrow B:$	$\{N, K\}_{K_{IS}}$	on c_B
Message 3b	$I \rightarrow B:$	$\{N, K\}_{K_{IS}}$	on c_B
Message 4a	$B \rightarrow S:$	$\{A, \{N, K\}_{K_{IS}}\}_{K_{BS}}$	on c_S
Message 4b	$B \rightarrow S:$	$\{I, \{N, K\}_{K_{IS}}\}_{K_{BS}}$	on c_S
Message 5a	$S \rightarrow B:$	$\{C\}_{K_{BS}}$	on c_B
Message 5b	$S \rightarrow B:$	$\{N, K\}_{K_{BS}}$	on c_B
Message 6b	$B \rightarrow I(A):$	$\{M\}_K$	on c_A

Where C is the term resulting from the attempt to decrypt the ciphertext $\{N, K\}_{K_{IS}}$ using K_{AS}, which in the semantics of the spi calculus should result in the decryption *case*-statement being blocked since the decryption key K_{AS} is different from the encryption key K_{IS}. However, the server succeeds in decrypting the second ciphertext, $\{N, K\}_{K_{IS}}$, using K_{IS} after which it sends back the resulting term encrypted with K_{BS} to agent B. This leads to B wrongly believing that A is present and active and that it has suggested key K for further communications. The reason behind this wrong conclusion is the absence of any relation between B's request in Message 4 and the reply from the server S in Message 5.

4.4 Conclusion

In this chapter, we have presented abstract interpretations for processes in the π-calculus and the spi calculus that captures the property of term substitution. The meaning of a process in the abstract semantics is denoted as a mapping from the variables of that process to sets of tags denoting terms that could instantiate those variables at runtime. We demonstrated the safety of the abstract semantics with respect to the concrete non-standard semantics. The interpretation was applied to a few examples of systems specified in the π-calculus and the spi calculus. These included the file transfer protocol (FTP), the Needham-Schroeder (with public keys), SPLICE/AS, Otway-Rees, Kerberos, Yahalom and Woo-Lam protocols. The abstract interpretations used were uniform; only one copy of each new name, input parameter and tag in the analysed system were generated throughout the interpretation (due to the use of integer constraints $k = 1$ and $k' = 1$).

Chapter 5

Security Properties

5.1 Introduction

We define in this chapter security properties based on the results of the abstract interpretation established in the previous chapter. More precisely, we reason about term substitutions, and their secrecy and authenticity implications as a means of gaining information. A process classified at a low secrecy level could input data created by another high-level process, or could decrypt a ciphertext revealing the underlying sensitive plaintext if it has the appropriate key. Similarly, data authenticity could be compromised if an untrusted process succeeds in communicating its data to a highly trusted process. We explain these properties in light of the examples of the FTP server and cryptographic protocols of the previous chapter.

5.2 Secrecy

In our formalization of the secrecy property, we assume the presence of predefined multilevel security policies that are determined by the administrators of the systems under analysis. To express these policies, we assume a finite lattice of *secrecy levels*, $L = (S_L, \sqsubseteq, \sqcap, \sqcup, \top, \bot)$, ranged over by $l, l' \in S_L$ with $\bot_L$ being the undefined level and $\top_L$ being the most sensitive level. A security policy then attempts to classify processes according to their sensitivity by assigning to each process its appropriate level. We use the notation $\lceil P \rceil^l$ to express the fact that process P is classified (running) at secrecy level, l. The approach to process classification is flexible but requires that every subprocess within the main specification be assigned some secrecy level. Usually, the bottom level, $\bot_L$, is preserved for the intruder's specification as given in Section 4.2.3 (for mobile systems) and Section 4.3.3 (for cryptographic protocols).

This is the safest assumption to make about the intruder's secrecy property.

In the following, we discuss a variation of the secrecy property, termed *process leakage*. A process leakage occurs whenever a process classified at a low secrecy level explicitly obtains high-level data through one of its input parameters or variables. The meaning of *explicitly obtains* depends on the choice of language. In the π-calculus, a process obtains data solely through communications (instantiating input parameters). On the other hand, processes in the spi calculus obtain data both due to communications and as a result of the success of cryptographic operations (instantiating variables), like decryption, verification etc.

5.2.1 Mobile Systems

In order to be able to analyse the secrecy property in the π-calculus, we need to translate process classifications to name classifications. That is to say, given a process, $\lceil P \rceil^l$, running at secrecy level, l, we need to know the levels of its names, $n(P)$. Such translation requires a prior knowledge of the level of the intruder running in parallel with the analysed process. As we mentioned in the introduction, the safest level given to such an intruder is $\bot_L$.

For this purpose, we define an environment $\zeta : N \to L$ that maps names to their secrecy levels. Initially, ζ_0 maps every name to the bottom level. Hence:

$$\forall x \in N : \ \zeta_0(x) = \bot_L$$

We also define the union of ζ environments defined as follows:

$$\forall x \in N : \ (\zeta_1 \cup_\zeta \zeta_2)(x) = \zeta_1(x) \sqcup \zeta_2(x)$$

To obtain secrecy levels for the bound names of process, P, a special encoding function $\mathcal{Z} : \mathcal{P} \times (N \to A) \to (N \to A)$ is needed to construct the final value for ζ. This function is defined in Figure 5.1. The most interesting cases of the rules for ζ are those of input actions, $\lceil x(y).P \rceil^l$, and restriction, $\lceil (\nu\, y)P \rceil^l$, where the bound name, y, is assigned the level of its process, l. As we mentioned earlier, input parameters are treated in the process leakage property as the means by which processes gain data, whereas restricted names are treated as local data. Furthermore, the computation of $\mathcal{Z}(\lceil P \rceil^l)\ \zeta$ will terminate since the process on the right side of each rule is always a subprocess of the process on the left side.

For simplicity, we have chosen a uniform definition for the ζ function, in the sense that all the copies of bound names of a process are assigned a single level, which is the level of the root name appearing in the specification. A more flexible classification would be to assign different levels to different copies, or in other words, to allow the level of a name to change during runtime.

$$
\begin{array}{lcl}
\mathcal{Z}(\lceil \mathbf{0} \rceil^l)\ \zeta & = & \zeta \\
\mathcal{Z}(\lceil x(y).P \rceil^l)\ \zeta & = & \mathcal{Z}(P)\ \zeta[y \mapsto l] \\
\mathcal{Z}(\lceil \overline{x}\langle y \rangle.P \rceil^l)\ \zeta & = & \mathcal{Z}(P)\ \zeta \\
\mathcal{Z}(\lceil \mathit{if}\ x = y\ \mathit{then}\ P\ \mathit{else}\ Q \rceil^l)\ \zeta & = & \mathcal{Z}(P)\ \zeta\ \cup_\zeta\ \mathcal{Z}(Q)\ \zeta \\
\mathcal{Z}(\lceil P + Q \rceil^l)\ \zeta & = & \mathcal{Z}(P)\ \zeta\ \cup_\zeta\ \mathcal{Z}(Q)\ \zeta \\
\mathcal{Z}(\lceil P \mid Q \rceil^l)\ \zeta & = & \mathcal{Z}(P)\ \zeta\ \cup_\zeta\ \mathcal{Z}(Q)\ \zeta \\
\mathcal{Z}(\lceil (\nu\, y)P \rceil^l)\ \zeta & = & \mathcal{Z}(P)\ \zeta[y \mapsto l] \\
\mathcal{Z}(\lceil !P \rceil^l)\ \zeta & = & \mathcal{Z}(P)\ \zeta
\end{array}
$$

Figure 5.1: Rules of the $\mathcal{Z}(\lceil P \rceil^l)\ \zeta$ function for π-calculus processes.

The following predicate formalises the *process leakage* property. The predicate takes as input a process, P, analyzes it, and returns true whenever a high-level name, x, is *leaked* to some low-level input parameter, y (as a result of some communication).

Property 1 (Process leakage in the π-calculus)
A name, x, is said to be leaked within a process, P, if and only if the following holds true:
$\phi_{\mathcal{A}} = \mathcal{A}^{\pi}([\![P]\!])\ \rho_0\ \phi_{\mathcal{A}0}, \zeta = \mathcal{Z}(\lceil P \rceil^l)\ \zeta_0, \exists y \in dom(\phi_{\mathcal{A}}), \exists x \in \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, y) :\ \zeta(y) \sqsubseteq_L \zeta(x)$

Where ρ_0 may contain the intruder, I, in the case it is running in parallel with P. Hence, process leakage occurs whenever some low-level process manages to obtain the tag of a name, which was originally created by some other process with a higher secrecy level.

5.2.2 Cryptographic Protocols

The secrecy property of process leakage defined in the previous section for mobile systems is extended here to the case of cryptographic systems specified in the spi calculus. For this purpose, it is necessary to redefine the environment $\zeta : (V + N) \rightarrow L$. This is done by introducing the uniform function, $\mathcal{Z}(\lceil P \rceil^l)\ \zeta : \mathcal{P} \times ((V+N) \rightarrow L) \rightarrow ((V+N) \rightarrow L)$, as shown in Figure 5.2, which makes use of the union of ζ-environments, $\cup_\zeta$, defined in the previous section. Note that local variables holding the results of the decryption, signature verification and tuple-splitting processes obtain the levels of those processes. This is consistent with the manner in which input parameters are classified.

We now extend the predicate of Property 1 of the previous section to the spi calculus.

Property 2 (Process leakage in the spi calculus)
A name, a, is said to be leaked within a process, P, if and only if the following holds true:
$\phi_{\mathcal{A}} = \mathcal{A}^{spi}([\![P]\!])\ \rho_0 \phi_{\mathcal{A}0}, \zeta = \mathcal{Z}(\lceil P \rceil^l)\ \zeta_0, \exists y \in dom(\phi_{\mathcal{A}}), \exists a \in untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, y)) : \zeta(y) \sqsubseteq_L \zeta(a)$

$$
\begin{array}{lcl}
\mathcal{Z}(\lceil \mathbf{0} \rceil^l)\ \zeta & = & \zeta \\
\mathcal{Z}(\lceil M(y).P \rceil^l)\ \zeta & = & \mathcal{Z}(P)\ \zeta[y \mapsto l] \\
\mathcal{Z}(\lceil \overline{M}\langle N \rangle.P \rceil^l)\ \zeta & = & \mathcal{Z}(P)\ \zeta \\
\mathcal{Z}(\lceil (\nu\, a)P \rceil^l)\ \zeta & = & \mathcal{Z}(P)\ \zeta[a \mapsto l] \\
\mathcal{Z}(\lceil P \mid Q \rceil^l)\ \zeta & = & \mathcal{Z}(P)\ \zeta\ \cup_\zeta\ \mathcal{Z}(Q)\ \zeta \\
\mathcal{Z}(\lceil !P \rceil^l)\ \zeta & = & \mathcal{Z}(P)\ \zeta \\
\mathcal{Z}(\lceil \mathit{if}\ M = N\ \mathit{then}\ P\ \mathit{else}\ Q \rceil^l)\ \zeta & = & \mathcal{Z}(P)\ \zeta\ \cup_\zeta\ \mathcal{Z}(Q)\ \zeta \\
\mathcal{Z}(\lceil \mathit{let}\ (x_1, \ldots, x_n) = M\ \mathit{in}\ P\ \mathit{else}\ Q \rceil^l)\ \zeta & = & \mathcal{Z}(P)\ \zeta[x_1 \mapsto l, \ldots, x_n \mapsto l] \cup_\zeta \\
 & & \mathcal{Z}(Q)\ \zeta[x_1 \mapsto l, \ldots, x_n \mapsto l] \\
\mathcal{Z}(\lceil \mathit{case}\ L\ \mathit{of}\ \{x\}_N\ \mathit{in}\ P\ \mathit{else}\ Q \rceil^l)\ \zeta & = & \mathcal{Z}(P)\ \zeta[x \mapsto l]\ \cup_\zeta\ \mathcal{Z}(Q)\ \zeta[x \mapsto l] \\
\mathcal{Z}(\lceil \mathit{case}\ L\ \mathit{of}\ \{\!|x|\!\}_N\ \mathit{in}\ P\ \mathit{else}\ Q \rceil^l)\ \zeta & = & \mathcal{Z}(P)\ \zeta[x \mapsto l]\ \cup_\zeta\ \mathcal{Z}(Q)\ \zeta[x \mapsto l] \\
\mathcal{Z}(\lceil \mathit{case}\ L\ \mathit{of}\ [\{x\}]_N\ \mathit{in}\ P\ \mathit{else}\ Q \rceil^l)\ \zeta & = & \mathcal{Z}(P)\ \zeta[x \mapsto l]\ \cup_\zeta\ \mathcal{Z}(Q)\ \zeta[x \mapsto l]
\end{array}
$$

Figure 5.2: Rules of the $\mathcal{Z}(\lceil P \rceil^l)\ \zeta$ function for spi calculus processes.

The property captures instances where high-level names substitute low-level variables, which in turn, reflect the levels of their processes. Note that the property only captures the secrecy of names, as opposed to the secrecy of ciphertexts. This stems from our assumption that names are the only sensative data whose secrecy may be compromised. Ciphertexts provide a secure mechanism with which the secure transmission of names is achievable. For example, in the process $\overline{a}\langle k \rangle.\overline{a}\langle \{m\}_k \rangle \mid a(x).a(y).\mathit{case}\ y\ \mathit{of}\ \{z\}_x\ \mathit{in}\ P$, it is the secrecy of m, rather than $\{m\}_k$, that is undermined by P.

5.3 Authenticity

The treatment of the process leakage property in the previous section was made possible by using the notion of secrecy levels that distinguished the secrecy requirements of each process according to a well-defined security policy. In dealing with the authenticity property, the main notion of interest is that of process *trust level*. Trust levels, $a, a' \in S_A$, are ordered by a web of trust such that $A = (S_A, \sqsubseteq, \sqcap, \sqcup, \top, \bot)$ is a finite lattice with the bottom element, $\bot_A$, being the undefined level and the top element, $\top_A$, being the most trusted level. Examples of undefined trust levels include the levels of machines connected to the Internet, which belong to unknown entities. A well-trusted level, on the other hand, could be a Certification Authority (CA) in a Public-Key Infrastructure (PKI). The work of [23] represents one example in which mechanisms for implementing trust levels, called *addresses*, are provided as a primitive in an extension of the π-calculus language.

As a result of the presence of malicious attackers, a threat of process authenticity may occur in situations where a process obtains data that originated at a trust level lower than the level of the process itself. Intuitively, such threats are directly comparable to process leaks. However, with the process leakage property, it is the high-level data that are compromised by a secrecy breach (being obtained by a low-level process), whereas with the process authenticity property, it is the high-trust process that is compromised by an authenticity breach (obtaining the low-trust data). We emphasise here the direction of concern, which is different in each case.

In the following sections, we describe the process authenticity property in the π-calculus and spi calculus. We write, $\lfloor P \rfloor^a$, to express the fact that P is running at trust level, a. The requirement then is to translate process trust levels to name trust levels assuming that all subprocesses Q of P have the form, $\lfloor Q \rfloor^{a'}$, for some level a'.

5.3.1 Mobile Systems

In the π-calculus, we translate process trust levels to name trust levels using the environment, $\theta : N \to A$, which maps names to their levels. Initially, θ_0 maps every name to the bottom element, $a_I = \bot_A$, also assumed to be the level of the intruder, I:

$$\forall x \in N : \ \theta_0(x) = \bot_A$$

Additionally, we define the union of θ-environments, $\cup_\theta$, as follows:

$$(\theta_1 \cup_\theta \theta_2)(x) = \theta_1(x) \sqcup \theta_2(x)$$

This is similar to the union of ζ-environments defined in the previous section on secrecy. The use of the least upper bound operator $\sqcup$ is useful for adding any values of a name from θ_1 and θ_2 that are equal to $\bot_A$.

Defining a special encoding function, $\mathcal{U} : \mathcal{P} \times (N \to A) \to (N \to A)$, as in Figure 5.3, is necessary to construct θ from the specification of $\lfloor P \rfloor^a$. The rules of the $\mathcal{U}(\lfloor P \rfloor^a)\ \theta$ function are directly comparable to their secrecy counterpart given by the $\mathcal{Z}(\lceil P \rceil^a)\ \zeta$ function (in fact, the two functions only differ in whether they deal with secrecy or trust levels). The treatment of bound names is similar. These are assigned the trust levels of their process. In the case of restricted names, this reflects their locality, whereas in the case of input parameters, this reflects the fact they are used as means by which the residual process acquires further information (names). All the other cases do not affect the value of θ.

The following property formalises the process authenticity breach in the π-calculus.

$$
\begin{array}{lcl}
\mathcal{U}(\lfloor \mathbf{0} \rfloor^a)\ \theta & = & \theta \\
\mathcal{U}(\lfloor x(y).P \rfloor^a)\ \theta & = & \mathcal{U}(P)\ \theta[y \mapsto a] \\
\mathcal{U}(\lfloor \overline{x}\langle y \rangle.P \rfloor^a)\ \theta & = & \mathcal{U}(P)\ \theta \\
\mathcal{U}(\lfloor \mathit{if}\ x = y\ \mathit{then}\ P\ \mathit{else}\ Q \rfloor^a)\ \theta & = & \mathcal{U}(P)\ \theta \ \cup_\theta \ \mathcal{U}(Q)\ \theta \\
\mathcal{U}(\lfloor P + Q \rfloor^a)\ \theta & = & \mathcal{U}(P)\ \theta \ \cup_\theta \ \mathcal{U}(Q)\ \theta \\
\mathcal{U}(\lfloor P \mid Q \rfloor^a)\ \theta & = & \mathcal{U}(P)\ \theta \ \cup_\theta \ \mathcal{U}(Q)\ \theta \\
\mathcal{U}(\lfloor (\nu\, x)P \rfloor^a)\ \theta & = & \mathcal{U}(P)\ \theta[x \mapsto a] \\
\mathcal{U}(\lfloor !P \rfloor^a)\ \theta & = & \mathcal{U}(P)\ \theta
\end{array}
$$

Figure 5.3: Rules of the $\mathcal{U}(\lfloor P \rfloor^a)\ \theta$ function for π-calculus processes.

Property 3 (Process authenticity breach in the π-calculus)
The authenticity of a name, y, is said to be breached within a process, P, if and only if the following holds true:
$\phi_{\mathcal{A}} = \mathcal{A}^{\pi}(\llbracket P \rrbracket)\ \rho_0\ \phi_{\mathcal{A}0}, \theta = \mathcal{U}(\lfloor P \rfloor^a)\ \theta_0, \exists y \in dom(\phi_{\mathcal{A}}), \exists x \in \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, y) :\ \theta(y) \sqsupseteq \theta(x)$

Where ρ_0 may contain the intruder, I. When compared to the definition of the process leakage property (Property 1), the direction of the ordering relation clearly demonstrates the difference in concern. Intuitively, a process authenticity breach occurs whenever a tag, whose name value has a low trust level, instantiates another name with a higher trust level. Hence, we are concerned with highly trusted processes obtaining data not at the same level of trust. This could be as a result of that data originating from malicious sources.

5.3.2 Cryptographic Protocols

We extend here the process authenticity property given in the previous section to cryptographic protocols. A redefinition of the $\mathcal{U} : \mathcal{P} \times (N + V \to A) \to (N + V \to A)$ encoding function is necessary to construct the environment, $\theta : N + V \to A$, for processes in the spi calculus. The rules of this encoding function are given in Figure 5.4.

From the definition of $\mathcal{U}(\lfloor P \rfloor^a)\ \theta$, we can now redefine the process authenticity breach property for processes in the spi calculus as follows.

Property 4 (Process authenticity breach in the spi calculus)
The authenticity of a variable, y, is said to be breached within a process, P, if and only if the following holds true:
$\phi_{\mathcal{A}} = \mathcal{A}^{spi}(\llbracket P \rrbracket)\ \rho_0\ \phi_{\mathcal{A}0}, \theta = \mathcal{U}(\lfloor P \rfloor^a)\ \theta_0, \exists y \in dom(\phi_{\mathcal{A}}), \exists a \in untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, y)) :\ \theta(y) \sqsupseteq \theta(a)$

$$
\begin{array}{lcl}
\mathcal{U}(\lfloor \mathbf{0} \rfloor^a)\ \theta & = & \theta \\
\mathcal{U}(\lfloor M(y).P \rfloor^a)\ \theta & = & \mathcal{U}(P)\ \theta[y \mapsto a] \\
\mathcal{U}(\lfloor \overline{M}\langle N \rangle.P \rfloor^a)\ \theta & = & \mathcal{U}(P)\ \theta \\
\mathcal{U}(\lfloor (\nu\, a)P \rfloor^a)\ \theta & = & \mathcal{U}(P)\ \theta[a \mapsto a] \\
\mathcal{U}(\lfloor P \mid Q \rfloor^a)\ \theta & = & \mathcal{U}(P)\ \theta\ \cup_\theta\ \mathcal{U}(Q)\ \theta \\
\mathcal{U}(\lfloor !P \rfloor^a)\ \theta & = & \mathcal{U}(P)\ \theta \\
\mathcal{U}(\lfloor \mathit{if}\ M = N\ \mathit{then}\ P\ \mathit{else}\ Q \rfloor^a)\ \theta & = & \mathcal{U}(P)\ \theta\ \cup_\theta\ \mathcal{U}(Q)\ \theta \\
\mathcal{U}(\lfloor \mathit{let}\ (x_1, \ldots, x_n) = M\ \mathit{in}\ P\ \mathit{else}\ Q \rfloor^a)\ \theta & = & \mathcal{U}(P)\ \theta[x_1 \mapsto a, \ldots, x_n \mapsto a] \cup_\theta \\
 & & \mathcal{U}(Q)\ \theta[x_1 \mapsto a, \ldots, x_n \mapsto a] \\
\mathcal{U}(\lfloor \mathit{case}\ L\ \mathit{of}\ \{x\}_N\ \mathit{in}\ P\ \mathit{else}\ Q \rfloor^a)\ \theta & = & \mathcal{U}(P)\ \theta[x \mapsto a]\ \cup_\theta\ \mathcal{U}(Q)\ \theta[x \mapsto a] \\
\mathcal{U}(\lfloor \mathit{case}\ L\ \mathit{of}\ \{\!|x|\!\}_N\ \mathit{in}\ P\ \mathit{else}\ Q \rfloor^a)\ \theta & = & \mathcal{U}(P)\ \theta[x \mapsto a]\ \cup_\theta\ \mathcal{U}(Q)\ \theta[x \mapsto a] \\
\mathcal{U}(\lfloor \mathit{case}\ L\ \mathit{of}\ [\!\{x\}\!]_N\ \mathit{in}\ P\ \mathit{else}\ Q \rfloor^a)\ \theta & = & \mathcal{U}(P)\ \theta[x \mapsto a]\ \cup_\theta\ \mathcal{U}(Q)\ \theta[x \mapsto a]
\end{array}
$$

Figure 5.4: Rules of the $\mathcal{U}(\lfloor P \rfloor^a)\ \theta$ function for spi calculus processes.

Notice also that only names are captured and not digital signatures. This is due to the assumption that digital signatures only provide the means by which names (data) are transmitted in an authentic manner over public channels. For example, in the process $\overline{a}\langle \{\!|m|\!\}_{k^-} \rangle \mid a(x).\mathit{case}\ x\ \mathit{of}\ \{\!|y|\!\}_{k^+}\ \mathit{in}\ P$, if P expects a different name, m', to instantiate t, then it is m, rather than $\{\!|m|\!\}_{k^-}$, that breaks the authenticity requirement that P expects from y. Our treatment of the examples that follow enhances this view.

5.4 Examples

We revisit, in the following sections, the examples of the ftp server and the authentication protocols and discuss their secrecy and authenticity properties given the results of the abstract interpretation and the definitions of secrecy and authenticity presented earlier.

5.4.1 The FTP Server Example

The ftp server example we presented in Section 4.2.4 was analysed for two cases. The first case has a correct specification for the client process, which communicates with the server without interference from the intruder, whereas the second case has a faulty client specification that leaks its password to the intruder process, I. We analyse the authenticity and secrecy properties of both cases of the uniform and non-uniform abstract interpretations.

Secrecy. We adopt the following classification of secrecy levels:

$$
\begin{aligned}
ftp \stackrel{\text{def}}{=}\ & \lceil (!start(x).(\nu\, login)(if\ x = start\ then\ \lceil (\nu\, pwd)(Server \mid Client)\rceil^{l} \\
& \qquad\qquad\qquad\qquad else\ \lceil (\nu\, pwd)(Server)\rceil^{l} \mid \lceil I\rceil^{\perp_L}) \\
& \quad) \mid \overline{start}\langle start\rangle.\overline{start}\langle start\rangle.\overline{start}\langle start\rangle.\overline{start}\langle I\_start\rangle\rceil^{\perp_L} \\
Server \stackrel{\text{def}}{=}\ & (\nu\, deal)\ login(z).if\ z = pwd\ then \\
& \quad login(data).(\\
& \quad \overline{deal}\langle data\rangle \mid\ !deal(w).\overline{login}\langle w\rangle.login(u).if\ u = logout\ then\ \mathbf{0} \\
& \quad else\ \overline{deal}\langle u\rangle\ )\ \ else\ \mathbf{0}
\end{aligned}
$$

$$
Client \stackrel{\text{def}}{=}\ (\nu\, request)\ (\overline{login}\langle pwd\rangle.\overline{login}\langle request\rangle.login(res).\overline{login}\langle logout\rangle)
$$

With $\perp_L \sqsubseteq l$. By applying the encoding function $\mathcal{Z}$, it is possible to obtain the following value for the ζ environment, mapping each name to its secrecy level:

$$
\begin{aligned}
&\zeta(\kappa) = \zeta(x) = \zeta(net) = \zeta(login) = \zeta(logout) = \zeta(start) = \zeta(I\_start) = \perp_L \\
&\zeta(pwd) = \zeta(deal) = \zeta(request) = l
\end{aligned}
$$

The results of the uniform abstract interpretation (correct client specification) with α_1 indicate that the intruder is capable of capturing the *pwd* and *request* names, both of which have a higher secrecy level than κ. Refining these results further, we find that the non-uniform abstract interpretation, with α_4, reveals a correct distribution of names to input parameters in the final value for $\phi_{\mathcal{A}}$. In particular, the intruder's knowledge, κ, could not obtain any names with higher secrecy levels and remained limited to names *net*, *login* and *logout*. In general, $\forall x \in dom(\phi_{\mathcal{A}}), y \in \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, x) : \zeta(y) \sqsubseteq_L \zeta(x)$.

However, examining the results of the second non-uniform abstract interpretation (faulty client specification) with α_4, the analysis captures an instance of a process leakage as defined by Property 1. This secrecy breach occurs with the value of κ, which captures the secret password, *pwd*. Since $\zeta(pwd) = l$ and $\zeta(\kappa) = \perp_L$ according to the above name classification, we have that $\zeta(\kappa) \sqsubseteq \zeta(pwd)$. It is clear that this breach happened as a result of the client sending its password over the *covert* channel, which is a free name recognised by the intruder.

Authenticity. We adopt the following classification for trust levels:

$$
\begin{aligned}
ftp \stackrel{\text{def}}{=}\ & \lfloor (!start(x).(\nu\, login)(if\ x = start\ then\ \lfloor (\nu\, pwd)(Server \mid Client)\rfloor^{a} \\
& \qquad\qquad\qquad\qquad else\ \lfloor (\nu\, pwd)(Server)\rfloor^{a} \mid \lfloor I\rfloor^{\perp_A} \\
& \qquad)
\end{aligned}
$$

$$) \mid \overline{start}\langle start\rangle.\overline{start}\langle start\rangle.\overline{start}\langle start\rangle.\overline{start}\langle I\_start\rangle\rfloor^{\perp_A}$$

$$Server \stackrel{\texttt{def}}{=} (\nu\, deal)\ \lfloor login(z).\lfloor if\ z = pwd\ then$$
$$login(data).($$
$$\overline{deal}\langle data\rangle \mid\ !deal(w).\overline{login}\langle w\rangle.login(u).if\ u = logout\ then\ \mathbf{0}$$
$$else\ \overline{deal}\langle u\rangle\ )\ \ else\ \mathbf{0}\rfloor^{a}\rfloor^{\perp_A}$$

$$Client \stackrel{\texttt{def}}{=} (\nu\, request)\ (\overline{login}\langle pwd\rangle.\overline{login}\langle request\rangle.login(res).\overline{login}\langle logout\rangle)$$

With $\perp_A \sqsubseteq a$. Applying the encoding function $\mathcal{U}$ to each of the above (sub)processes, it is possible to construct a θ environment mapping names to their trust levels, as follows:

$$\theta(\kappa) = \theta(x) = \theta(z) = \theta(net) = \theta(login) = \theta(logout) = \theta(start) = \theta(I\_start) = \perp_A$$
$$\theta(pwd) = \theta(deal) = \theta(request) = a$$

In both analyses (with correct and faulty client specifications) with α_4, we find that input parameters captured the appropriate names. In general, $\forall x \in dom(\phi_{\mathcal{A}}), y \in \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, x)$: $\theta(x) \sqsubseteq_A \theta(y)$. Hence, no instance of the process authenticity breach occurs (Property 3). The case of the faulty client is interesting, since the intruder, I, failed in passing its low-level data to the clients, even though it clearly breaches the secrecy of their passwords as indicated above. This is due to the fact that the passwords are per login session, i.e. each time a copy of the system is spawned the passwords and the login channel are renamed. Hence, the intruder cannot use a password it obtains from the client in its own session with the FTP server. Attempting to do that means that it will fail as a result of the conditional process that the server runs and that maintains a correct distribution of password-login session names. Hence, the authenticity property is preserved despite a failure in the secrecy property of the protocol.

5.4.2 The Needham-Schroeder Public-Key Protocol Example

As we mentioned in Section 4.3.4, the Needham-Schroeder public-key protocol is susceptible to a form of impersonation attack, i.e. the man-in-the-middle attack, which was first discovered and fixed by Lowe [87]. From the results of the abstract interpretation given in Figure 4.7 (Section 4.3.4), we find that this attack is captured in the final fixed point value of $\phi_{\mathcal{A}}$. The fact that the intruder obtained nonces N'_{AB1}, N'_{BA1} and messages M'_{AB1}, M'_{BA1} demonstrates the success of this attack. The intruder was capable of accessing and

tampering with nonces and messages from A to B and vice versa. The intruder also succeeds in passing $untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, \kappa))$ to both B and A through variables $msg2_{AB1}$ and $msg2_{BA1}$.

Authenticity. To discuss the authenticity property of the Needham-Schroeder protocol, we apply the following trust-level classifications to the protocol of Figure 4.6 (Section 4.3.4):

$$\lfloor(\nu\, K_A^-)\ \ !(\lfloor Init\langle A, B\rangle\rfloor^a \ \mid\ \lfloor Init\langle A, I\rangle\rfloor^{\perp_A} \ \mid\ \lfloor Resp\langle B, A\rangle\rfloor^a \ \mid\ \lfloor Resp\langle I, A\rangle\rfloor^{\perp_A})\rfloor^a$$
$$\lfloor(\nu\, K_B^-)\ \ !(\lfloor Init\langle B, A\rangle\rfloor^a \ \mid\ \lfloor Init\langle B, I\rangle\rfloor^{\perp_A} \ \mid\ \lfloor Resp\langle A, B\rangle\rfloor^a \ \mid\ \lfloor Resp\langle I, B\rangle\rfloor^{\perp_A})\rfloor^a$$
$$\lfloor(\nu\, K_I^-)(I)\rfloor^{\perp_A}$$

Where $\perp_A \sqsubseteq a$. From this classification, we obtain the following value for θ by the application of $\mathcal{U}(Protocol)$ θ_0:

$$\theta(msg1_{AB}) = \theta(msg1_{BA}) = \theta(msg2_{AB}) = \theta(msg2_{BA}) = a$$
$$\theta(msg1_{AI}) = \theta(msg1_{BI}) = \theta(msg2_{IA}) = \theta(msg2_{IB}) = \perp_A$$
$$\theta(\kappa) = \theta(net) = \perp_A$$
$$\theta(N_{AB}) = \theta(N_{BA}) = \theta(N'_{AB}) = \theta(N'_{BA}) = \theta(M_{AB}) = \theta(M_{BA}) = \theta(M'_{AB}) = \theta(M'_{BA}) =$$
$$\theta(v_{AB}) = \theta(v_{BA}) = \theta(r'_{AB}) = \theta(r'_{BA}) = \theta(h'_{AB}) = \theta(h'_{BA}) = \theta(r_{AB}) = \theta(r_{BA}) = a$$
$$\theta(N_{AI}) = \theta(N_{BI}) = \theta(N'_{IB}) = \theta(N'_{IA}) = \theta(M_{AI}) = \theta(M_{BI}) = \theta(M'_{IB}) = \theta(M'_{IA}) =$$
$$\theta(v_{AI}) = \theta(v_{BI}) = \theta(r'_{IB}) = \theta(r'_{IA}) = \theta(v'_{IA}) = \theta(h'_{IA}) = \theta(r_{AI}) = \theta(r_{BI}) = \perp_A$$

It is clear from these classifications and the result of the abstract interpretation of Figure 4.7 (Section 4.3.4) that the process authenticity property is not preserved. For example, we have that any of the intruder's fresh data can be captured by agent B acting as the responder to A. Hence, $net \in untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, msg2_{AB}))$ and since $\theta(net) \sqsubseteq \theta(msg2_{AB})$, we have an instance of Property 4 indicating a breach in process authenticity. The same applies to $msg2_{BA}$.

Secrecy. We first classify the processes in the specification of the Needham-Schroeder protocol according to their secrecy requirements, as follows:

$$\lceil(\nu\, K_A^-)\ \ !(\lceil Init\langle A, B\rangle\rceil^l \ \mid\ \lceil Init\langle A, I\rangle\rceil^{\perp_L} \ \mid\ \lceil Resp\langle B, A\rangle\rceil^l \ \mid\ \lceil Resp\langle I, A\rangle\rceil^{\perp_L})\rceil^l$$
$$\lceil(\nu\, K_B^-)\ \ !(\lceil Init\langle B, A\rangle\rceil^l \ \mid\ \lceil Init\langle B, I\rangle\rceil^{\perp_L} \ \mid\ \lceil Resp\langle A, B\rangle\rceil^l \ \mid\ \lceil Resp\langle I, B\rangle\rceil^{\perp_L})\rceil^l$$
$$\lceil(\nu\, K_I^-)(I)\rceil^{\perp_L}$$

Where $\perp_L \sqsubseteq l$. Next, we obtain the value for ζ by the application of $\mathcal{Z}(Protocol)$ ζ_0:

$\zeta(msg1_{AB}) = \zeta(msg1_{BA}) = \zeta(msg2_{AB}) = \zeta(msg2_{BA}) = l$
$\zeta(msg1_{AI}) = \zeta(msg1_{BI}) = \zeta(msg2_{IA}) = \zeta(msg2_{IB}) = \bot_L$
$\zeta(\kappa) = \zeta(net) = \bot_L$
$\zeta(N_{AB}) = \zeta(N_{BA}) = \zeta(N'_{AB}) = \zeta(N'_{BA}) = \zeta(M_{AB}) = \zeta(M_{BA}) = \zeta(M'_{AB}) = \zeta(M'_{BA}) = \zeta(v_{AB}) = \zeta(v_{BA}) = \zeta(r'_{AB}) = \zeta(r'_{BA}) = \zeta(h'_{AB}) = \zeta(h'_{BA}) = \zeta(r_{AB}) = \zeta(r_{BA}) = l$
$\zeta(N_{AI}) = \zeta(N_{BI}) = \zeta(N'_{IB}) = \zeta(N'_{IA}) = \zeta(M_{AI}) = \zeta(M_{BI}) = \zeta(M'_{IB}) = \zeta(M'_{IA}) = \zeta(v_{AI}) = \zeta(v_{BI}) = \zeta(r'_{IB}) = \zeta(r'_{IA}) = \zeta(r_{BI}) = \zeta(r_{AI}) = \zeta(h'_{IB}) = \zeta(h'_{IA}) = \bot_L$

Considering the results of the abstract interpretation of Figure 4.7 (Section 4.3.4), it is clear that instances of the process leakage property occur, as the intruder is capable of capturing highly classified data. For example, $M'_{AB1} \in untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, \kappa))$, which is a message created by B in response to A. Given that $\zeta(\kappa) \sqsubseteq \zeta(M'_{AB1})$, we have an instance of Property 2.

5.4.3 The SPLICE/AS Protocol Example

Examining the results of the abstract interpretation of Figure 4.9 (Section 4.3.5), we find that the attack mentioned by [42] has caused anomalous values to appear in the intruder's knowledge, κ. These values are messages and nonces created by initiators in sessions involving A and B only. Also, the intruder succeeds in passing its knowledge as values for messages received by the initiator in sessions with the honest responder.

Authenticity. To discuss the authenticity property of the SPLICE/AS protocol as specified in Figure 4.8 (Section 4.3.5), we use the following trust-level classifications:

$\lfloor (\nu\, K_A^-)\ \ !(\lfloor Init\langle A, B\rangle \rfloor^a \ \mid\ \lfloor Init\langle A, I\rangle \rfloor^{\bot_A} \ \mid\ \lfloor Resp\langle B, A\rangle \rfloor^a \ \mid\ \lfloor Resp\langle I, A\rangle \rfloor^{\bot_A}) \rfloor^a$
$\lfloor (\nu\, K_B^-)\ \ !(\lfloor Init\langle B, A\rangle \rfloor^a \ \mid\ \lfloor Init\langle B, I\rangle \rfloor^{\bot_A} \ \mid\ \lfloor Resp\langle A, B\rangle \rfloor^a \ \mid\ \lfloor Resp\langle I, B\rangle \rfloor^{\bot_A}) \rfloor^a$
$\lfloor (\nu\, K_I^-)(I) \rfloor^{\bot_A}$

Where $\bot_A \sqsubseteq a$. From this classification, we obtain the following value for θ by the application of $\mathcal{U}(Protocol)$ θ_0:

$\theta(msg1_{AB}) = \theta(msg1_{BA}) = \theta(msg2_{AB}) = \theta(msg2_{BA}) = a$
$\theta(msg1_{AI}) = \theta(msg1_{BI}) = \theta(msg2_{IA}) = \theta(msg2_{IB}) = \bot_A$
$\theta(\kappa) = \theta(net) = \bot_A$
$\theta(N_{AB}) = \theta(N_{BA}) = \theta(M_{AB}) = \theta(M_{BA}) = \theta(M'_{AB}) = \theta(M'_{BA}) = \theta(v_{AB}) = \theta(v_{BA}) = \theta(v'_{AB}) = \theta(v'_{BA}) = a$
$\theta(N_{AI}) = \theta(N_{BI}) = \theta(M_{AI}) = \theta(M_{BI}) = \theta(M'_{IB}) = \theta(M'_{IA}) = \theta(v_{AI}) = \theta(v_{BI}) = \theta(v'_{IB}) = \theta(v'_{IA}) = \bot_A$

Examining the results of the abstract interpretation given in Figure 4.9 (Section 4.3.5) in light of the above value for θ, we find that authenticity is breached. For example, consider the value of the $msg1_{AB1}$ variable, which belongs to agent A acting as the initiator of the protocol to agent B. Here, we find that the result $net_1 \in untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, msg1_{AB1}))$ is possible, and due to the classification $\theta(net_1) \sqsubseteq \theta(msg1_{AB1})$, we have an instance of Property 4, i.e. process authenticity breach. A similar breach also occurs with $msg1_{BA1}$.

Secrecy. We adopt the following classification of secrecy levels for SPLICE/AS protocol:

$$\lceil (\nu\, K_A^-)\ \ !(\lceil Init\langle A, B\rangle \rceil^l \ \mid\ \lceil Init\langle A, I\rangle \rceil^{\perp_L} \ \mid\ \lceil Resp\langle B, A\rangle \rceil^l \ \mid\ \lceil Resp\langle I, A\rangle \rceil^{\perp_L})\rceil^l$$
$$\lceil (\nu\, K_B^-)\ \ !(\lceil Init\langle B, A\rangle \rceil^l \ \mid\ \lceil Init\langle B, I\rangle \rceil^{\perp_L} \ \mid\ \lceil Resp\langle A, B\rangle \rceil^l \ \mid\ \lceil Resp\langle I, B\rangle \rceil^{\perp_L})\rceil^l$$
$$\lceil (\nu\, K_I^-)(I)\rceil^{\perp_L}$$

Where $\perp_L \sqsubseteq l$. One can now obtain the following value for ζ by applying $\mathcal{Z}(Protocol)\ \zeta_0$:

$$\zeta(msg1_{AB}) = \zeta(msg1_{BA}) = \zeta(msg2_{AB}) = \zeta(msg2_{BA}) = l$$
$$\zeta(msg1_{AI}) = \zeta(msg1_{BI}) = \zeta(msg2_{IA}) = \zeta(msg2_{IB}) = \perp_L$$
$$\zeta(\kappa) = \zeta(net) = \perp_L$$
$$\zeta(N_{AB}) = \zeta(N_{BA}) = \zeta(M_{AB}) = \zeta(M_{BA}) = \zeta(v_{AB}) = \zeta(v_{BA}) = \zeta(v'_{AB}) = \zeta(v'_{BA}) = l$$
$$\zeta(N_{AI}) = \zeta(N_{BI}) = \zeta(M_{AI}) = \zeta(M_{BI}) = \zeta(v_{AI}) = \zeta(v_{BI}) = \zeta(v'_{IB}) = \zeta(v'_{IA}) = \perp_L$$

The results of the abstract interpretation of Figure 4.9 (Section 4.3.5) demonstrate breaches in the process leakage property, as the intruder is capable of capturing data created by agents A and B in sessions involving these two agents. For example, $M_{AB1} \in untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, \kappa))$, which is a message created by A initiating a session with B. Given that $\zeta(\kappa) \sqsubseteq \zeta(M_{AB1})$, we have an instance of Property 2.

5.4.4 The Otway-Rees Protocol Example

As we mentioned in Section 4.3.6, the version of the Otway-Rees protocol specified in Figure 4.10 is vulnerable to two forms of impersonation attacks, published in [106] and [28]. Furthermore, we find that these attacks caused anomalous results in Figure 4.11 (Section 4.3.6). The intruder was successful in obtaining messages M_{AB1}, M_{BA1}, M'_{AB1} and M'_{BA1} from sessions involving A and B only. Also the presence of keys K_{IA1} and K_{IB1} shared between the intruder and agents A and B in the values for the variables f'_{AB1}, f'_{BA1}, v_{AB1} and v_{BA1} indicates the presence of extra communication sessions. Finally, we find that any

of the names in the intruder's knowledge, κ, can be passed as messages to agents A and B in sessions involving these two agents alone.

Authenticity. The authenticity property of the Otway-Rees protocol is discussed under the following trust address classifications:

$$\lfloor(\nu\, K_{AS})\ \ !(\lfloor \mathit{Init}\langle A,B\rangle\rfloor^{a}\ \mid\ \lfloor \mathit{Init}\langle A,I\rangle\rfloor^{\perp_A}\ \mid\ \lfloor \mathit{Resp}\langle B,A\rangle\rfloor^{a}\ \mid\ \lfloor \mathit{Resp}\langle I,A\rangle\rfloor^{\perp_A})\rfloor^{a}$$
$$\lfloor(\nu\, K_{BS})\ \ !(\lfloor \mathit{Init}\langle B,A\rangle\rfloor^{a}\ \mid\ \lfloor \mathit{Init}\langle B,I\rangle\rfloor^{\perp_A}\ \mid\ \lfloor \mathit{Resp}\langle A,B\rangle\rfloor^{a}\ \mid\ \lfloor \mathit{Resp}\langle I,B\rangle\rfloor^{\perp_A})\rfloor^{a}$$
$$\lfloor(\nu\, K_{IS})(I)\rfloor^{\perp_A}$$

Where $\perp_A \sqsubseteq a$, and we drop the server S since we are not interested in its security properties. From this classification, we obtain the following value for θ by the application of $\mathcal{U}$ to the above processes:

$$\theta(msg1_{AB}) = \theta(msg1_{BA}) = \theta(msg2_{AB}) = \theta(msg2_{BA}) = a$$
$$\theta(msg1_{AI}) = \theta(msg1_{BI}) = \theta(msg2_{IA}) = \theta(msg2_{IB}) = \perp_A$$
$$\theta(\kappa) = \theta(net) = \perp_A$$
$$\theta(M_{AB}) = \theta(M_{BA}) = \theta(M'_{AB}) = \theta(M'_{BA}) = \theta(v_{AB}) = \theta(v_{BA}) = \theta(f'_{AB}) = \theta(f'_{BA}) = a$$
$$\theta(M_{AI}) = \theta(M_{BI}) = \theta(M'_{IB}) = \theta(M'_{IA}) = \theta(v_{AI}) = \theta(v_{BI}) = \theta(f'_{IB}) = \theta(f'_{IA}) = \perp_A$$

We find breaches of the process authenticity property. Taking the case of initiator A as an example, we have from the first attack that, $net_1 \in \mathit{untag}(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, msg2_{AB1}))$, is a possible instantiation for the $msg2_{AB1}$ variable owned by A. Hence, we have that $\theta(net_1) \sqsubseteq \theta(msg2_{AB1})$. The authenticity breach occurs as a result of the address classification above, which states that $\theta(net_1) = \perp_A$ and $\theta(msg2_{AB1}) = a$. In the case of the second attack, and taking the example case of B as the responder, we have that, $net_1 \in \mathit{untag}(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, msg1_{AB1}))$, is a possible instantiation. The process authenticity property is breached again as $\theta(net_1) \sqsubseteq \theta(msg1_{AB1})$, where $\theta(net_1) = \perp_A$ and $\theta(msg1_{AB1}) = a$.

Secrecy. Consider the following secrecy classification for the Otway-Rees protocol:

$$\lceil(\nu\, K_{AS})\ \ !(\lceil \mathit{Init}\langle A,B\rangle\rceil^{l}\ \mid\ \lceil \mathit{Init}\langle A,I\rangle\rceil^{\perp_L}\ \mid\ \lceil \mathit{Resp}\langle B,A\rangle\rceil^{l}\ \mid\ \lceil \mathit{Resp}\langle I,A\rangle\rceil^{\perp_L})\rceil^{l}$$
$$\lceil(\nu\, K_{BS})\ \ !(\lceil \mathit{Init}\langle B,A\rangle\rceil^{l}\ \mid\ \lceil \mathit{Init}\langle B,I\rangle\rceil^{\perp_L}\ \mid\ \lceil \mathit{Resp}\langle A,B\rangle\rceil^{l}\ \mid\ \lceil \mathit{Resp}\langle I,B\rangle\rceil^{\perp_L})\rceil^{l}$$
$$\lceil(\nu\, K_{IS})(I)\rceil^{\perp_A}$$

Where $\perp_L \sqsubseteq l$, and we are not interested in the security level of the server S. It is now

possible to construct ζ by applying $\mathcal{Z}$ to the above classified processes. This will result in the following value for ζ:

$\zeta(msg1_{AB}) = \zeta(msg1_{BA}) = \zeta(msg2_{AB}) = \zeta(msg2_{BA}) = l$
$\zeta(msg1_{AI}) = \zeta(msg1_{BI}) = \zeta(msg2_{IA}) = \zeta(msg2_{IB}) = \bot_L$
$\zeta(\kappa) = \zeta(net) = \bot_L$
$\zeta(M_{AB}) = \zeta(M_{BA}) = \zeta(M'_{AB}) = \zeta(M'_{BA}) = \zeta(v_{AB}) = \zeta(v_{BA}) = \zeta(f'_{AB}) = \zeta(f'_{BA}) = l$
$\zeta(M_{AI}) = \zeta(M_{BI}) = \zeta(M'_{IB}) = \zeta(M'_{IA}) = \zeta(v_{AI}) = \zeta(v_{BI}) = \zeta(f'_{IB}) = \zeta(f'_{IA}) = \bot_L$

The attacks mentioned in Section 4.3.6 introduce breaches in the secrecy of messages created by agents A and B in the Otway-Rees protocol. Again we refer to the results of the abstract interpretation of Figure 4.11 (Section 4.3.6) where we find that for the case of the first attack, messages created by A and B as initiators end up in I's knowledge, as $\{M_{AB1}, M_{BA1}\} \subseteq untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, \kappa))$. From the second attack, we have that messages created by A and B as responders are captured by I, as $\{M'_{AB1}, M'_{BA1}\} \subseteq \phi_{\mathcal{A}}(\kappa)$. Taking messages M_{AB1} and M'_{AB1} as examples, we have that $\zeta(\kappa) \sqsubseteq \zeta(M_{AB1})$ and $\zeta(\kappa) \sqsubseteq \zeta(M'_{AB1})$ since from the above classification, we have $\zeta(\kappa) = \bot_L$ and $\zeta(M_{AB1}) = \zeta(M'_{AB1}) = a$. Hence, instances of process leakage property as stated in Property 2 occur in the above results.

5.4.5 The Kerberos Protocol Example

The results of the abstract interpretation of the version of the Kerberos protocol given in Section 4.3.7 reveal correct distributions of keys and nonces created by the server S for the different sessions. For example, we find that all the keys K_{IX} and K_{XI} for $X \in \{A, B\}$ are assigned to input parameters u and w' only belonging to sessions where A or B communicate with I. The intruder naturally obtains these keys in its knowledge. Similarly, nonces N_{IX} and N_{XI} are only bound to variables t, t' and κ in sessions involving I.

Authenticity. To analyse the authenticity property of the Kerberos protocol, we assume the following classification of trust levels for the specification of the Kerberos protocol as given in Figure 4.12 (Section 4.3.7):

$\lfloor(\nu\, K_{AS})\ \ !(\lfloor Init\langle A, B\rangle\rfloor^a \ \mid\ \lfloor Init\langle A, I\rangle\rfloor^{\bot_A} \ \mid\ \lfloor Resp\langle B, A\rangle\rfloor^a \ \mid\ \lfloor Resp\langle I, A\rangle\rfloor^{\bot_A})\rfloor^a$
$\lfloor(\nu\, K_{BS})\ \ !(\lfloor Init\langle B, A\rangle\rfloor^a \ \mid\ \lfloor Init\langle B, I\rangle\rfloor^{\bot_A} \ \mid\ \lfloor Resp\langle A, B\rangle\rfloor^a \ \mid\ \lfloor Resp\langle I, B\rangle\rfloor^{\bot_A})\rfloor^a$
$\lfloor(\nu\, K_{IS})(I)\rfloor^{\bot_A}$

Where $\bot_A \sqsubseteq a$, and we drop the server S since we are not interested in its security properties. From this classification, we obtain the following value for θ by the application of $\mathcal{U}$ to the above processes:

$\theta(msg1_{AB}) = \theta(msg1_{BA}) = \theta(msg2_{AB}) = \theta(msg2_{BA}) = a$
$\theta(msg1_{AI}) = \theta(msg1_{BI}) = \theta(msg2_{IA}) = \theta(msg2_{IB}) = \bot_A$
$\theta(\kappa) = \theta(net) = \bot_A$
$\theta(N_{AB}) = \theta(N_{BA}) = \theta(M_{AB}) = \theta(M_{BA}) = \theta(M'_{AB}) = \theta(M'_{BA}) = \theta(u_{AB}) = \theta(u_{BA}) = \theta(w'_{AB}) = \theta(w'_{BA}) = \theta(t_{AB}) = \theta(t_{BA}) = \theta(t'_{AB}) = \theta(t'_{BA}) = a$
$\theta(N_{AI}) = \theta(N_{BI}) = \theta(M_{AI}) = \theta(M_{BI}) = \theta(M'_{IB}) = \theta(M'_{IA}) = \theta(u_{AI}) = \theta(u_{BI}) = \theta(w'_{IB}) = \theta(w'_{IA}) = \theta(t_{AI}) = \theta(t_{BI}) = \theta(t'_{IB}) = \theta(t'_{IA}) = \bot_A$

From the results of the abstract interpretation of Figure 4.13 (Section 4.3.7) and taking as an example input variables $msg2_{AB}$ and $msg1_{AB}$, we have, $M_{AB1} \in untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, msg1_{AB1}))$ and $M'_{AB1} \in untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, msg2_{AB1}))$ are possible values. Referring to the above classification of variables and messages, we have that $\theta(M_{AB1}) = \theta(msg1_{AB1})$ and $\theta(M'_{AB1}) = \theta(msg2_{AB1})$ since $\theta(M_{AB1}) = \theta(M'_{AB1}) = a$ and $\theta(msg1_{AB1}) = \theta(msg2_{AB1}) = a$. The same statement is true about the rest of the variables in the domain of $\phi_{\mathcal{A}}$ (except for u and w', which capture the keys created by the server). These results demonstrate that no breaches of the process authenticity property as defined in Property 4 have occurred.

Secrecy. We adopt the following secrecy classification for processes in the specification of the Kerberos protocol 4.12 (Section 4.3.7):

$\lceil(\nu\, K_{AS})\ \ !(\lceil Init\langle A, B\rangle\rceil^{l}\ \mid\ \lceil Init\langle A, I\rangle\rceil^{\bot_L}\ \mid\ \lceil Resp\langle B, A\rangle\rceil^{l}\ \mid\ \lceil Resp\langle I, A\rangle\rceil^{\bot_L})\rceil^{l}$
$\lceil(\nu\, K_{BS})\ \ !(\lceil Init\langle B, A\rangle\rceil^{l}\ \mid\ \lceil Init\langle B, I\rangle\rceil^{\bot_L}\ \mid\ \lceil Resp\langle A, B\rangle\rceil^{l}\ \mid\ \lceil Resp\langle I, B\rangle\rceil^{\bot_L})\rceil^{l}$
$\lceil(\nu\, K_{IS})(I)\rceil^{\bot_A}$

Where $\bot_L \sqsubseteq l$, and we are not interested in the security level of the server S. To evaluate ζ, we apply $\mathcal{Z}$ to the above classified processes. This will result in the following value:

$\zeta(msg1_{AB}) = \zeta(msg1_{BA}) = \zeta(msg2_{AB}) = \zeta(msg2_{BA}) = l$
$\zeta(msg1_{AI}) = \zeta(msg1_{BI}) = \zeta(msg2_{IA}) = \zeta(msg2_{IB}) = \bot_L$
$\zeta(\kappa) = \zeta(net) = \bot_L$
$\zeta(N_{AB}) = \zeta(N_{BA}) = \zeta(M_{AB}) = \zeta(M_{BA}) = \zeta(M'_{AB}) = \zeta(M'_{BA}) = \zeta(u_{AB}) = \zeta(u_{BA}) =$

$\zeta(w'_{AB}) = \zeta(w'_{BA}) = \zeta(t_{AB}) = \zeta(t_{BA}) = \zeta(t'_{AB}) = \zeta(t'_{BA}) = l$
$\zeta(N_{AI}) = \zeta(N_{BI}) = \zeta(M_{AI}) = \zeta(M_{BI}) = \zeta(M'_{IB}) = \zeta(M'_{IA}) = \zeta(u_{AI}) = \zeta(u_{BI}) = \zeta(w'_{IB}) = \zeta(w'_{IA}) = \zeta(t_{AI}) = \zeta(t_{BI}) = \zeta(t'_{IB}) = \zeta(t'_{IA}) = \bot_L$

The secrecy property of messages exchanged between A and B in the presence of I is also preserved as a result of the inability of the intruder I to undermine any of the session keys created for communications between A and B. More accurately, we find from the abstract interpretation of the protocol (Figure 4.13, Section 4.3.7) that $K_{AB1}, K_{BA1} \notin \mathit{untag}(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, \kappa))$ for all the instances of the session keys created for communications A and B. As a result, the interpretation reveals that $M_{AB1}, M_{BA1}, M'_{AB1}, M'_{BA1} \notin \mathit{untag}(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, \kappa))$. For the remaining keys and messages captured by $\mathit{untag}(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, \kappa))$, we have that $\zeta_l(M_{AI1}) = \zeta_l(M_{BI1}) = \zeta_l(M'_{IA1}) = \zeta_l(M'_{IB1}) = \bot_L$. Therefore, the secrecy of messages created in sessions between A and B is preserved according to the process leakage definition in Property 2.

5.4.6 The Yahalom Protocol Example

The uniform abstract interpretation of the Yahalom protocol in Section 4.3.8 demonstrated correct distributions of session keys and messages among the participants in the protocol including the intruder I. This is despite the possibility of some replay attacks that were reported by Paulson in [105] and that are based on compromised old session keys. Since we do not emphasize the behaviour of agents A and B after they exchange the follow-up messages (i.e. the continuation processes $F(msg2)$ and $F'(msg1)$), the results of our abstract interpretation are not as sensitive as the results of [105].

Authenticity. We adopt the following trust-level classifications for processes in the Yahalom protocol specification given in Figure 4.14 (Section 4.3.8), except for the server:

$\lfloor(\nu\, K_{AS})\ \ !(\lfloor \mathit{Init}\langle A, B\rangle \rfloor^{a}\ \mid\ \lfloor \mathit{Init}\langle A, I\rangle \rfloor^{\bot_A}\ \mid\ \lfloor \mathit{Resp}\langle B, A\rangle \rfloor^{a}\ \mid\ \lfloor \mathit{Resp}\langle I, A\rangle \rfloor^{\bot_A})\rfloor^{a}$
$\lfloor(\nu\, K_{BS})\ \ !(\lfloor \mathit{Init}\langle B, A\rangle \rfloor^{a}\ \mid\ \lfloor \mathit{Init}\langle B, I\rangle \rfloor^{\bot_A}\ \mid\ \lfloor \mathit{Resp}\langle A, B\rangle \rfloor^{a}\ \mid\ \lfloor \mathit{Resp}\langle I, B\rangle \rfloor^{\bot_A})\rfloor^{a}$
$\lfloor(\nu\, K_{IS})(I)\rfloor^{\bot_A}$

Where $\bot_A \sqsubseteq a$. The following θ is obtained by the application of $\mathcal{U}$ to the above processes:

$\theta(msg1_{AB}) = \theta(msg1_{BA}) = \theta(msg2_{AB}) = \theta(msg2_{BA}) = a$
$\theta(msg1_{AI}) = \theta(msg1_{BI}) = \theta(msg2_{IA}) = \theta(msg2_{IB}) = \bot_A$
$\theta(\kappa) = \theta(net) = \bot_A$

$\theta(N_{AB}) = \theta(N_{BA}) = \theta(N'_{AB}) = \theta(N'_{BA}) = \theta(M_{AB}) = \theta(M_{BA}) = \theta(M'_{AB}) = \theta(M'_{BA}) =$
$\theta(v_{AB}) = \theta(v_{BA}) = \theta(r'_{AB}) = \theta(r'_{BA}) = \theta(w_{AB}) = \theta(w_{BA}) = \theta(f'_{AB}) = \theta(f'_{BA}) = a$
$\theta(N_{AI}) = \theta(N_{BI}) = \theta(N'_{IB}) = \theta(N'_{IA}) = \theta(M_{AI}) = \theta(M_{BI}) = \theta(M'_{IB}) = \theta(M'_{IA}) =$
$\theta(v_{AI}) = \theta(v_{BI}) = \theta(r'_{IB}) = \theta(r'_{IA}) = \theta(w_{AI}) = \theta(w_{BI}) = \theta(f'_{IB}) = \theta(f'_{IA}) = \bot_A$

From the results of the abstract interpretation of Figure 4.15 (Section 4.3.8) and the above classifications, no breaches of the process authenticity property (Property 4) are detected.

Secrecy. We adopt the following secrecy classification for the Yahalom protocol:

$\lceil(\nu\, K_{AS})\ \ !(\lceil Init\langle A, B\rangle\rceil^{l}\ \mid\ \lceil Init\langle A, I\rangle\rceil^{\bot_L}\ \mid\ \lceil Resp\langle B, A\rangle\rceil^{l}\ \mid\ \lceil Resp\langle I, A\rangle\rceil^{\bot_L})\rceil^{l}$
$\lceil(\nu\, K_{BS})\ \ !(\lceil Init\langle B, A\rangle\rceil^{l}\ \mid\ \lceil Init\langle B, I\rangle\rceil^{\bot_L}\ \mid\ \lceil !Resp\langle A, B\rangle\rceil^{l}\ \mid\ \lceil Resp\langle I, B\rangle\rceil^{\bot_L})\rceil^{l}$
$\lceil(\nu\, K_{IS})(I)\rceil^{\bot_L}$

Where $\bot_L \sqsubseteq l$. We now construct ζ by applying $\mathcal{Z}$ to the above processes:

$\zeta(msg1_{AB}) = \zeta(msg1_{BA}) = \zeta(msg2_{AB}) = \zeta(msg2_{BA}) = l$
$\zeta(msg1_{AI}) = \zeta(msg1_{BI}) = \zeta(msg2_{IA}) = \zeta(msg2_{IB}) = \bot_L$
$\zeta(\kappa) = \zeta(net) = \bot_L$
$\zeta(N_{AB}) = \zeta(N_{BA}) = \zeta(N'_{AB}) = \zeta(N'_{BA}) = \zeta(M_{AB}) = \zeta(M_{BA}) = \zeta(M'_{AB}) = \zeta(M'_{BA}) =$
$\zeta(v_{AB}) = \zeta(v_{BA}) = \zeta(r'_{AB}) = \zeta(r'_{BA}) = \zeta(w_{AB}) = \zeta(w_{BA}) = \zeta(f'_{AB}) = \zeta(f'_{BA}) = l$
$\zeta(N_{AI}) = \zeta(N_{BI}) = \zeta(N'_{IB}) = \zeta(N'_{IA}) = \zeta(M_{AI}) = \zeta(M_{BI}) = \zeta(M'_{IB}) = \zeta(M'_{IA}) =$
$\zeta(v_{AI}) = \zeta(v_{BI}) = \zeta(r'_{IB}) = \zeta(r'_{IA}) = \zeta(w_{AI}) = \zeta(w_{BI}) = \zeta(f'_{IB}) = \zeta(f'_{IA}) = \bot_L$

Again similar to the authenticity property, we find that the results of the abstract interpretation of Figure 4.15 (Section 4.3.8) have revealed no instances of the process leakage property (Property 2) in the light of the above secrecy-level classifications.

5.4.7 The Woo-Lam Protocol Example

We review here the authenticity and secrecy properties of the Woo-Lam protocol, as specified by the version of Figure 4.16 in Section 4.3.9. The results of the abstract interpretation of Figure 4.17 (Section 4.3.9) revealed some anomalous values for the κ, s'_{AB} and s'_{BA} variables. These values are caused by the attack discovered by [7] and reviewed in Section 4.3.9.

Authenticity. Consider the following classifications of trust levels for the specification of the Woo-Lam protocol:

$$\lfloor(\nu\, K_{AS})\ \ !(\lfloor \mathit{Init}\langle A, B\rangle\rfloor^{a}\ \mid\ \lfloor \mathit{Init}\langle A, I\rangle\rfloor^{\perp_A}\ \mid\ \lfloor !\mathit{Resp}\langle B, A\rangle\rfloor^{a}\ \mid\ \lfloor \mathit{Resp}\langle I, A\rangle\rfloor^{\perp_A})\rfloor^{a}$$
$$\lfloor(\nu\, K_{BS})\ \ !(\lfloor \mathit{Init}\langle B, A\rangle\rfloor^{a}\ \mid\ \lfloor \mathit{Init}\langle B, I\rangle\rfloor^{\perp_A}\ \mid\ \lfloor !\mathit{Resp}\langle A, B\rangle\rfloor^{a}\ \mid\ \lfloor \mathit{Resp}\langle I, B\rangle\rfloor^{\perp_A})\rfloor^{a}$$
$$\lfloor(\nu\, K_{IS})(I)\rfloor^{\perp_A}$$

Where $\perp_A \sqsubseteq a$, and we drop the server S since we are not interested in its security properties. From this classification, we obtain the following value for θ by the application of $\mathcal{U}$ to the above processes:

$$\theta(msg1_{AB}) = \theta(msg1_{BA}) = \theta(msg2_{AB}) = \theta(msg2_{BA}) = a$$
$$\theta(msg1_{AI}) = \theta(msg1_{BI}) = \theta(msg2_{IA}) = \theta(msg2_{IB}) = \perp_A$$
$$\theta(\kappa) = \theta(net) = \perp_A$$
$$\theta(N'_{AB}) = \theta(N'_{BA}) = \theta(M'_{AB}) = \theta(M'_{BA}) = \theta(s'_{AB}) = \theta(s'_{BA}) = \theta(x_{AB}) = \theta(x_{BA}) = \theta(r'_{AB}) = \theta(r'_{BA}) = a$$
$$\theta(N'_{IA}) = \theta(N'_{IB}) = \theta(M'_{IB}) = \theta(M'_{IA}) = \theta(s'_{IB}) = \theta(s'_{IA}) = \theta(x_{AI}) = \theta(x_{BI}) = \theta(r'_{IB}) = \theta(r'_{IA}) = \perp_A$$

Analysing the results of the abstract interpretation along with the above classifications reveals the success of the attack of [7] in undermining the process authenticity property as defined by Property 4. This is clear from the values of the s'_{AB} and s'_{BA} variables. For example, we have that $net_1 \in \mathit{untag}(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, s'_{AB1}))$ and $net_1 \in \mathit{untag}(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, s'_{BA1}))$ are possible instantiations of these variables. From the above value for θ, we have that $\theta(net_1) \sqsubseteq \theta(s'_{AB1})$ and $\theta(net_1) \sqsubseteq \theta(s'_{AB1})$, since $\theta(net_1) = \perp_A$ and $\theta(s'_{AB1}) = \theta(s'_{AB1}) = a$.

Secrecy. For analysing the secrecy property for the Woo-Lam protocol, we adopt the following secrecy level classifications for the involved processes, except S:

$$\lceil(\nu\, K_{AS})\ \ !(\lceil \mathit{Init}\langle A, B\rangle\rceil^{l}\ \mid\ \lceil \mathit{Init}\langle A, I\rangle\rceil^{\perp_L}\ \mid\ \lceil \mathit{Resp}\langle B, A\rangle\rceil^{l}\ \mid\ \lceil \mathit{Resp}\langle I, A\rangle\rceil^{\perp_L})\rceil^{l}$$
$$\lceil(\nu\, K_{BS})\ \ !(\lceil \mathit{Init}\langle B, A\rangle\rceil^{l}\ \mid\ \lceil \mathit{Init}\langle B, I\rangle\rceil^{\perp_L}\ \mid\ \lceil \mathit{Resp}\langle A, B\rangle\rceil^{l}\ \mid\ \lceil \mathit{Resp}\langle I, B\rangle\rceil^{\perp_L})\rceil^{l}$$
$$\lceil(\nu\, K_{IS})(I)\rceil^{\perp_A}$$

Where $\perp_L \sqsubseteq l$. Computing ζ results in the following value:

$\zeta(msg1_{AB}) = \zeta(msg1_{BA}) = \zeta(msg2_{AB}) = \zeta(msg2_{BA}) = l$
$\zeta(msg1_{AI}) = \zeta(msg1_{BI}) = \zeta(msg2_{IA}) = \zeta(msg2_{IB}) = \bot_L$
$\zeta(\kappa) = \zeta(net) = \bot_L$
$\zeta(N'_{AB}) = \zeta(N'_{BA}) = \zeta(M'_{AB}) = \zeta(M'_{BA}) = \zeta(s'_{AB}) = \zeta(s'_{BA}) = \zeta(x_{AB}) = \zeta(x_{BA}) = \zeta(r'_{AB}) = \zeta(r'_{BA}) = l$
$\zeta(N'_{IA}) = \zeta(N'_{IB}) = \zeta(M'_{IB}) = \zeta(M'_{IA}) = \zeta(s'_{IB}) = \zeta(s'_{IA}) = \zeta(x_{AI}) = \zeta(x_{BI}) = \zeta(r'_{IB}) = \zeta(r'_{IA}) = \bot_L$

The secrecy property of the Woo-Lam protocol is also breached according to Property 2. This is clear from examining the final value of the intruder's knowledge, κ. We find that $M'_{AB1}, M'_{BA1} \in untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, \kappa))$, both of which were created by agents A and B in communications involving these two agents alone. Process leakage occurs since $\zeta(\kappa) \sqsubseteq \zeta(M'_{AB1})$ where $\zeta(\kappa) = \bot_L$ and $\zeta(M'_{AB1}) = l$. The same argument applies to M'_{BA1} captured by $untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, \kappa))$.

5.5 Conclusion

In this chapter, we have formally defined the notions of secrecy and authenticity of information in processes specified in the π-calculus and the spi calculus. The secrecy property is centred on the idea of a low-level process obtaining high-level data, whereas the authenticity property is based on the reverse idea of a high-level process capturing data created by low-level processes. These definitions were then combined with the results of the abstract interpretation of the system examples given earlier in Chapter 4, for detecting instances of secrecy and authenticity breaches in those systems. In the FTP example, we demonstrated that using per-session passwords (once-off passwords) protects authenticity, even though the passwords were leaked after their usage. For the case of authentication protocols, the security analysis detected the man-in-the-middle-attack in the Needham-Schroeder public-key protocol. It also detected well-known impersonation attacks in the SPLICE/AS, Otway-Rees and Woo-Lam protocols. No attacks were found in the Kerberos or the Yahalom protocols.

Chapter 6

Automatic Tools

6.1 Introduction

In this chapter, we review briefly two implementations of the abstract interpretation and the process leakage analysis as presented in Chapters 4 and 5, respectively. Two initial prototypes were developed: *Picasso*, a static analyser for processes in the π-calculus, and *Spicasso*, its cryptographic sibling for the language of the spi calculus. The programming language used is Objective Caml (OCaml) version 3.02, a functional programming environment.

6.2 Picasso: A Pi-Calculus Analyser for Secrecy and Security Objectives

The Picasso program is a direct implementation of the abstract semantics of the π-calculus discussed in Section 4.2.2 and its current version incorporates the secrecy property of process leakage (Property 1, Section 5.2.1). The program receives as its main input the specification of the process under analysis (along with other parameters related to context information and uniformity of the analysis), and returns information about any term (name) substitutions that may take place in that process. The process leakage function also permits the detection of any instances of the process leakage threat in the analysed process. These are captured as lists signifying the leaked names.

The design of the Picasso program is fully modular. The main modules are described in the following paragraphs, along with the main type and functions in each module.

Module `Process`**.** This module introduces a π-calculus process type as shown in Figure 6.1, and some functions on processes (e.g. for performing substitutions and labelling bound names and tags).

```
type name    =  string
and process  =  NULL
             |  INPUT of name * name * process
             |  OUTPUT of name * name * process
             |  MATCH of name * name * process
             |  MISMATCH of name * name * process
             |  REST of name * process
             |  SUM of process * process
             |  COM of process * process
             |  REP of process
```

Figure 6.1: The `name` and `process` data types in Picasso.

For example, the process:

$$x(y).y(z).\mathbf{0} \quad | \quad (\nu\, data)(\nu\, Info)\ \overline{x}\langle data^{t1}\rangle.\overline{data}\langle Info^{t2}\rangle.\mathbf{0}$$

instantiates the following type:

```
Process.COM(Process.INPUT("x","y",Process.INPUT("y","z",Process.NULL)),
Process.REST("Data",Process.REST("Info",Process.OUTPUT("x","t1",
Process.OUTPUT("Data","t2",Process.NULL)))))
```

While it keeps a mapping from tags to their name values. Hence, `Data` can be retrieved from `t1` and `Info` can be retrieved from `t2`.

Module `Phi`**.** The main type in this module is the type denoting $\phi_{\mathcal{A}}$ environments:

```
phi = (Process.name * Process.name list) list
```

This type represents a list of pairs. The first element of each pair belongs to the set of variables of the analysed process (input parameters in the case of the π-calculus) and the second element is a list of names denoting possible tags instantiating that variable. For example, the following value for $\phi_{\mathcal{A}}$:

$$\phi_{\mathcal{A}} = \begin{bmatrix} x_1 & \mapsto & \{t_1\} \\ x_2 & \mapsto & \{t_1, t_2\} \\ x_3 & \mapsto & \{t_1, t_2, t_3\} \end{bmatrix}$$

is represented as the type instantiation:

```
- :  (Process.name * Process.name list) list =
[("x1", ["t1"]); ("x2", ["t1";"t2"]);("x3", ["t1";"t2";"t3"])]
```

Module `Rho`**.** This module introduces the ρ multiset and other related functions over ρ. The ρ multiset is represented as a list of processes composed with the analysed process:

```
type rho = Process.process list
```

Where `Process.process` is the process data type defined in module `Process`.

Module `Picasso`**.** This is the main abstract interpretation module for the π-calculus. The abstract semantic relation $\mathcal{A}^{\pi}(\![P]\!)\ \rho\ \phi_{\mathcal{A}}$ is defined by the following function signature:

```
val picasso :  Process.process -> Process.process list -> int ->
Process.name list -> (Process.name * Process.name list) list
```

where the function takes as arguments a process type instantiation of the analysed process, P, a list of processes running in parallel with the analysed process, ρ_0, a number representing the abstraction constraint, n, and a list of names representing the initial knowledge of the intruder, κ_{init}. The outcome reflects a set of names, $\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, x)$, that can substitute an input parameter, x. The mapping also contains a variable, `intruder`, in its domain, which signifies the intruder's knowledge, κ.

For example, consider the following system,

$$system \stackrel{\texttt{def}}{=} \ !x(y) \ \mid \ !(\nu\ data)\overline{x}\langle data^t \rangle$$

Applying `Picasso.picasso system [ ] 1 ["x"]`, we obtain the following result:

```
- :  (Process.name * Process.name list) list =
[("intruder", ["DATA1"; "x"]); ("y1", ["DATA1"; "x"])]
```

Where the intruder process manages to capture `Data1`, by communicating over `x` with the replicated output process, $!(\nu\ data)\overline{x}\langle data^t\rangle$. On the other hand, variable `y1` captures both `Data1`, due to the communication with the replicated output, $!(\nu\ data)\overline{x}\langle data^t\rangle$, and `x`, due to the communication with the intruder, who can output any of the names in its knowledge over any other name in that knowledge.

Module `Security_Levels.` The module provides a data type for security policies that are used in classifying processes. This type is shown in Figure 6.2.

```
proc_level  =  L0 of int
            |  L1 of int * proc_level
            |  L2 of int * proc_level * proc_level
```

Figure 6.2: The `proc_level` data type in Picasso.

Security levels are modelled as binary trees. Leaves are the security levels of processes that have no subprocesses (e.g. null), nodes with single child being the levels of processes with a single subprocess (e.g. guards, restrictions and replications) and finally, nodes with two children being the levels of processes with two subprocesses (e.g. parallel compositions and summations).

For example, in the following system, we assume that P has security level 2 and Q, Q' have security level 1 ($\mathbf{0}$ is assumed to have no level, which is denoted as -1):

$P \stackrel{\text{def}}{=} Q \mid Q'$
$Q \stackrel{\text{def}}{=} x(y).\overline{y}\langle z\rangle.\mathbf{0}$
$Q' \stackrel{\text{def}}{=} \overline{x}\langle r\rangle.\mathbf{0}$

Then, the corresponding security policy will be written as:

```
Security_Levels.L2(2,
Security_Levels.L1(1,Security_Levels.L0(-1)),
Security_Levels.L1(1,Security_Levels.L0(-1))
```

Module `Pi.` This module represents an implementation of the process leakage analysis that is based on the results obtained from applying the `Picasso.picasso` function. The module consists of the following function:

```
val proc_leakage :  Process.process -> Process.process list ->
int -> Security_Levels.proc_level -> int -> Process.name list ->
Process.name list
```

The `proc_leakage` function takes the same arguments as the `Picasso.picasso` function in addition to a security policy and an integer representing the security level of the intruder. The function then returns a list of names that have been leaked to lower level processes.

As an example, applying `proc_leakage` to system (6.1), mentioned earlier, with $\rho_0 = x(e).\mathbf{0}$, $k = 1$, $\kappa = \{x\}$ and the security level of the network is 0, is shown as follows:

```
Pi.proc_leakage
Process.COM(
Process.INPUT("x","y",Process.INPUT("y","z",Process.NULL)),
Process.REST("Data",Process.REST("Info",Process.OUTPUT("x","Data",
Process.OUTPUT("Data","Info",Process.NULL))))))
[Process.INPUT("x","e",Process.NULL)] 1
Security_Levels.L2(1,
Security_Levels.L1(1,Security_Levels.L1(1,Security_Levels.L0(-1))),
Security_Levels.L1(1,Security_Levels.L1(1,Security_Levels.L1(1,
Security_Levels.L1(1,Security_Levels.L0(-1))))))
0 ["x"];;
```

With the security policy made simple by specifying the same security level, 1, for all the subprocesses of the system (except for the intruder process, which is running at level 0). The following result is obtained:

```
- :  Process.name list = ["Data"; "Info"; "x";]
```

Which shows that the above names have been leaked to the intruder.

6.3 Spicasso: A Spi-Calculus Analyser for Secrecy and Security Objectives

The Spicasso program is a direct extension of Picasso to cover processes specified in the spi calculus. The program represents an implementation of the abstract semantics of the spi calculus given in Section 4.3.2. The main input to the program is the specification of the process under analysis, along with context and uniformity information. The main modules of the program are stated in the following paragraphs.

Module `Term`. This module defines the structure of terms in the spi calculus. The main data type in this module is `term`, and its definition is shown in Figure 6.3, where we have, for simplicity, considered 2-element tuples (pairs) only.

```
term  =  NAME of string
      |  VAR of string
      |  PUB_KEY of term
      |  PRV_KEY of term
      |  PAIR of term * term
      |  SECCIPHER of term * term
      |  PUBCIPHER of term * term
      |  SIGNATURE of term * term
```

Figure 6.3: The `term` data type in Spicasso.

`SECCIPHER` is the secret-key ciphertext constructor, `PUBCIPHER` is the public-key ciphertext constructor and `SIGCIPHER` is the digital signature constructor. The first element of each constructor represents the plaintext and the second represents the encryption/signature key. Names are represented by `NAME`, variables by `VAR`, private keys by `PRV_KEY` and public keys by `PUB_KEY`. Secret session keys are instantiated as names, constructed by `NAME`.

For example, the complex term $[\![(x,(n,\{m\}_{k'}))]\!]_{k^-}$ is represented as follows:

```
Term.SIGNATURE(Term.PAIR(Term.VAR("x"),
Term.PAIR(Term.NAME("n"),Term.SECCIPHER(Term.NAME("m"), Term.NAME("k'")))),
Term.PRV_KEY(Term.NAME("k")))
```

Module `Process`**.** The module denotes what a process is in the spi calculus. This is defined in Figure 6.4. The module also provides operations for substitutions of terms and the renaming of bound names, variables and tags.

```
and process  =  NULL
             |  INPUT of term * term * process
             |  OUTPUT of term * term * process
             |  COND of term * term * process * process
             |  REST of term * process
             |  COM of process * process
             |  REP of process
             |  SPLIT of term * term * term * process * process
             |  SECDECRYPT of term * term * term * process * process
             |  PUBDECRYPT of term * term * term * process * process
             |  SIGVERIFY of term * term * term * process * process
```

Figure 6.4: The `process` data type in Spicasso.

Although the use of type `term` is generic in the definition of the `process` type, we still assume the appropriate usage of names as channels and secret session keys. Other keys must also have the appropriate type. In the case of `SPLIT`, the first term is the pair to be split. The subsequent terms are the variables instantiated by the two elements of the pair. These variables are bound to the first process. Similarly, for the case of the secret-key decryption construct, `SECDECRYPT`, the first term represents the ciphertext to be decrypted, using the second term (a name). The result, if successful, is bound to the third term in the first process. The same applies to public-key decryption, `PUBDECRYPT`, and digital signature verification, `SIGVERIFY`.

The process module also includes the environments, `tag_to_term` and `term_to_tag`, mapping tags to terms and vice versa. For example, the term $[\{(x^{\dot{t}3}, (n^{\dot{t}2}, \{m^{\dot{t}1}\}_{k'}^{\ddot{t}4})^{\ddot{t}3})^{\ddot{t}2}\}]_{k-}^{\ddot{t}1}$ yields the tags (note that $\ddot{t}$ becomes `tt` and $\dot{t}$ becomes `t`):

`Term.NAME(tt1)`, `Term.NAME(tt2)`, `Term.NAME(tt3)`, `Term.NAME(tt4)`,
`Term.NAME(t1)`, `Term.NAME(t2)` and `Term.NAME(t3)`

each of which has a value in the `tag_to_term` environment corresponding to a different substructure of the above term. For example, applying `tag_to_term(Term.NAME(tt1))` yields:

```
Term.SIGNATURE(Term.PAIR(Term.VAR("x"),
Term.PAIR(Term.NAME("n"),Term.SECCIPHER(Term.NAME("m"), Term.NAME("k'")))),
Term.PRV_KEY(Term.NAME("k")))
```

Whereas applying `term_to_tag(Term.VAR("x"))` results in `Term.NAME("t3")`. For example, the following system:

$$(\nu\, k)\ (\overline{a}\langle\{(M^{\dot{t}1}, N^{\dot{t}2})^{\ddot{t}2}\}_k^{\ddot{t}1}\rangle \quad | \quad a(x).case\ x\ of\ \{y^{\dot{t}4}\}_k\ in\ let\ (u,w) = y\ in\ \overline{a}\langle u^{\dot{t}3}\rangle)$$

is represented by the following type instantiation:

```
Process.REST(Term.NAME("k"),Process.COM(Process.OUTPUT(Term.NAME("a"),
Term.NAME(tt1),Process.NULL),Process.INPUT(Term.NAME("a"),Term.VAR("x"),
Process.SECDECRYPT(Term.VAR("x"),Term.NAME("k"),Term.NAME("t4"),
Process.SPLIT(Term.VAR("y"),Term.VAR("u"),Term.VAR("w"),
Process.OUTPUT(Term.NAME("a"),Term.NAME("t3"),Process.NULL),Process.NULL),
Process.NULL))))
```

Using the `tag_to_term` function, it is possible to retrieve the value of each of the tags, `tt1`, `t3` and `t4` included in the above type instantiation. Similarly, using the `term_to_tag` function, it is possible to convert any term to its tag.

Module `Phi`. This module is the same as in the Picasso tool, holding mappings from variables to sets of tags. The main type here is `phi = (term * term list) list`, where the generic `term` will only be used for the particular case of `NAME`. For example, consider the following $\phi_{\mathcal{A}}$ environment revisited from the previous section:

$$\phi_{\mathcal{A}} = \begin{bmatrix} x & \mapsto & \{t_1\} \\ y & \mapsto & \{t_1, t_2\} \\ z & \mapsto & \{t_1, t_2, t_3\} \end{bmatrix}$$

Which instantiates the following type:

```
- :  (Process.term * Process.term list) list =
[(Process.NAME "x", [Process.NAME "t1"]);
(Process.NAME "y", [Process.NAME "t1"; Process.NAME "t2"]);
(Process.NAME "z", [Process.NAME "t1"; Process.NAME "t2";
Process.NAME "t3"])]
```

Module `Rho`**.** This module is the same as in the Picasso tool described in the previous section. The main type is:

```
rho = Process.process list
```

Which holds a list of processes that are running in parallel with the current analysed process.

Module `Spicasso`**.** This is the main abstract interpretation module in Spicasso. The module contains the following function:

```
val spicasso :  Process.process -> Process.process list -> int ->
Process.name list -> (Process.term * Process.term) list ->
(Process.term * Process.term list) list
```

which represents the abstract semantic relation $\mathcal{A}^{spi}(\llbracket P \rrbracket)\ \rho\ \phi_{\mathcal{A}}$. The function takes as inputs the specification of the process to be analysed P, a list of initial processes running in parallel ρ, a number representing the abstraction constraint n, a list of names representing the initial knowledge of the intruder and finally, a list of pairs representing the mappings from terms to their tags. The outcome is the list representing $\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, x)$ for each variable, x, in the domain of $\phi_{\mathcal{A}}$. The variable, `intruder`, is preserved for the final knowledge of the intruder process, κ, which is modelled by the capabilities of the Dolev-Yao attacker.

Taking the system:

$$(\nu\, k)\ (\overline{a}\langle\{(M^{\dot{t}1}, N^{\dot{t}2})^{\ddot{t}2}\}_k^{\ddot{t}1}\rangle \quad | \quad a(x).case\ x\ of\ \{y^{\dot{t}4}\}_k\ in\ let\ (u,w) = y\ in\ \overline{a}\langle u^{\dot{t}3}\rangle)$$

as an example, and applying the above `Spicasso.spicasso` function:

```
(Spicasso.spicasso
Process.REST(Term.NAME("k"),
Process.COM(
Process.OUTPUT(Term.NAME("a"),
Term.SECCIPHER(Term.PAIR(Term.NAME("M"),Term.NAME("N")),Term.NAME("k")),
Process.NULL),
Process.INPUT(Term.NAME("a"),Term.VAR("x"),
Process.SECDECRYPT(Term.VAR("x"),Term.NAME("k"),Term.VAR("y"),
Process.SPLIT(Term.VAR("y"),Term.VAR("u"),Term.VAR("w"),
Process.OUTPUT(Term.NAME("a"),Term.VAR("u"),Process.NULL),
Process.NULL),Process.NULL)))))
```

```
[]
1
["a";"net"]
[(Term.NAME("M"), Term.NAME("t1"));
(Term.NAME("N"), Term.NAME("t2"));
(Term.PAIR(Term.NAME("M"), Term.NAME("N")), Term.NAME("tt2"));
(Term.NAME("N"), Term.NAME("t2"));
(Term.SECCIPHER(Term.PAIR(Term.NAME("M"),Term.NAME("N")),Term.NAME("k")),
Term.NAME("tt2"));
(Term.VAR("y"), Term.NAME("t4"));
(Term.VAR("u"), Term.NAME("t3"))]
```

will result in the following outcome, which represents the name subset only:

```
- :  (Process.term * Process.term list) list =
[(Process.NAME "u", [Process.NAME "M"]);
(Process.NAME "w", [Process.NAME "N"])];
(Process.NAME "x", [Process.NAME "a"; Process.NAME "net"])];
(Process.NAME "intruder", [Process.NAME "a"; Process.NAME "net";
Process.NAME "M"])];
```

Clearly `u` and `w` will obtain the values of the pair `(M,N)`. However, note that `x` obtains as values the names `a` and `net` from the intruder by communicating over name `a`, which is a channel shared with the intruder. The intruder itself manages to obtain the first element `M` of the pair `(M,N)` using channel `a`.

6.4 Conclusion

In this chapter we have presented two initial prototype implementations for the abstract interpretations for the π-calculus and the spi calculus languages. The implementation of both prototypes captures the term substitution property, and in the current version of the Picasso prototype, a special function for the detection of process secrecy leaks is implemented. Both prototypes were written in the Objective Caml language (OCaml), a functional language.

Future developments of the tools include the addition of other functions for the detection of security breaches, like the process authenticity breach, the building of a language parser, the improvement of tool performance and the building of a graphical user interface.

Chapter 7

Conclusion and Future Work

The modern advances in the design and implementation of distributed computing systems and the advent of mobile technologies has increased the demands for information security and the protection of computing resources beyond any stage in the past. The task of preventing unauthorized malicious intruders and erroneous programs from compromising the secrecy, authenticity and other desirable security properties is growing into a complex and delicate task as a result of the dynamic nature of network topologies, which modern distributed systems are characterised by. This dynamic nature facilitated a plethora of novel methods by which attacks from such intruders could be mounted. Therefore, it is of great concern to ensure that more secure and robust systems are designed that coincide with the important role information technology plays in modern life activities and business transactions.

One significant approach to the analysis of program security is static analysis, which is used to determine properties about programs and their runtime behaviour prior to their execution. This is necessary to determine whether programs exhibit certain security flaws, like information leakage and lack of authenticity. It is also necessary when designing newer systems and languages that avoid such pitfalls. Therefore, static analysers constitute standard tools often used as part of language compilers. Their applications reach beyond the areas of code safety and security into other areas like program optimisation and transformation.

Due to the importance of program static analysis, it was chosen as the main subject of this thesis, where a denotational abstract interpretation-based framework was introduced for the analysis of mobile systems and cryptographic protocols specified within the models of the π-calculus and its cryptographic extension, the spi calculus. In this concluding chapter, we review the major contributions of this framework and discuss prospects for future extensions and modifications that can benefit the framework.

First, we summarise the work presented in the main chapters of this thesis.

Chapter 3: Nominal Calculi

A domain-theoretic model for processes in the π-calculus was given in this chapter based on Stark's predomain equations [121], where the semantic domain of processes and a denotational function from the language syntax to elements of the domain were fully specified. The resulting denotational semantics is precise, compositional, and includes a labelling mechanism for renaming new instances of bound names introduced as a result of replicated processes. Such a mechanism facilitates tracing copies of names to their origin in the specification. Furthermore, the denotational model was extended to be able to model cryptographic capabilities of processes in the spi calculus. This extension allowed complex data structures, like ciphertexts, digital signatures and tuples to be captured in the resulting semantics. The soundness and adequacy of both models were shown with respect to transitions in the structural operational semantics.

Chapter 4: Abstract Interpretation

In this chapter, abstract interpretations for the π-calculus and the spi calculus were built based on extensions of the denotational semantics presented in Chapter 3. The abstract semantic domain was constructed in order to capture the security-sensitive property of term substitutions. Such substitutions occur as a result of communications in the π-calculus, and communications and cryptographic operations in the spi calculus. The meaning of a process was then given as a mapping from its set of variables (input parameters, local variables), to sets of tags, representing the captured terms. The number of tags and variables constituting this meaning was kept finite by placing an integer limit in order to ensure the termination of the analysis. The resulting interpretation is non-uniform in that it can distinguish between multiple copies (up to the integer limit) of the bound variables, names and tags created as a result of replicated behaviour. The interpretation was then applied to a number of examples including the File Transfer Protocol and the Needham-Schroeder, SPLICE/AS, Otway-Rees, Kerberos, Yahalom and Woo-Lam authentication protocols.

Chapter 5: Security Analysis

The results of the abstract interpretation were used to build security analyses for the detection of breaches in the secrecy and authenticity properties of mobile systems and cryptographic protocols. Breaches in data secrecy were formalised by the *process leakage* property,

whereas breaches in data authenticity were formalised by the *process authenticity breach* property. In the former, the secrecy of data is compromised whenever a low-secrecy process obtains a name that was created by a high-secrecy process. On the other hand, authenticity is compromised when a low-trust process manages to send a name to a high-trust process. The definitions of these properties were used in explaining the results of the abstract interpretations of the examples given in Chapter 4.

Chapter 6: Automatic Tools

The framework, defined in Chapters 4 and 5, was implemented using the functional language of Objective Caml for the cases of the π-calculus and the spi calculus and two prototypes are available, PiCASSO and SpiCASSO. The resulting implementations are closely related to the denotational definition of the abstract interpretation adopted in the framework.

7.1 Research Contributions

The work presented in this thesis contributes directly to the following areas.

7.1.1 Denotational Semantics

The work presented in this thesis has resulted in the definition of a sound and adequate denotational semantics for the languages of the π-calculus and the spi calculus. The π-calculus semantics was based on the predomain equations presented by Stark in [121], whereas the spi calculus semantics was built on an extension of these equations that included a predomain of complex terms, and it permitted the presence of complex terms as messages of output actions. These modifications allowed for the modelling of the cryptographic capabilities of processes in the spi calculus. To the best of our knowledge, it is the first domain-theoretic model for the spi calculus.

7.1.2 The Static Analysis of Nominal Calculi

We have presented an abstract interpretation-based static analysis for reasoning about the term-substitution property in mobile and cryptographic systems that are potentially infinite. The analysis is non-uniform in that it can distinguish between the different instances of bound names (local data) and variables belonging to the different copies of replicated processes. It also caters for the presence of intruder processes (e.g. the network) that

exhibit potentially harmful behaviour, by allowing the specification of the most general attacker to be included in the analysis of any system. Finally, the major novelty about the framework presented in this thesis is that it adopts a denotational rather than an operational approach for the analysis of mobile and cryptographic programs. This approach has the following advantages:

- The resulting theory is often simple and close to its implementation in functional programming. The use of the operational approach in constructing static analyses for mobile systems has often resulted in complex analyses that are difficult to implement (e.g. [126]).

- Certain mathematical concepts that are computationally significant, like domains and least fixed points, are directly available for the theory and implementation of the static analysis.

- Finally, a denotational model allows for program properties (in our case, the term substitution property) to be defined compositionally. In other words, the property of a program can be defined from the properties of its subprograms. Such features provide an interesting research foundation for exploration in the future.

7.1.3 Program Security

The state environment, $\phi_{\mathcal{A}}$, resulting from the abstract interpretation defined within our framework offers a common ground for the definition of security properties based on the fundamental property of term substitution. Thus far, we have provided definitions for the process leakage and authenticity properties. We plan to extend the framework to define other security properties, like communication security, freshness, anonymity and non-interference-based security. Such unifying frameworks exist in the literature, the most notable one being the framework introduced by the team led by Focardi and Gorrieri [57] based on the non-interference property.

The secrecy and authenticity definitions have been used in analysing a number of protocols. These include a simple version of a File Transfer Protocol and the Needham-Schroeder, SPLICE/AS, Otway-Rees, Kerberos, Yahalom and Woo-Lam authentication protocols. Results confirmed the presence of a number of well-known impersonation attacks on the Needham-Schroeder, SPLICE/AS, Otway-Rees (two attacks) and Woo-Lam authentication protocols.

7.2 Future Work

There are several directions towards which the framework presented in this thesis can be extended. We summarise a few ideas in the following sections.

7.2.1 Communication Secrecy

Another security property that can be formalised using an extension of the current framework is the *communication secrecy* property. Communication secrecy refers to the secrecy of communicated data with reference to the secrecy levels of the channels over which the data are communicated. Informally, a communication secrecy breach occurs whenever a high-level name is sent over a low-level channel. The conceptual difference between a communication secrecy breach and a process leakage, as defined in Sections 5.2.1 and 5.2.2, is that the former is a name-channel relationship, whereas the latter is a name-process relationship.

For example, consider the system, $\lceil(\nu\ y)\overline{x}\langle y^t\rangle.P\rceil^l \quad | \quad \lceil x(z).Q\rceil^l$, specified in the π-calculus, where x is a free name and the level of the network is $\bot_L \sqsubset l$. Then, we obtain, $\zeta(y) = \zeta(z) = l$, and, $\zeta(x) = \bot_L$, according to the definition of the $\mathcal{Z}(P)$ function in Section 5.2.1. Analysing the system reveals that $t \in \phi_{\mathcal{A}}(z)$, but according to the above classification of secrecy levels, this does not constitute any process leakage. However, it is clear that y is sent over a channel, x, with low secrecy guarantees (since x is a public channel and its secrecy level is that of the intruder, i.e. $\bot_L$).

To be able to capture breaches in the communication secrecy, the channel over which a term travels must be associated in the abstract interpretation with that name. For example, it is possible to include in the meaning of a process, an environment, $\xi_{\mathcal{A}} : \mathit{Tag} \rightarrow \wp(\mathcal{N})$, mapping every message, as represented by its tag, to a set of possible channels over which the message is sent. Such association could be used then to compare the level of the message to the levels of its possible channels. The presence of a channel with a lower security level indicates an insecure communication.

7.2.2 Message Independence

As we discussed in Section 1.5 of the introductory chapter, other notions of security, like non-interference-based and equivalence-based security, attempt to capture breaches that occur through means subtler than the explicit message passing or message processing, like changing the input/output behaviour of processes. We believe that it is possible to extend the static analysis framework presented here to cover one such security property, known as *message independence*.

The definition of message independence in the spi calculus is closely related to the notion of *testing equivalence*, $\simeq$, as suggested by Abadi and Gordon in [5]. According to this equivalence, two processes are identified by the surrounding context if and only if they exhibit the same sequence of barbs in the presence of a testing process composed in parallel with each of the two processes. Based on this, a process is *message independent* with respect to a particular variable if and only if instantiations of that variable by different names form a class of equivalent processes.

The idea of *barbs* is fundamental in the study of implicit information flow in nominal calculi and finds its roots in synchronisation-based calculi, in particular in the work of Milner in CCS [94]. The importance of this idea arises from the fact that barbs stand for observable channels that are ready for communications, and therefore, may be utilized to implicitly send information from the analysed system to its context. This flow of information utilises messages, names of channels and the sequence in which channels interact with their context.

In the theory of the static analysis framework presented in this thesis, it is possible to detect a weak form of the message independence property. Intuitively, assuming that, $\mathcal{A}^{spi}(\llbracket P \mid I \rrbracket)\ \rho\ \phi_{\mathcal{A}} = \phi_{\mathcal{A}P}$ and $\mathcal{A}^{spi}(\llbracket Q \mid I \rrbracket)\ \rho\ \phi_{\mathcal{A}} = \phi_{\mathcal{A}Q}$, are the results of analysing processes P and Q in parallel with the intruder, I, then P is *abstract testing equivalent* to Q, written as $P \simeq_{\mathcal{A}} Q$, if and only if $\phi_{\mathcal{A}P}(\kappa) = \phi_{\mathcal{A}Q}(\kappa)$. In other words, the intruder is incapable of distinguishing (using its knowledge, κ), between running in parallel with P and running in parallel with Q.

It is possible to demonstrate that the notion of testing equivalence [5] implies abstract testing equivalence, i.e. $\simeq\ \Rightarrow\ \simeq_{\mathcal{A}}$. This result is mainly due to the fact that the abstract interpretation used in this framework does not maintain any temporal features of the set of names substituting a variable. Such temporal features could be used to transmit information. Hence, note for example, that the two processes $\overline{c}\langle a\rangle.\overline{c}\langle b\rangle$ and $\overline{c}\langle b\rangle.\overline{c}\langle a\rangle$ are distinguished by $\simeq$, but are identified by $\simeq_{\mathcal{A}}$.

Now, assume that an *open process*, $P(x)$, is a process that can be instantiated by substituting a variable, x, for any name, m, which yields the instance $P[m/x]$. $P(x)$, is then said to be abstract message-independent with respect to the intruder, I, if and only if $\forall m, n \in \mathcal{N} : P[m/x] \simeq_{\mathcal{A}} P[n/x]$. This implies that choosing any two names in $\mathcal{N}$ to obtain instances of $P(x)$ should always result in the same abstract knowledge of the intruder composed with each of those instances. Hence, the intruder cannot find any difference when abstractly interpreted in parallel with $P[m/x]$ from $P[n/x]$.

7.2.3 Language Extensions

Finally, we mention the important research direction in which we could consider other formalisms for mobility, or extensions of current formalisms, that incorporate concepts related to the security of mobile systems. We give an overview below of two such extensions.

PKI-based Extension

Currently, work is under way[1] in utilising the framework to build an abstract interpretation that captures the term substitution property in an extended version of the spi calculus with primitives for modelling systems with a Public-Key Infrastructure (PKI) state. The *spi-pki* language allows for certain PKI requirements in security protocols to be expressed, for example, the validity of the binding between a public key and its owner, which is maintained through valid digital certificates that link the key to its owner and are signed by a Trusted Third Party (TTP). Such requirements cannot be expressed in the spi calculus.

The extension allows for the authenticity of a particular PKI user to be defined as the capturing of a digital signature that has signed fresh data with a private key, of which the public part could be verified to be validly bound (in the current PKI state) to that user. Capturing digital signatures relies on the term substitution property of our framework.

Location-based Extension

Another possibility of using language extensions would be to include location-based languages, like the mobile ambients calculus and the seal calculus. The main issue involved is the construction of an appropriate denotational semantics that models the concept of locations in a correct manner. Term substitutions could then be selected as the abstract semantic domain and used in formalising security properties of the analysed systems. For example, in the mobile ambients calculus, one such property could be the capturing of the evolution of system topology during which certain ambients may be *present* within parent ambients. This property depends on the manner in which communications take place: by the movement of ambients into and out of other ambients. Hence, it could be possible to have a non-standard meaning of a process denoting an environment that maps each ambient name to a set of ambient names that may exist at runtime within the location of that ambient during runtime. The breach of secrecy then, could be expressed by having an ambient with a lower security level entering inside the boundaries of a higher-level parent ambient or vice versa.

[1] Project IMPROVE - Enterprise Ireland, with David Gray, Geoff Hamilton and David Sinclair.

So, far, there have been a few attempts to extend location-based languages with cryptographic primitives, which add another dimension to the problem of the denotational modelling and abstract interpretation of these languages. The mobile ambients calculus has been extended with primitives for shared-key cryptography in [34]. The seal calculus has also been extended in [22], however, with a more generic model that deals with public-key as well as shared-key cryptography. Moreover, a primitive is suggested for expressing code mobility as passive data. The language allows for the modelling of many features in technologies, like Java and .NET, including for example the modelling of secure downloading and running of Java applets within locations called *sandboxes*. The results show that it is possible to model an example, like the *certified email protocol* suggested by [4], which relies heavily on the concepts of cryptography and code mobility. Future research plans include extending the current static analysis framework to the crypto-loc language.

Bibliography

[1] Martín Abadi. Secrecy by typing in security protocols. In Martín Abadi and Takayasu Ito, editors, *Proceedings of the* 3rd *International Symposium on Theoretical Aspects of Computer Software*, volume 1281 of *Lecture Notes in Computer Science*, pages 611–638, Sendai, Japan, September 1997. Springer Verlag.

[2] Martín Abadi, Anindya Banerjee, Nevin Heintze, and Jon Riecke. A core calculus of dependency. In *Proceedings of the* 26th *ACM SIGPLAN-SIGACT on Principles of Programming Languages*, pages 147–160, San Antonio, USA, January 1999. ACM Press.

[3] Martín Abadi and Bruno Blanchet. Analyzing security protocols with secrecy types and logic programs. In *Proceedings of the* 29th *Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 33–44, Portland, USA, January 2002. ACM Press.

[4] Martín Abadi, Neal Glew, Bill Horne, and Benny Pinkas. Certified Email with a Light On-line Trusted Third Party: Design and Implementation. In *Proceedings of the Eleventh International World Wide Web Conference*, pages 387–395. ACM, May 2002.

[5] Martín Abadi and Andrew Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proceedings of the* 4th *ACM Conference on Computer and Communications Security*, pages 36–47, Zurich, Switzerland, April 1997. ACM Press.

[6] Martín Abadi and Andrew Gordon. A calculus for cryptographic protocols: The spi calculus. Technical Report 414, University of Cambridge Computer Laboratory, January 1997.

[7] Martín Abadi and Roger Needham. Prudent engineering practice for cryptographic protocols. In *Proceedings of the IEEE Symposium on Security and Privacy 1994*, volume 22(1), pages 6–15, Oakland, USA, May 1996. IEEE Press.

[8] Samson Abramsky. A domain equation for bisimulation. *Information and Computation*, 92(2):161–218, June 1991.

[9] Franz Achermann, Markus Lumpe, Jean-Guy Schneider, and Oscar Nierstrasz. Piccola - a small composition language. *Formal Methods for Distributed Processing - A Survey of Object-Oriented Approaches*, pages 403–426, September 2001.

[10] Peter Aczel. Final universes of processes. In Stephen D. Brookes, Michael G. Main, Austin Melton, Michael W. Mislove, and David A. Schmidt, editors, *Proceedings of the* 9^{th} *International Conference in Mathematical Foundations of Programming Semantics*, volume 802 of *Lecture Notes in Computer Science*, pages 1–28, New Orleans, USA, April 1994. Springer Verlag.

[11] Roberto Amadio and Denis Lugiez. On the reachability problem in cryptographic protocols. In Catuscia Palamidessi, editor, *Proceedings of the* 11^{th} *International Conference on Concurrency Theory*, volume 1877 of *Lecture Notes in Computer Science*, pages 380–394, Pennsylvania, USA, August 2000. Springer Verlag.

[12] Roberto M. Amadio, Denis Lugiez, and Vincent Vanackére. On the symbolic reduction of processes with cryptographic functions. *Theoretical Computer Science*, 290(1):695–740, January 2003.

[13] Andrew W. Appel and Edward W. Felten. Proof-carrying authentication. In *Proceedings of the* 6^{th} *ACM Conference on Computer and Communications Security*, pages 52–62, Singapore, Singapore, November 1999. ACM Press.

[14] Benjamin Aziz and Geoff Hamilton. A denotational semantics for the π-calculus. In *Proceedings of the* 5^{th} *Irish Workshop in Formal Methods*, Electronic Workshops in Computing, Dublin, Ireland, July 2001. British Computing Society Publishing.

[15] Benjamin Aziz and Geoff Hamilton. A privacy analysis for the π-calculus: The denotational approach. In *Proceedings of the* 2^{nd} *Workshop on the Specification, Analysis and Validation for Emerging Technologies*, number 94 in Datalogiske Skrifter, Copenhagen, Denmark, July 2002. Roskilde University.

[16] Roland Backhouse. *Syntax of Programming Languages: Theory and Practice*. Prentice Hall International, 1979.

[17] John W. Backus, Friedrich L. Bauer, Julien Green, C. Katz, John L. McCarthy, Alan J. Perlis, Heinz Rutishauser, Klaus Samelson, Bernard Vauquois, Joseph Henry Wegstein, Adriaan van Wijngaarden, Michael Woodger, and Peter Naur. Revised report on the algorithmic language algol 60. *Communications of the ACM*, 6(1):1–17, January 1963.

[18] Lujo Bauer, Michael A. Schneider, and Edward W. Felten. A proof-carrying authorization system. Technical Report CS–TR–638–01, Department of Computer Science, Princeton University, April 2001.

[19] D.E. Bell and L.J. La Padula. Secure computer systems: Unified exposition and multics interpretation. Technical Report ESD–TR–75–306, Mitre Corporation, Bedford, M.A., U.S.A., July 1975.

[20] K. G. Biba. Integrity consideration for secure computer systems. Technical Report ESDTR–76–372, Mitre Corporation, Bedford, M.A., U.S.A., April 1977.

[21] Bruno Blanchet. From secrecy to authenticity in security protocols. In Manuel V. Hermenegildo and German Puebla, editors, *Proceedings of the* 9^{th} *International Symposium in Static Analysis*, volume 2477 of *Lecture Notes in Computer Science*, pages 342–359, Madrid, Spain, September 2002. Springer Verlag.

[22] Bruno Blanchet and Benjamin Aziz. A calculus for secure mobility. In Vijay A. Saraswat, editor, *Proceedings of the* 8^{th} *Asian Computing Science Conference*, volume 2896 of *Lecture Notes in Computer Science*, pages 188–204, Mumbai, India, December 2003. Springer Verlag.

[23] Chiara Bodei and Pierpaolo Dagano. Primitives for authentication in process algebra. *Theoretical Computer Science*, 283(2):271–304, June 2002.

[24] Chiara Bodei, Pierpaolo Dagano, Flemming Nielson, and Hanne Riis Nielson. Control flow analysis for the π-calculus. In *Proceedings of the* 9^{th} *Conference on Concurrency Theory*, volume 1466 of *Lecture Notes in Computer Science*, pages 84–98, Nice, France, September 1998. Springer Verlag.

[25] Chiara Bodei, Pierpaolo Dagano, Flemming Nielson, and Hanne Riis Nielson. Static analysis of processes for no read-up and no write-down. In *Proceedings of the Conference on Foundations of Software Science and Computation Structures*, volume 1578 of *Lecture Notes in Computer Science*, pages 120–134, Lisbon, Portugal, March 1999. Springer Verlag.

[26] Chiara Bodei, Pierpaolo Dagano, Flemming Nielson, and Hanne Riis Nielson. Static analysis for secrecy and non-interference in networks of processes. In *Proceedings of the* 6th *International Conference in Parallel Computing Technologies*, volume 2127 of *Lecture Notes in Computer Science*, pages 27–41, Novosibirsk, Russia, September 2001. Springer Verlag.

[27] Chiara Bodei, Pierpaolo Dagano, Flemming Nielson, and Hanne Riis Nielson. Static analysis for the π-calculus with applications to security. *Information and Computation*, 168(1):68–92, July 2001.

[28] Colin Body and Wenbo Mao. On a limitation of ban logic. In *Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques*, volume 765 of *Lecture Notes in Computer Science*, pages 240–247, Lofthus, Norway, May 1994. Springer Verlag.

[29] Michele Boreale. Symbolic trace analysis of cryptographic protocols. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *Proceedings of the* 28th *International Colloquium on Automata, Languages and Programming*, volume 2076 of *Lecture Notes in Computer Science*, pages 667–681, Crete, Greece, July 2001. Springer Verlag.

[30] Michele Boreale and Maria Grazia Buscemi. Experimenting with sta: A tool for automatic analysis of security protocols. In *Proceedings of the ACM Symposium on Applied Computing*, pages 281–285, Madrid, Spain, March 2002. ACM Press.

[31] Gérard Boudol. Asynchrony and the π-calculus. Technical Report 1702, INRIA-Sophia Antipolis, Sophia Antipolis, France, May 1992.

[32] Gérard Boudol and Ilaria Castellani. Non-interference for concurrent programs and thread systems. *Theoretical Computer Science*, 281(1):109–130, June 2002.

[33] Roberto Bruni and U. Montanari. Cartesian closed double categories, their lambda-notation, and the pi-calculus. In *Proceedings of the* 14th *Symposium on Logic in Computer Science*, pages 246–265, Trento, Italy, July 1999. IEEE Computer Society Press.

[34] Michele Bugliesi, Silvia Crafa, Amela Prelić, and Vladimiro Sassone. Secrecy in Untrusted Networks. In *13th International Colloquium on Automata, Languages and Programming (ICALP'03)*, volume 2719 of *Lecture Notes in Computer Science*, pages 969–983. Springer Verlag, July 2003.

[35] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. In *Proceedings of the Royal Society of London A*, volume 426, pages 233–271, 1989.

[36] Nadia Busi and Roberto Gorrieri. A petri net semantics for π-calculus. In *Proceedings of the* 6^{th} *International Conference on Concurrency Theory*, volume 962 of *Lecture Notes in Computer Science*, pages 145–159, Philadelphia, PA, USA, August 1995. Springer Verlag.

[37] Luca Cardelli, Giorgio Ghelli, and Andrew Gordon. Secrecy and group creation. In *Proceedings of the* 11^{th} *International Conference on Concurrency Theory*, volume 1877 of *Lecture Notes in Computer Science*, pages 365–379, Penn State University, State College, Pennsylvania, USA, August 2000. Springer Verlag.

[38] Luca Cardelli and Andrew Gordon. Mobile ambients. In *Proceedings of the* 1^{st} *International Conference on the Foundations of Software Science and Computation Structures*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155, Lisbon, Portugal, March 1998. Springer Verlag.

[39] Gian Luca Cattani, Ian Stark, and Glenn Winskel. Presheaf models for the π-calculus. In *Proceedings of the* 7^{th} *International Conference on Category Theory and Computer Science*, volume 1290 of *Lecture Notes in Computer Science*, pages 106–126, S. Margherita Ligure, Italy, April 1997. Springer Verlag.

[40] Iliano Cervesato. The dolev-yao intruder is the most powerful attacker. In J. Halpern, editor, *Proceedings of the* 16^{th} *Annual Symposium on Logic in Computer Science*, pages 246–265, Boston, MA, U.S.A., June 2001. IEEE Computer Society Press.

[41] Noam Chomsky. Three models for the description of language. *IRE transactions on information theory*, IT–2(3):113–124, September 1956.

[42] Jacob Clark and Jeremy Jacob. On the security of recent protocols. *Information Processing Letters*, 56(3):151–155, November 1995.

[43] Rance Cleaveland, Joachim Parrow, and Bernhard Steffen. The concurrency workbench: A semantic based tool for the verification of concurrent systems. *ACM Transactions on Programming Languages and Systems*, 15(1):36–72, January 1993.

[44] Véronique Cortier. Observational equivalence and trace equivalence in an extension of the spi calculus. Technical Report LSV-02-3, Laboratoire Spécification et Vérification, ENS de Cachan, France, March 2002.

[45] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the* 4th *ACM Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, U.S.A., January 1977. ACM Press.

[46] Dorothy Denning. A lattice model of secure information flow. *ACM Transactions on Programming Languages and Systems*, 19(5):236–243, May 1976.

[47] Danny Dolev and A. Yao. On the security of public key protocols. In *Proceedings of the* 22nd *Annual Symposium on Foundations of Computer Science*, pages 350–357, October 1981.

[48] Matthew Dwyer. Data flow analysis frameworks for concurrent programs. Technical Report 95–8, Kansas State University, Kansas, U.S.A., 1995.

[49] A.S. Elkjær, M. Höhle, H. Hüttel, and K. Overgård. Towards automatic bisimilarity checking in the spi calculus. *Combinatorics, Computation & Logic, Australian Computer Science Communications*, 21(3):175–189, January 1999.

[50] Urban Engberg and Mogens Nielsen. A calculus of communicating systems with name-passing. Technical Report DAIMI PB-208, Computer Science Department, University of Aarhus, Aarhus, Denmark, 1986.

[51] Jérôme Feret. Confidentiality analysis of mobile systems. In *Proceedings of the* 7th *International Static Analysis Symposium*, volume 1824 of *Lecture Notes in Computer Science*, pages 135–154, University of California, Santa Barbara, USA, June 2000. Springer Verlag.

[52] Jérôme Feret. Occurrence counting analysis for the pi-calculus. In Patrick Cousot, Eric Goubault, Jeremy Gunawardena, Maurice Herlihy, Martin Raussen, and Vladimiro Sassone, editors, *GEometry and Topology in COncurrency theory*, volume 39 of *Electronic Notes in Theoretical Computer Science*, PennState, USA, August 2001. Elsevier Science Publishers.

[53] Jérôme Feret. Dependency analysis of mobile systems. In *Proceedings of the* 11th *European Symposium on Programming*, volume 2305 of *Lecture Notes in Computer Science*, pages 314–330, Grenoble, France, April 2002. Springer Verlag.

[54] Marcelo Fiore, Eugenio Moggi, and Davide Sangiorgi. A fully-abstract model for the π-calculus. In *Proceedings of the* 11th *Annual IEEE Symposium On Logic In Computer*

Science, pages 43–54, New Brunswick, New Jersey, USA, July 1996. IEEE Computer Society Press.

[55] Riccardo Focardi and Roberto Gorrieri. A classification of security properties for process algebras. *Journal of Computer Security*, 3(1):5–33, 1995.

[56] Riccardo Focardi and Roberto Gorrieri. The compositional security checker: A tool for the verification of information flow security properties. *IEEE Transactions on Software Engineering*, 23(9):550–571, September 1997.

[57] Riccardo Focardi and Roberto Gorrieri. Classification of security properties (part i: Information flow). In Riccardo Focardi and Roberto Gorrieri, editors, *Tutorials of the* 2nd *International School on Foundations of Security Analysis and Design*, volume 2171 of *Lecture Notes in Computer Science*, pages 331–396. Springer Verlag, 2001.

[58] Riccardo Focardi, Roberto Gorrieri, and Fabio Martinelli. Secrecy in security protocols as non interference. In Steve Schneider and Peter Ryan, editors, *Proceedings of DERA/RHUL Workshop on Secure Architectures and Information Flow*, volume 32 of *Electronic Notes in Theoretical Computer Science*, 1999.

[59] Riccardo Focardi, Roberto Gorrieri, and Fabio Martinelli. Message authentication through non interference. In Teodor Rus, editor, *Proceedings of International Conference on Algebraic Methodology And Software Technology*, volume 1816 of *Lecture Notes in Computer Science*, pages 258–272, Iowa City, Iowa, USA, May 2000. Springer Verlag.

[60] Riccardo Focardi, Roberto Gorrieri, and Fabio Martinelli. Non interference for the analysis of cryptographic protocols. In Ugo Montanari, José D.P. Rolim, and Emo Welzl, editors, *Proceedings of the* 27th *International Colloquium on Automata, Languages and Programming*, volume 1853 of *Lecture Notes in Computer Science*, pages 354–372, Malaga, Spain, July 2000. Springer Verlag.

[61] Riccardo Focardi and Fabio Martinelli. A uniform approach for the analysis of cryptographic protocols. In *Proceedings of the* 2nd *Conference on Security in Communication Networks*, Amalfi, Italy, September 1999.

[62] Riccardo Focardi and Fabio Martinelli. A uniform approach for the definition of security properties. In Jeannette M. Wing, Jim Woodcock, and Jim Davies, editors, *Proceedings of World Congress on Formal Methods*, volume 1708 of *Lecture Notes in*

Computer Science, pages 794–813, Toulouse, France, September 1999. IEEE Computer Society Press.

[63] Warwaick Ford and Michael S. Baum. *Secure Electronic Commerce: Building the Infrastructure for Digital Signatures and Encryption*. Prentice Hall PTR, 2 edition, December 2000.

[64] Cédric Fournet and Georges Gonthier. The reflexive chemical abstract machine and the join-calculus. In *Proceedings of the* 23^{rd} *ACM Symposium on Principles of Programming Languages*, pages 372–385, St. Petersburg Beach, Florida, USA, January 1996. ACM Press.

[65] Marc Gengler and Matthieu Martel. Self-applicable partial evaluation for the pi-calculus. In *Proceedings of the ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, pages 36–46, Amsterdam, The Netherlands, June 1997. ACM Press.

[66] Stefania Gnesi, Diego Latella, and Gabriele Lenzini. A "brutus" model checking of a spi-calculus dialect. In *Proceedings of the Workshop on Formal Methods and Computer Security*, University of British Columbia, Vancouver, Canada, June 2000.

[67] Joseph A. Goguen and Jose Meseguer. Security policy and security models. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 11–20, Oakland, CA, USA, April 1982. IEEE Computer Society Press.

[68] Dieter Gollmann. *Computer Security*. John Wiley & Son Ltd, February 1999.

[69] Takis Hartonas. Denotational semantics for a higher-order extension of the monadic π-calculus. Technical Report Math&CS1999-1, Technical Education Institute (TEI) of Larissa, Larissa, Greece, 1999.

[70] Matthew Hennessy. The security pi calculus and non-interference. Technical Report 05/2000, University of Sussex, Sussex, UK, 2000.

[71] Matthew Hennessy. A fully abstract denotational semantics for the π-calculus. *Theoretical Computer Science*, 278(1–2):53–89, May 2002.

[72] Matthew Hennessy and John Riely. Resource access control in systems of mobile agents. In *Proceedings of the* 3^{rd} *International Workshop on High-Level Concurrent Languages*, volume 16(3) of *Electronic Notes in Theoretical Computer Science*, pages 1–15, Nice, France, September 1998. Elsevier.

[73] Matthew Hennessy and John Riely. Information flow vs. resource access in the asynchronous pi-calculus. In Ugo Montanari, José D.P. Rolim, and Emo Welzl, editors, *Proceedings of the* 27^{th} *International Colloquium on Automata, Languages and Programming*, volume 1853 of *Lecture Notes in Computer Science*, pages 415–427, Geneva, Switzerland, July 2000. Springer Verlag.

[74] Mark Hepburn and David Wright. Trust in the pi-calculus. In *Proceedings of the* 3^{rd} *Conference on Principles and Practice of Declarative Programming*, pages 103–114, Florence, Italy, September 2001. ACM Press.

[75] C.A.R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, August 1978.

[76] Kohei Honda and Mario Tokoro. On asynchronous communication semantics. In Mario Tokoro, Oscar Nierstrasz, and Peter Wegner, editors, *Proceedings of the ECOOP'91 Workshop on Object-Based Concurrent Computing*, volume 612 of *Lecture Notes in Computer Science*, pages 21–51, Geneva, Switzerland, July 1992. Springer Verlag.

[77] Kohei Honda, Vasco Thudichum Vasconcelos, and Nobuko Yoshida. Secure information flow as typed process behaviour. In Gert Smolka, editor, *Proceedings of the* 9^{th} *European Symposium on Programming*, volume 1782 of *Lecture Notes in Computer Science*, pages 180–199, Berlin, Germany, March 2000. Springer Verlag.

[78] Furio Honsell, Marina Lenisa, Ugo Montanari, and Marco Pistore. Final semantics for the π-calculus. In David Gries and Willem P. de Roever, editors, *Proceedings of the IFIP Working Conference on Programming Concepts and Methods*, volume 125 of *IFIP Conference Proceedings*, pages 21–51, Shelter Island, New York, USA, June 1998. Chapman & Hall.

[79] Antti Huima. Efficient infinite-state analysis of security protocols. In *Proceedings of the FLOC 1999 Formal Methods and Security Protocols Workshop*, pages 21–51, Trento, Italy, July 1999.

[80] Tzonelih Hwang and Yung-Hsiang Chen. On the security of splice/as: The authentication system in wide internet. *Information Processing Letters*, 53(2):97–101, January 1995.

[81] Lalita Jategaonkar Jagadeesan and Radha Jagadeesan. Causality and true concurrency: A data-flow analysis of the pi-calculus. In Vangalur S. Alagar and Maurice

Nivat, editors, *Proceedings of the* 4^{th} *International Conference in Algebraic Methodology and Software Technology*, volume 936 of *Lecture Notes in Computer Science*, pages 277–291, Montreal, Canada, July 1995. Springer Verlag.

[82] H.B.M. Jonkers. Abstract storage structures. *Algorithmic Languages*, pages 321–343, 1981.

[83] John B. Kam and Jeffrey D. Ullman. Monotone data flow analysis frameworks. *Acta Informatica*, 7:305–317, January 1977.

[84] Gary A. Kildall. A unified approach to global program optimization. In *Proceedings of ACM Symposium on Principles of Programming Languages*, pages 194–206, Boston, Massachusetts, USA, October 1973. ACM Press.

[85] Butler Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, 1973.

[86] Barbara Liskov, Alan Snyder, Russell R. Atkinson, and Craig Schaffert. Abstraction mechanisms in clu. *Communications of the ACM*, 20(8):564–576, 1977.

[87] Gavin Lowe. Breaking and fixing the needham-schroeder public-key protocol using fdr. In Tiziana Margaria and Bernhard Steffen, editors, *Proceedings of the Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166, Passau, Germany, March 1996. Springer Verlag.

[88] Thomas J. Marlowe and Barbara G. Ryder. Properties of data flow frameworks - a unified model. *Acta Informatica*, 28(2):121–163, 1990.

[89] John McCarthy. A basis for a mathematical theory of computation. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, pages 33–70, Amsterdam, Holland, 1965. North-Holland.

[90] John McLean. Security models and information flow. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 180–189, Oakland, California, USA, May 1990. IEEE Computer Society Press.

[91] Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, October 1996.

[92] Steven P. Miller, Clifford Neuman, Jeffery I. Schiller, and Jerome H. Saltzer. Kerberos authentication and authorization system - project athena technical plan. Technical Report Section E.2.1, MIT, USA, October 1987.

[93] Robin Milner. A calculus of communicating systems. *Lecture Notes in Computer Science*, 92, 1980.

[94] Robin Milner. *Communication and Concurrency*. Prentice Hall International, 1989.

[95] Robin Milner. The polyadic π-calculus: A tutorial. Technical Report ECS-LFCS-91-180, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, UK, 1991.

[96] Robin Milner. *Communicating and Mobile Systems: The Pi-Calculus*. Cambridge University Press, 1999.

[97] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes (parts i & ii). *Information and Computation*, 100(1):1–77, September 1992.

[98] Eugenio Moggi. Notions of computation and monads. *Information and Control*, 93(1):55–92, July 1991.

[99] David Monniaux. Abstracting cryptographic protocols with tree automata. In Agostino Cortesi and Gilberto Filé, editors, *Proceedings of the* 6^{th} *International Static Analysis Symposium*, volume 1694 of *Lecture Notes in Computer Science*, pages 149–163, Venice, Italy, September 1999. Springer Verlag.

[100] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[101] Flemming Nielson and Hanne Riis Nielson. Flow logic and operational semantics. In Andrew Pitts Andrew Gordon and Carolyn Talcott, editors, *Electronic Notes in Theoretical Computer Science*, volume 10. Elsevier Science Publishers, 2000.

[102] Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer-Verlag, 1999.

[103] Donal O'Mahony, Michael Peirce, and Hitesh Tewari. *Electronic Payment Systems for E-Commerce*. Artech House, 1997.

[104] David Otway and Owen Rees. Efficient and timely mutual authentication. *Operating Systems Review*, 21(1):8–10, January 1987.

[105] Larry C. Paulson. On two formal analyses of the yahalom protocol. Technical Report 432, The Computer Laboratory, University of Cambridge, UK, 1997.

[106] Laurence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1/2):85–128, January 1998.

[107] Carl Adam Petri. Kommunikation mit automaten. Technical Report 2, Schriften des Institutes für Instrumentelle Mathematik, 1962.

[108] Benjamin Pierce and David Turner. Pict: A programming language based on the pi-calculus. Technical Report CSCI 476, Computer Science Department, Indiana University, 1997.

[109] Andrew Pitts. Relational properties of domains. *Information and Computation*, 127(2):66–90, June 1996.

[110] François Pottier. A simple view of type-secure information flow in the π-calculus. In *Proceedings of the* 15^{th} *IEEE Computer Security Foundations Workshop*, pages 320–330, Cape Breton, Nova Scotia, Canada, June 2002. IEEE Press.

[111] Amela Prelić. Cryptographic Primitives in Ambient Calculi. Master's thesis, Università Ca' Foscari, July 2002.

[112] James Riely and Matthew Hennessy. Trust and partial typing in open systems of mobile agents. In *Proceedings of the* 26^{th} *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 93–104, San Antonio, TX, USA, January 1999. ACM Press.

[113] John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12(1):23–41, February 1965.

[114] Davide Sangiorgi and David Walker. *The Pi-Calculus - A Theory of Mobile Processes*. Cambridge University Press, Cambridge, UK, 2001.

[115] David Schmidt. *Denotational Semantics: A Methodology for Language Development*. Allyn & Bacon, 1986.

[116] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, 2 edition, 1995.

[117] Dana S. Scott. Outline of a mathematical theory of computation. In *Proceedings of the* 4th *Annual Princeton Conference on Information Science and Systems*, pages 169–176, Princeton University, Princeton, New Jersey, USA, March 1970.

[118] Dana S. Scott and Christopher Strachey. Towards a mathematical semantics for computer languages. In J. Fox, editor, *Proceedings of Symposium on Computers and Automata*, pages 19–46, Polytechnic Institute of Brooklyn, New York, USA, April 1971. ACM Press.

[119] Olin Shivers. Control flow analysis in scheme. In *Proceedings of the ACM SIGPLAN'88 Conference on Programming Language Design and Implementation*, volume 23(7) of *ACM SIGPLAN Notices*, pages 164–174, Atlanta, Georgia, USA, June 1988. ACM Press.

[120] Geoffrey Smith and Dennis M. Volpano. Secure information flow in a multi-threaded imperative language. In *Proceedings of the* 25th *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 355–364, San Diego, CA, USA, January 1998. ACM Press.

[121] Ian Stark. A fully abstract domain model for the π-calculus. In *Proceedings of the* 11th *Annual IEEE Symposium on Logic in Computer Science*, pages 36–42, New Brunswick, New Jersey, USA, July 1996. IEEE Computer Society.

[122] Joseph E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory.* MIT Press, 1977.

[123] Christopher Strachey. Towards a formal semantics. *Formal Language Description Languages for Computer Programming*, pages 198–220, 1966.

[124] D Sutherland. A model of information. In *Proceedings of the* 9th *National Computer Security Conference*, U. S. National Computer Security Center and U. S. National Bureau of Standards, 1986. IEEE Computer Society.

[125] Robert D. Tennent. *Semantics of Programming Languages.* Prentice Hall International, June 1991.

[126] Arnaud Venet. Abstract interpretation of the π-calculus. In Mads Dam, editor, *Proceedings of the* 5th *LOMAPS Workshop on Analysis and Verification of Multiple-Agent Languages*, volume 1192 of *Lecture Notes in Computer Science*, pages 51–75, Stockholm, Sweden, June 1996. Springer Verlag.

[127] Arnaud Venet. Automatic determination of communication topologies in mobile systems. In Giorgio Levi, editor, *Proceedings of the* 5th *International Static Analysis Symposium*, volume 1503 of *Lecture Notes in Computer Science*, pages 152–167, Pisa, Italy, September 1998. Springer Verlag.

[128] Björn Victor and Faron Moller. The Mobility Workbench — a tool for the π-calculus. In David Dill, editor, *Proceedings of the* 6th *International Conference in Computer-Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 428–440, Stanford, California, USA, June 1994. Springer Verlag.

[129] Jan Vitek and Guiseppe Castagna. Seal: A framework for secure mobile computations. In Henri E. Bal, Boumediene Belkhouche, and Luca Cardelli, editors, *Proceedings of the ICCL workshop: Internet Programming Languages*, volume 1686 of *Lecture Notes in Computer Science*, pages 47–77, Chicago, Illinois, USA, May 1998. Springer Verlag.

[130] Dennis Volpano, Cynthia Irvine, and Geoffrey Smith. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(2,3):167–187, January 1996.

[131] Glynn Winskel. *The Formal Semantics of Programming Languages - An Introduction.* MIT Press, February 1993.

[132] Pawel Wojciechowski and Peter Sewell. Nomadic pict: Language and infrastructure design for mobile agents. In Henri E. Bal, Boumediene Belkhouche, and Luca Cardelli, editors, *Proceedings of the* 1st *International Symposium on Agent Systems and Applications, and the* 3rd *International Symposium on Mobile Agents*, pages 821–826, Palm Springs, California, USA, 1999. IEEE Computer Society Press.

[133] Thomas Y. C. Woo and Simon S. Lam. Authentication for distributed systems. *Computer*, 25(1):39–52, January 1992.

[134] S. Yamaguchi, K. Okayama, and H. Miyahara. Design and implementation of an authentication system in wide internet environment. In *Proceedings of the* 10th *IEEE Region Conference on Computer and Communication Systems.* IEEE Press, 1990.

[135] P. Yang, C.R. Ramakrishnan, and S.A. Smolka. A logical encoding of the π-calculus: Model checking mobile processes using tabled resolution. In Lenore D. Zuck, Paul C. Attie, Agostino Cortesi, and Supratik Mukhopadhyay, editors, *Proceedings of the* 4th *International Conference in Verification, Model Checking, and Abstract Interpretation*, volume 2575 of *Lecture Notes in Computer Science*, pages 116–131, New York, USA, January 2003. Springer Verlag.

Appendix A

Proofs

A.1 Safety of the $\cup_\phi$ operation in the π-calculus (Lemma 1)

$$\forall i \in \{1 \dots n\}, n \in \mathbb{N}, \phi_i \in D_\perp, \phi'_i \in D^\sharp_\perp :$$
$$(\phi = \bigcup_{i=1\dots n}^{\phi} \phi_i) \ \wedge \ (\phi' = \bigcup_{i=1\dots n}^{\phi} \phi'_i) \ \wedge$$
$$(\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi_i, y) \in \phi_i(x) \ \Rightarrow \ \exists t \in \phi'_i(\alpha_k(x)) : \ \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$$

$$\Rightarrow \quad (\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi, y) \in \phi(x) \ \Rightarrow \ \exists t \in \phi'(\alpha_k(x)) : \ \mathit{value\_of}\,(\{t\}) = \{\alpha_k(y)\})$$

The proof proceeds by mathematical induction on sets of environments.

The base case: n=0

$$(\phi = \bigcup_{i=1\dots 0}^{\phi} \phi_i = \phi_0) \ \wedge \ (\phi' = \bigcup_{i=1\dots 0}^{\phi} \phi'_i = \phi_0)$$

$$\Rightarrow \quad (\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi_0, y) \in \phi_0(x) \ \Rightarrow \ \exists t \in \phi'_0(\alpha_k(x)) : \ \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$$

Since, $\forall x \in \mathcal{N} : \phi_0(x) = \phi'_0(\alpha_k(x)) = \emptyset$.

The induction step. First, assume the hypothesis is true for a union of n environments:

$$(\phi = \bigcup_{i=1\dots n}^{\phi} \phi_i) \ \wedge \ (\phi' = \bigcup_{i=1\dots n}^{\phi} \phi'_i) \ \wedge$$
$$(\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi_i, y) \in \phi_i(x) \ \Rightarrow \ \exists t \in \phi'_i(\alpha_k(x)) : \ \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$$

$$\Rightarrow \quad (\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi, y) \in \phi(x) \ \Rightarrow \ \exists t \in \phi'(\alpha_k(x)) : \ \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$$

Proving the induction step demands that the above statement is true for a union of $n+1$ environments. Assuming that, $\phi = \bigcup_{i=1\ldots n}^{\phi} \phi_i$ and $\phi' = \bigcup_{i=1\ldots n}^{\phi} \phi'_i$, as above, we have that:

$\bigcup_{i=1\ldots n+1}^{\phi} \phi_i = \phi \cup_\phi \phi_{n+1}$

$\bigcup_{i=1\ldots n+1}^{\phi} \phi'_i = \phi' \cup_\phi \phi'_{n+1}$

From the definition of $\cup_\phi$ in Section 4.2.1, we have:

$(\phi \cup_\phi \phi_{n+1})(x) = \phi(x) \cup \phi_{n+1}(x)$

$(\phi' \cup_\phi \phi'_{n+1})(x) = \phi'(x) \cup \phi'_{n+1}(x)$

We obtain $\forall y \in \phi(x), y' \in \phi_{n+1}(x) \;\Rightarrow\; y, y' \in (\phi(x) \cup \phi_{n+1}(x))$

and, $\forall t \in \phi'(\alpha_k(x)), t' \in \phi'_{n+1}(\alpha_k(x)) \;\Rightarrow\; t, t' \in (\phi'(\alpha_k(x)) \cup \phi'_{n+1}(\alpha_k(x)))$

By the inductive hypothesis, we have that:

$(\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi, y) \in \phi(x) \quad \Rightarrow \quad \exists t \in \phi'(\alpha_k(x)) : \; \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$

and with the assumption that the new added environments are also safe:

$(\exists y' \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi_{n+1}, y') \in \phi_{n+1}(x) \quad \Rightarrow \quad \exists t' \in \phi'_{n+1}(\alpha_k(x)) : \; \mathit{value\_of}(\{t'\}) = \{\alpha_k(y')\})$

Then by the properties of set union, it is possible to arrive at the following result:

$y \in (\phi(x) \;\cup\; \phi_{n+1}(x))$

$\Rightarrow \quad \exists t \in (\phi'(\alpha_k(x)) \;\cup\; \phi'_{n+1}(\alpha_k(x))) : \; \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\}$

□

A.2 Safety of the abstract semantics of the π-calculus (Theorem 5)

$$\begin{array}{l}\forall P, \rho, \phi_{\mathcal{E}}, \phi_{\mathcal{A}} : \\ (\mathcal{E}^{\pi}[\![P]\!]\ \rho\ \phi_{\mathcal{E}} = (p, \phi'_{\mathcal{E}}))\ \wedge\ (\mathcal{A}^{\pi}[\![P]\!]\ \rho\ \phi_{\mathcal{A}} = \phi'_{\mathcal{A}})\ \wedge \\ (\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, y) \in \phi_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi_{\mathcal{A}}(\alpha_k(x)) : \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\}) \\ \\ \Rightarrow\quad (\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, y) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi'_{\mathcal{A}}(\alpha_k(x)) : \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})\end{array}$$

The proof proceeds by structural induction on P (in all that follows, $snd(x_1, x_2) = x_2$).

The base case: 0

$\mathcal{A}^{\pi}[\![\mathbf{0}]\!]\ \rho\ \phi_{\mathcal{A}} = \phi_{\mathcal{A}}$ by $(\mathcal{A}^{\pi}1)$

$\mathcal{E}^{\pi}[\![\mathbf{0}]\!]\ \rho\ \phi_{\mathcal{E}} = (\emptyset, \phi_{\mathcal{E}})$ by $(\mathcal{E}^{\pi}1)$

Here we have from the antecedent:

$\mathcal{A}^{\pi}[\![P]\!]\ \rho_0\ \phi_{\mathcal{A}} = \phi'_{\mathcal{A}}$

$\mathcal{E}^{\pi}[\![P]\!]\ \rho_0\ \phi_{\mathcal{E}} = (p, \phi'_{\mathcal{E}})$

Since $\phi'_{\mathcal{E}} = \phi_{\mathcal{E}}$ and $\phi'_{\mathcal{A}} = \phi_{\mathcal{A}}$, this satisfies the safety requirement as follows:

$(\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, y) \in \phi_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi_{\mathcal{A}}(\alpha_k(x)) : \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$

$\Rightarrow\quad (\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, y) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi'_{\mathcal{A}}(\alpha_k(x)) : \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$

The induction step.

Case 1: $x(y).P$

$\mathcal{A}^{\pi}[\![x(y).P]\!]\ \rho\ \phi_{\mathcal{A}} = \phi_{\mathcal{A}}$ by $(\mathcal{A}^{\pi}2)$

$\mathcal{E}^{\pi}[\![x(y).P]\!]\ \rho\ \phi_{\mathcal{E}} = (p, \phi_{\mathcal{E}})$ by $(\mathcal{E}^{\pi}2)$

From the antecedent, we have that:

$\mathcal{A}^{\pi}[\![P]\!]\ \rho_0\ \phi_{\mathcal{A}} = \phi'_{\mathcal{A}}$

$\mathcal{E}^{\pi}[\![P]\!]\ \rho_0\ \phi_{\mathcal{E}} = (p, \phi'_{\mathcal{E}})$

Since $\phi'_{\mathcal{E}} = \phi_{\mathcal{E}}$ and $\phi'_{\mathcal{A}} = \phi_{\mathcal{A}}$, this satisfies the safety requirement as follows:

$(\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, y) \in \phi_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi_{\mathcal{A}}(\alpha_k(x)) : \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$

$\Rightarrow \quad (\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, y) \in \phi'_{\mathcal{E}}(x) \ \Rightarrow \ \exists t \in \phi'_{\mathcal{A}}(\alpha_k(x)) : \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$

Case 2: $\overline{x}\langle y\rangle.P$

$\mathcal{A}^{\pi}(\llbracket \overline{x}\langle y^t\rangle.P\rrbracket)\ \rho\ \phi_{\mathcal{A}} \ = \ \bigcup_{\substack{\phi \\ x'(z).P' \in \rho}} \phi'_{\mathcal{A}} \ \cup_{\phi} \ \phi_{\mathcal{A}}$ by $(\mathcal{A}^{\pi}3)$

where, $\phi'_{\mathcal{A}} = \mathcal{R}^{\pi}(\llbracket \{\!|P|\!\}_{\rho} \uplus_{\rho} \rho[P'/x'(z).P']\rrbracket)\ \phi_{\mathcal{A}}[\alpha_k(z) \mapsto \ \phi_{\mathcal{A}}(\alpha_k(z)) \cup \{\alpha_k(t)\}]$

$= \bigcup_{\substack{\phi \\ P \in \rho'}} \mathcal{A}^{\pi}(\llbracket P\rrbracket)\ (\{\!|P|\!\}_{\rho} \uplus_{\rho} \rho[P'/x'(z).P'] \backslash \{\!|P|\!\}_{\rho})\ \phi_{\mathcal{A}}[\alpha_k(z) \mapsto \ \phi_{\mathcal{A}}(\alpha_k(z)) \cup \{\alpha_k(t)\}]$

if, $\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, x) \cap \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, x') \neq \{\}$

$\mathcal{E}^{\pi}(\llbracket \overline{x}\langle y\rangle.P\rrbracket)\ \rho\ \phi_{\mathcal{E}} \ = \ (p, \ \bigcup_{\substack{\phi \\ x'(z).P' \in \rho}} \phi'_{\mathcal{E}} \ \cup_{\phi} \ \phi_{\mathcal{E}})$ by $(\mathcal{E}^{\pi}3)$

where, $\phi'_{\mathcal{E}} = \mathit{snd}(\mathcal{R}^{\pi}(\llbracket \{\!|P|\!\}_{\rho} \uplus_{\rho} \rho[P'/x'(z).P']\rrbracket)\ \phi_{\mathcal{E}}[z \mapsto \ \{\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, y)\}])$

$= \mathit{snd}(p', \bigcup_{\substack{\phi \\ P \in \rho'}} \mathit{snd}(\mathcal{E}^{\pi}(\llbracket P\rrbracket)\ (\{\!|P|\!\}_{\rho} \uplus_{\rho} \rho[P'/x'(z).P'] \backslash \{\!|P|\!\}_{\rho})\ \phi_{\mathcal{E}}[z \mapsto \ \{\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, y)\}]))$

if, $\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, x) = \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, x')$

From the antecedent, we have that:

$(\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, y) \in \phi_{\mathcal{E}}(x) \ \Rightarrow \ \exists t \in \phi_{\mathcal{A}}(\alpha_k(x)) : \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$

Setting $\phi'_{\mathcal{E}} = \phi_{\mathcal{E}}[z \mapsto \ \{\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, y)\}]$ and $\phi'_{\mathcal{A}} = \phi_{\mathcal{A}}[\alpha_k(z) \mapsto \ \phi_{\mathcal{A}}(\alpha_k(z)) \cup \{\alpha_k(t)\}]$, we have that $\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, y) \in \phi'_{\mathcal{E}}(z) \ \Rightarrow \ \exists t \in \phi'_{\mathcal{A}}(\alpha_k(z)) : \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\}$ (by the properties of set union)

$\Rightarrow \quad (\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, y) \in \phi'_{\mathcal{E}}(x) \ \Rightarrow \ \exists t \in \phi'_{\mathcal{A}}(\alpha_k(x)) : \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$

By the induction hypothesis for P: $(\mathcal{E}^{\pi}(\llbracket P\rrbracket)\ \rho''\ \phi'_{\mathcal{E}} = p'', \phi''_{\mathcal{E}}) \ \wedge \ (\mathcal{A}^{\pi}(\llbracket P\rrbracket)\ \rho''\ \phi'_{\mathcal{A}} = \phi''_{\mathcal{A}}) \wedge$

$(\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, y) \in \phi'_{\mathcal{E}}(x) \ \Rightarrow \ \exists t \in \phi'_{\mathcal{A}}(\alpha_k(x)) : \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$

$\Rightarrow \quad (\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi''_{\mathcal{E}}, y) \in \phi''_{\mathcal{E}}(x) \ \Rightarrow \ \exists t \in \phi''_{\mathcal{A}}(\alpha_k(x)) : \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$

By the safety of the $\cup_{\phi}$ relation (Lemma 1), we arrive at the following result:

$p, ((\bigcup_{\substack{\phi \\ x(z).Q \in \rho}} \bigcup_{\substack{\phi \\ P \in \rho'}} \mathit{snd}(\mathcal{E}^{\pi}(\llbracket P\rrbracket)\ (\{\!|P|\!\}_{\rho} \uplus_{\rho} \rho[P'/x'(z).P'] \backslash \{\!|P|\!\}_{\rho})\ \phi'_{\mathcal{E}})) \ \cup_{\phi} \ \phi_{\mathcal{E}}) = p, \phi \quad \wedge$

$(\bigcup_{\substack{\phi \\ x(z).Q \in \rho}} \bigcup_{\substack{\phi \\ P \in \rho'}} \mathcal{A}^{\pi}(\llbracket P\rrbracket)\ (\{\!|P|\!\}_{\rho} \uplus_{\rho} \rho[P'/x'(z).P'] \backslash \{\!|P|\!\}_{\rho})\ \phi'_{\mathcal{A}}) \ \cup_{\phi} \ \phi_{\mathcal{A}} = \phi' \quad \wedge$

$(\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, y) \in \phi'_{\mathcal{E}}(x) \ \Rightarrow \ \exists t \in \phi'_{\mathcal{A}}(\alpha_k(x)) : \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$

$\Rightarrow \quad (\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi, y) \in \phi(x) \ \Rightarrow \ \exists t \in \phi'(\alpha_k(x)) : \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$

Which satisfies the induction step and the safety requirement.

Case 3: *if* $x = y$ *then* P *else* Q

$$\mathcal{A}^\pi(\![\mathit{if}\ x = y\ \mathit{then}\ P\ \mathit{else}\ Q]\!)\ \rho\ \phi_{\mathcal{A}} = \begin{cases} \mathcal{R}^\pi(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)\ \phi_{\mathcal{A}}, & \\ \quad \text{if } \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, x) \cap \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, y) \neq \{\} & \\ \mathcal{R}^\pi(\![\{\!|Q|\!\}_\rho \uplus_\rho \rho]\!)\ \phi_{\mathcal{A}}, & \text{otherwise} \end{cases}$$

where, $\mathcal{R}^\pi(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)\ \phi_{\mathcal{A}} = \bigcup_{P \in \rho'_P}^{\phi} \mathcal{A}^\pi(\![P]\!)\ \rho''_P\ \phi_{\mathcal{A}}$

and, $\mathcal{R}^\pi(\![\{\!|Q|\!\}_\rho \uplus_\rho \rho]\!)\ \phi_{\mathcal{A}} = \bigcup_{Q \in \rho'_Q}^{\phi} \mathcal{A}^\pi(\![Q]\!)\ \rho''_Q\ \phi_{\mathcal{A}}$ (by $\mathcal{A}^\pi 4$)

$$\mathcal{E}^\pi(\![\mathit{if}\ x = y\ \mathit{then}\ P\ \mathit{else}\ Q]\!)\ \rho\ \phi_{\mathcal{E}} = \begin{cases} \mathcal{R}^\pi(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)\ \phi_{\mathcal{E}}, & \text{if } \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, x) = \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, y) \\ \mathcal{R}^\pi(\![\{\!|Q|\!\}_\rho \uplus_\rho \rho]\!)\ \phi_{\mathcal{E}}, & \text{otherwise} \end{cases}$$

where, $\mathcal{R}^\pi(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)\ \phi_{\mathcal{E}} = p,\ \bigcup_{P \in \rho'_P}^{\phi} snd(\mathcal{E}^\pi(\![P]\!)\ \rho''_P\ \phi_{\mathcal{E}})$

and, $\mathcal{R}^\pi(\![\{\!|Q|\!\}_\rho \uplus_\rho \rho]\!)\ \phi_{\mathcal{E}} = q,\ \bigcup_{Q \in \rho'_Q}^{\phi} snd(\mathcal{E}^\pi(\![Q]\!)\ \rho''_Q\ \phi_{\mathcal{E}})$ (by $\mathcal{E}^\pi 4$)

Where, $\rho'_P = \{\!|P|\!\}_\rho \uplus_\rho \rho$, $\rho'_Q = \{\!|Q|\!\}_\rho \uplus_\rho \rho$, $\rho''_P = \rho'_P \backslash \{\!|P|\!\}_\rho$ and $\rho''_Q = \rho'_Q \backslash \{\!|Q|\!\}_\rho$.

By the induction hypothesis for P (Q):

$(\mathcal{E}^\pi(\![P]\!)\ \rho''\ \phi_{\mathcal{E}} = p', \phi'_{\mathcal{E}})\ \wedge\ (\mathcal{A}^\pi(\![P]\!)\ \rho''\ \phi_{\mathcal{A}} = \phi'_{\mathcal{A}})\ \wedge$

$(\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, y) \in \phi_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi_{\mathcal{A}}(\alpha_k(x)) : value\_of(\{t\}) = \{\alpha_k(y)\})$

$\Rightarrow\quad (\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, y) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi'_{\mathcal{A}}(\alpha_k(x)) : value\_of(\{t\}) = \{\alpha_k(y)\})$

By the safety of the $\cup_\phi$ relation (Lemma 1), we arrive at the following result:

$p, (\bigcup_{P \in \rho'}^{\phi} snd(\mathcal{E}^\pi(\![P]\!)\ \rho''\ \phi_{\mathcal{E}})) = p, \phi\quad \wedge\quad (\bigcup_{P \in \rho'}^{\phi} \mathcal{A}^\pi(\![P]\!)\ \rho''\ \phi_{\mathcal{A}}) = \phi'\quad \wedge$

$(\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, y) \in \phi_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi_{\mathcal{A}}(\alpha_k(x)) : value\_of(\{t\}) = \{\alpha_k(y)\})$

$\Rightarrow\quad (\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi, y) \in \phi(x)\ \Rightarrow\ \exists t \in \phi'(\alpha_k(x)) : value\_of(\{t\}) = \{\alpha_k(y)\})$

Which satisfies the induction step and the safety requirement.

Case 4: $P + Q$

$\mathcal{A}^\pi(\![P + Q]\!)\ \rho\ \phi_{\mathcal{A}} = \mathcal{R}^\pi(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)\ \phi_{\mathcal{A}}\ \cup_\phi\ \mathcal{R}^\pi(\![\{\!|Q|\!\}_\rho \uplus_\rho \rho]\!)\ \phi_{\mathcal{A}}\ =$

$(\bigcup_{P \in \rho'_P}^{\phi} \mathcal{A}^\pi(\![P]\!)\ \rho''_P\ \phi_{\mathcal{A}})\ \cup_\phi\ (\bigcup_{Q \in \rho'_Q}^{\phi} \mathcal{A}^\pi(\![Q]\!)\ \rho''_Q\ \phi_{\mathcal{A}})$ (by $\mathcal{A}^\pi 5$)

$\mathcal{E}^\pi(\![P + Q]\!)\ \rho\ \phi_{\mathcal{E}} = p, (snd(\mathcal{R}^\pi(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)\ \phi_{\mathcal{E}})\ \cup_\phi\ snd(\mathcal{R}^\pi(\![\{\!|Q|\!\}_\rho \uplus_\rho \rho]\!)\ \phi_{\mathcal{E}}\ )) =$

$$p, ((\bigcup_{P \in \rho'_P}^{\phi} snd(\mathcal{E}^\pi(\llbracket P \rrbracket)\ \rho''_P\ \phi_\mathcal{E}))\ \cup_\phi\ (\bigcup_{Q \in \rho'_Q}^{\phi} snd(\mathcal{E}^\pi(\llbracket Q \rrbracket)\ \rho''_Q\ \phi_\mathcal{E}))) \quad \text{(by } \mathcal{E}^\pi 5\text{)}$$

Where, $\rho'_P = \{\!|P|\!\}_\rho \uplus_\rho \rho$, $\rho'_Q = \{\!|Q|\!\}_\rho \uplus_\rho \rho$, $\rho''_P = \rho'_P \backslash \{\!|P|\!\}_\rho$ and $\rho''_Q = \rho'_Q \backslash \{\!|Q|\!\}_\rho$.

By the induction hypothesis for P:

$(\mathcal{E}^\pi(\llbracket P \rrbracket)\ \rho''\ \phi_\mathcal{E} = p', \phi'_\mathcal{E})\ \wedge\ (\mathcal{A}^\pi(\llbracket P \rrbracket)\ \rho''\ \phi_\mathcal{A} = \phi'_\mathcal{A})\ \wedge$

$(\exists y \in \mathcal{N} : \varphi_\mathcal{E}(\phi_\mathcal{E}, y) \in \phi_\mathcal{E}(x)\ \Rightarrow\ \exists t \in \phi_\mathcal{A}(\alpha_k(x)) : value\_of(\{t\}) = \{\alpha_k(y)\})$

$\Rightarrow \quad (\exists y \in \mathcal{N} : \varphi_\mathcal{E}(\phi'_\mathcal{E}, y) \in \phi'_\mathcal{E}(x)\ \Rightarrow\ \exists t \in \phi'_\mathcal{A}(\alpha_k(x)) : value\_of(\{t\}) = \{\alpha_k(y)\})$

By the safety of the $\cup_\phi$ relation (Lemma 1), we arrive at the following result:

$p, (\bigcup_{P \in \rho'}^{\phi} snd(\mathcal{E}^\pi(\llbracket P \rrbracket)\ \rho''\ \phi_\mathcal{E})) = p, \phi \quad \wedge \quad (\bigcup_{P \in \rho'}^{\phi} \mathcal{A}^\pi(\llbracket P \rrbracket)\ \rho''\ \phi_\mathcal{A}) = \phi' \quad \wedge$

$(\exists y \in \mathcal{N} : \varphi_\mathcal{E}(\phi_\mathcal{E}, y) \in \phi_\mathcal{E}(x)\ \Rightarrow\ \exists t \in \phi_\mathcal{A}(\alpha_k(x)) : value\_of(\{t\}) = \{\alpha_k(y)\})$

$\Rightarrow \quad (\exists y \in \mathcal{N} : \varphi_\mathcal{E}(\phi, y) \in \phi(x)\ \Rightarrow\ \exists t \in \phi'(\alpha_k(x)) : value\_of(\{t\}) = \{\alpha_k(y)\})$

Which satisfies the induction step and the safety requirement.

Case 5: $P \mid Q$

$$\mathcal{A}^\pi(\llbracket P \mid Q \rrbracket)\ \rho\ \phi_\mathcal{A} = \mathcal{R}^\pi(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \{\!|Q|\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi_\mathcal{A}\ =\ \bigcup_{P \in \rho'}^{\phi} \mathcal{A}^\pi(\llbracket P \rrbracket)\ \rho''\ \phi_\mathcal{A} \quad \text{(by } \mathcal{A}^\pi 6\text{)}$$

$$\mathcal{E}^\pi(\llbracket P \mid Q \rrbracket)\ \rho\ \phi_\mathcal{E} = \mathcal{R}^\pi(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \{\!|Q|\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi_\mathcal{E}\ =\ p, (\bigcup_{P \in \rho'}^{\phi} snd(\mathcal{E}^\pi(\llbracket P \rrbracket)\ \rho''\ \phi_\mathcal{E})) \quad \text{(by } \mathcal{E}^\pi 6\text{)}$$

Where, $\rho' = \{\!|P|\!\}_\rho \uplus_\rho \{\!|Q|\!\}_\rho \uplus_\rho \rho$ and $\rho'' = \rho' \backslash \{\!|P|\!\}_\rho$.

By the induction hypothesis for P:

$(\mathcal{E}^\pi(\llbracket P \rrbracket)\ \rho''\ \phi_\mathcal{E} = p', \phi'_\mathcal{E})\ \wedge\ (\mathcal{A}^\pi(\llbracket P \rrbracket)\ \rho''\ \phi_\mathcal{A} = \phi'_\mathcal{A})\ \wedge$

$(\exists y \in \mathcal{N} : \varphi_\mathcal{E}(\phi_\mathcal{E}, y) \in \phi_\mathcal{E}(x)\ \Rightarrow\ \exists t \in \phi_\mathcal{A}(\alpha_k(x)) : value\_of(\{t\}) = \{\alpha_k(y)\})$

$\Rightarrow \quad (\exists y \in \mathcal{N} : \varphi_\mathcal{E}(\phi'_\mathcal{E}, y) \in \phi'_\mathcal{E}(x)\ \Rightarrow\ \exists t \in \phi'_\mathcal{A}(\alpha_k(x)) : value\_of(\{t\}) = \{\alpha_k(y)\})$

By the safety of the $\cup_\phi$ relation (Lemma 1), we arrive at the following result:

$p, (\bigcup_{P \in \rho'}^{\phi} \mathcal{E}^\pi(\llbracket P \rrbracket)\ \rho''\ \phi_\mathcal{E}) = p, \phi \quad \wedge \quad (\bigcup_{P \in \rho'}^{\phi} \mathcal{A}^\pi(\llbracket P \rrbracket)\ \rho''\ \phi_\mathcal{A}) = \phi' \quad \wedge$

$(\exists y \in \mathcal{N} : \varphi_\mathcal{E}(\phi_\mathcal{E}, y) \in \phi_\mathcal{E}(x)\ \Rightarrow\ \exists t \in \phi_\mathcal{A}(\alpha_k(x)) : value\_of(\{t\}) = \{\alpha_k(y)\})$

$\Rightarrow \quad (\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi, y) \in \phi \ \Rightarrow \ \exists t \in \phi'(\alpha_k(x)) : \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$

Which satisfies the induction step and the safety requirement.

Case 6: $(\nu\, x)P$

$\mathcal{A}^{\pi}(\llbracket(\nu\, x)P\rrbracket)\ \rho\ \phi_{\mathcal{A}} = \mathcal{R}^{\pi}(\llbracket\{\!|P|\!\}_{\rho} \uplus_{\rho} \rho\rrbracket)\ \phi_{\mathcal{A}} \ = \ \bigcup_{P \in \rho'}^{\phi} \mathcal{A}^{\pi}(\llbracket P\rrbracket)\ \rho''\ \phi_{\mathcal{A}}$ (by $\mathcal{A}^{\pi}7$)

$\mathcal{E}^{\pi}(\llbracket(\nu\, x)P\rrbracket)\ \rho\ \phi_{\mathcal{E}} = p, \mathit{snd}(\mathcal{R}^{\pi}(\llbracket\{\!|P|\!\}_{\rho} \uplus_{\rho} \rho\rrbracket)\ \phi_{\mathcal{E}}) \ = \ p', \bigcup_{P \in \rho'}^{\phi} \mathit{snd}(\mathcal{E}^{\pi}(\llbracket P\rrbracket)\ \rho''\ \phi_{\mathcal{E}})$ (by $\mathcal{E}^{\pi}7$)

Where, $\rho' = \{\!|P|\!\}_{\rho} \uplus_{\rho} \rho$ and $\rho'' = \rho' \backslash \{\!|P|\!\}_{\rho}$

By the induction hypothesis for P:

$(\mathcal{E}^{\pi}(\llbracket P\rrbracket)\ \rho''\ \phi_{\mathcal{E}} = p', \phi'_{\mathcal{E}}) \ \wedge \ (\mathcal{A}^{\pi}(\llbracket P\rrbracket)\ \rho''\ \phi_{\mathcal{A}} = \phi'_{\mathcal{A}}) \wedge$

$(\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, y) \in \phi_{\mathcal{E}}(x) \ \Rightarrow \ \exists t \in \phi_{\mathcal{A}}(\alpha_k(x)) : \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$

$\Rightarrow \quad (\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, y) \in \phi'_{\mathcal{E}}(x) \ \Rightarrow \ \exists t \in \phi'_{\mathcal{A}}(\alpha_k(x)) : \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$

By the safety of the $\cup_{\phi}$ relation (Lemma 1), we arrive at the following result:

$p', (\bigcup_{P \in \rho'}^{\phi} \mathit{snd}(\mathcal{E}^{\pi}(\llbracket P\rrbracket)\ \rho''\ \phi_{\mathcal{E}})) = \phi \quad \wedge \quad (\bigcup_{P \in \rho'}^{\phi} \mathcal{A}^{\pi}(\llbracket P\rrbracket)\ \rho''\ \phi_{\mathcal{A}}) = \phi' \quad \wedge$

$(\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, y) \in \phi_{\mathcal{E}}(x) \ \Rightarrow \ \exists t \in \phi_{\mathcal{A}}(\alpha_k(x)) : \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$

$\Rightarrow \quad (\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi, y) \in \phi(x) \ \Rightarrow \ \exists t \in \phi'(\alpha_k(x)) : \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$

Which satisfies the induction step and the safety requirement.

Case 7: $!P$

$\mathcal{A}^{\pi}(\llbracket!P\rrbracket)\ \rho\ \phi_{\mathcal{A}} = \mathcal{F}^{\pi}_{\mathcal{A}}(-1)$

where,

$\mathcal{F}^{\pi}_{\mathcal{A}}(n) = \mathit{let}\ \phi_1 = \mathcal{A}^{\pi}(\llbracket\ \prod_{i=1}^{n} \mathit{ren}(P,i)\ \rrbracket)\ \rho\ \phi_{\mathcal{A}}\ \mathit{in}$

$\mathit{let}\ \phi_2 = \mathcal{A}^{\pi}(\llbracket\ \prod_{i=1}^{n+1} \mathit{ren}(P,i)\ \rrbracket)\ \rho\ \phi_{\mathcal{A}}\ \mathit{in}$

$\mathit{if}\ \phi_1 = \phi_2\ \mathit{then}\ \phi_1\ \mathit{else}\ \mathcal{F}^{\pi}_{\mathcal{A}}(n+1)$

and, $\forall x \in bn(P), y^t \in n(P) : \mathit{ren}(P,i) = (P[x_i/x])[y^{t_i}/y^t]$ (by $\mathcal{A}^{\pi}8$)

$\mathcal{E}^{\pi}(\llbracket!P\rrbracket)\ \rho\ \phi_{\mathcal{E}} = \mathcal{F}^{\pi}_{\mathcal{E}}(-1)$

where,

$\mathcal{F}^{\pi}_{\mathcal{E}}(n) = \mathit{let}\ v_1 = \mathcal{E}^{\pi}(\![\ \prod_{i=1}^{n} P[bn_i(P)/bn(P)]\ ]\!)\ \rho\ \phi_{\mathcal{E}}\ \mathit{in}$
$\mathit{let}\ v_2 = \mathcal{E}^{\pi}(\![\ \prod_{i=1}^{n+1} P[bn_i(P)/bn(P)]\ ]\!)\ \rho\ \phi_{\mathcal{E}}\ \mathit{in}$
$\mathit{if}\ v_1 = v_2\ \mathit{then}\ v_1\ \mathit{else}\ \mathcal{F}^{\pi}_{\mathcal{E}}(n+1)$
and, $bn_i(P) = \{x_i \mid x \in bn(P)\}$ (by $\mathcal{E}^{\pi}8$)

The proof proceeds by mathematical induction, where we demonstrate that $\mathcal{F}^{\pi}_{\mathcal{A}}(-1)$ is safe with respect to $\mathcal{F}^{\pi}_{\mathcal{E}}(-1)$ (base case) and that if $\mathcal{F}^{\pi}_{\mathcal{A}}(n)$ is safe with respect to $\mathcal{F}^{\pi}_{\mathcal{E}}(n)$ then $\mathcal{F}^{\pi}_{\mathcal{A}}(n+1)$ is safe with respect to $\mathcal{F}^{\pi}_{\mathcal{E}}(n+1)$ (inductive step).

The base case: $n = -1$

$\mathcal{F}^{\pi}_{\mathcal{A}}(-1) = \mathit{let}\ \phi_1 = \bot_{D^\sharp}\ \mathit{in}$
$\mathit{let}\ \phi_2 = \mathcal{A}^{\pi}(\![\ \mathbf{0}\ ]\!)\ \rho\ \phi_{\mathcal{A}}\ \mathit{in}$
$\mathit{if}\ \phi_1 = \phi_2\ \mathit{then}\ \phi_1\ \mathit{else}\ \mathcal{F}^{\pi}_{\mathcal{A}}(0)$

$\mathcal{F}^{\pi}_{\mathcal{E}}(-1) = \mathit{let}\ v_1 = (\bot_{Pi_\bot}, \bot_D)\ \mathit{in}$
$\mathit{let}\ v_2 = \mathcal{E}^{\pi}(\![\ \mathbf{0}\ ]\!)\ \rho\ \phi_{\mathcal{E}}\ \mathit{in}$
$\mathit{if}\ v_1 = v_2\ \mathit{then}\ v_1\ \mathit{else}\ \mathcal{F}^{\pi}_{\mathcal{E}}(0)$

From the above two calculations, we have that $\bot_{D^\sharp} = \phi_{\mathcal{A}0}$ is a safe abstraction of $\bot_D = \phi_{\mathcal{E}0}$ since $\forall x \in dom(\phi_{\mathcal{E}0}) : \phi_{\mathcal{E}0}(x) = \phi_{\mathcal{A}0}(x) = \{\}$. Also, since we have proved the case for $P = \mathbf{0}$ (The second computation of $\mathcal{F}^{\pi}$) earlier, then we have that:
$(\mathcal{E}^{\pi}(\![P]\!)\ \rho\ \phi_{\mathcal{E}} = p', \phi'_{\mathcal{E}})\ \wedge\ (\mathcal{A}^{\pi}(\![P]\!)\ \rho\ \phi_{\mathcal{A}} = \phi'_{\mathcal{A}})\ \wedge$
$(\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, y) \in \phi_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi_{\mathcal{A}}(\alpha_k(x)) : \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$

$\Rightarrow\quad (\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, y) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi'_{\mathcal{A}}(\alpha_k(x)) : \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$

The induction step:

We start by assuming the inductive hypothesis:
$(\mathcal{F}^{\pi}_{\mathcal{E}}(n) = p', \phi'_{\mathcal{E}})\ \wedge\ (\mathcal{F}^{\pi}_{\mathcal{A}}(n) = \phi'_{\mathcal{A}})\ \wedge$
$(\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, y) \in \phi_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi_{\mathcal{A}}(\alpha_k(x)) : \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$

$\Rightarrow\quad (\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, y) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi'_{\mathcal{A}}(\alpha_k(x)) : \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$

Now, we prove that the same safety relation holds for $\mathcal{F}^{\pi}_{\mathcal{A}}(n+1)$ with respect to $\mathcal{F}^{\pi}_{\mathcal{E}}(n+1)$, as follows:

$\mathcal{F}^{\pi}_{\mathcal{A}}(n+1) = let\ \phi_1 = \mathcal{A}^{\pi}(\![\ \prod_{i=1}^{n+1} ren(P,i)\ ]\!)\ \rho\ \phi_{\mathcal{A}}\ in$
$let\ \phi_2 = \mathcal{A}^{\pi}(\![\ \prod_{i=1}^{n+2} ren(P,i)\ ]\!)\ \rho\ \phi_{\mathcal{A}}\ in$
$if\ \phi_1 = \phi_2\ then\ \phi_1\ else\ \mathcal{F}^{\pi}_{\mathcal{A}}(n+2)$

$\mathcal{F}^{\pi}_{\mathcal{E}}(n+1) = let\ v_1 = \mathcal{E}^{\pi}(\![\ \prod_{i=1}^{n+1} P[bn_i(P)/bn(P)]\ ]\!)\ \rho\ \phi_{\mathcal{E}}\ in$
$let\ v_2 = \mathcal{E}^{\pi}(\![\ \prod_{i=1}^{n+2} P[bn_i(P)/bn(P)]\ ]\!)\ \rho\ \phi_{\mathcal{E}}\ in$
$if\ v_1 = v_2\ then\ v_1\ else\ \mathcal{F}^{\pi}_{\mathcal{E}}(n+2)$

From the induction hypothesis, we know that:
$(\mathcal{E}^{\pi}(\![\ \prod_{i=1}^{n+1} P[bn_i(P)/bn(P)]\ ]\!)\ \rho\ \phi_{\mathcal{E}} = p', \phi'_{\mathcal{E}})\ \wedge\ (\mathcal{A}^{\pi}(\![\ \prod_{i=1}^{n+1} ren(P,i)\ ]\!)\ \rho\ \phi_{\mathcal{A}} = \phi'_{\mathcal{A}})\ \wedge$
$(\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, y) \in \phi_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi_{\mathcal{A}}(\alpha_k(x)) : value\_of(\{t\}) = \{\alpha_k(y)\})$

$\Rightarrow\quad (\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, y) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi'_{\mathcal{A}}(\alpha_k(x)) : value\_of(\{t\}) = \{\alpha_k(y)\})$

The remaining case involves $\mathcal{A}^{\pi}(\![\ \prod_{i=1}^{n+2} ren(P,i)\ ]\!)\ \rho\ \phi_{\mathcal{A}}$ and $\mathcal{E}^{\pi}(\![\ \prod_{i=1}^{n+2} P[bn_i(P)/bn(P)]\ ]\!)\ \rho\ \phi_{\mathcal{E}}$. We can rewrite these as:
$\mathcal{A}^{\pi}(\![\ \prod_{i=1}^{n+2} ren(P,i)\ ]\!)\ \rho\ \phi_{\mathcal{A}} = \mathcal{A}^{\pi}(\![\ \prod_{i=1}^{n+1} ren(P,i)\ \mid\ ren(P,n+2)\ ]\!)\ \rho\ \phi_{\mathcal{A}}$

$\mathcal{E}^{\pi}(\![\ \prod_{i=1}^{n+2} P[bn_i(P)/bn(P)]\ ]\!)\ \rho\ \phi_{\mathcal{E}} = \mathcal{E}^{\pi}(\![\ \prod_{i=1}^{n+1} P[bn_i(P)/bn(P)]\ \mid\ P[bn_{n+2}(P)/bn(P)]\ ]\!)\ \rho\ \phi_{\mathcal{E}}$

From the induction hypothesis, we have that:
$(\mathcal{E}^{\pi}(\![\ \prod_{i=1}^{n+1} P[bn_i(P)/bn(P)]\ ]\!)\ \rho\ \phi_{\mathcal{E}} = p', \phi'_{\mathcal{E}})\ \wedge\ (\mathcal{A}^{\pi}(\![\ \prod_{i=1}^{n+1} ren(P,i)\ ]\!)\ \rho\ \phi_{\mathcal{A}} = \phi'_{\mathcal{A}})\ \wedge$
$(\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, y) \in \phi_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi_{\mathcal{A}}(\alpha_k(x)) : value\_of(\{t\}) = \{\alpha_k(y)\})$

$\Rightarrow\quad (\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, y) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi'_{\mathcal{A}}(\alpha_k(x)) : value\_of(\{t\}) = \{\alpha_k(y)\})$

And from the antecedent:
$(\mathcal{E}^{\pi}(\![\ P[bn_{n+2}(P)/bn(P)]\ ]\!)\ \rho\ \phi_{\mathcal{E}} = p', \phi'_{\mathcal{E}})\ \wedge\ (\mathcal{A}^{\pi}(\![\ ren(P,n+2)\ ]\!)\ \rho\ \phi_{\mathcal{A}} = \phi'_{\mathcal{A}})\ \wedge$
$(\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, y) \in \phi_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi_{\mathcal{A}}(\alpha_k(x)) : value\_of(\{t\}) = \{\alpha_k(y)\})$

$\Rightarrow\quad (\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, y) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi'_{\mathcal{A}}(\alpha_k(x)) : value\_of(\{t\}) = \{\alpha_k(y)\})$

We have already proven the case for the parallel composition of two processes, $\mathcal{A}^{\pi}(\![\ P \mid Q\ ]\!)\ \rho\ \phi_{\mathcal{A}}$ and $\mathcal{E}^{\pi}(\![\ P \mid Q\ ]\!)\ \rho\ \phi_{\mathcal{E}}$ (case 5). Hence, we can conclude that:

$(\mathcal{F}^{\pi}_{\mathcal{E}}(n+1) = p', \phi'_{\mathcal{E}}) \ \wedge \ (\mathcal{F}^{\pi}_{\mathcal{A}}(n+1) = \phi'_{\mathcal{A}}) \wedge$
$(\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, y) \in \phi_{\mathcal{E}}(x) \ \Rightarrow \ \exists t \in \phi_{\mathcal{A}}(\alpha_k(x)) : \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$

$\Rightarrow \quad (\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, y) \in \phi'_{\mathcal{E}}(x) \ \Rightarrow \ \exists t \in \phi'_{\mathcal{A}}(\alpha_k(x)) : \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$

From both the base case and the induction step, we can conclude that:
$(\mathcal{E}^{\pi}(\llbracket \ !P \ \rrbracket \ \rho \ \phi_{\mathcal{E}} = p', \phi'_{\mathcal{E}}) \ \wedge \ (\mathcal{A}^{\pi}(\llbracket \ !P \ \rrbracket \ \rho \ \phi_{\mathcal{A}} = \phi'_{\mathcal{A}}) \wedge$
$(\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, y) \in \phi_{\mathcal{E}}(x) \ \Rightarrow \ \exists t \in \phi_{\mathcal{A}}(\alpha_k(x)) : \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$

$\Rightarrow \quad (\exists y \in \mathcal{N} : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, y) \in \phi'_{\mathcal{E}}(x) \ \Rightarrow \ \exists t \in \phi'_{\mathcal{A}}(\alpha_k(x)) : \mathit{value\_of}(\{t\}) = \{\alpha_k(y)\})$

□

A.3 Safety of the $\cup_\phi$ operation in the spi calculus (Lemma 2)

$\forall i \in \{1 \ldots n\}, n \in \mathbb{N}, \phi_i \in D_\perp, \phi'_i \in D^\sharp_\perp :$
$(\phi = \bigcup_{i=1\ldots n}^{\phi} \phi_i) \ \wedge \ (\phi' = \bigcup_{i=1\ldots n}^{\phi} \phi'_i) \ \wedge$
$(\exists M \in \mathit{Term} : \varphi_{\mathcal{E}}(\phi_i, M) \in \phi_i(x) \ \Rightarrow \ \exists t \in \phi'_i(\alpha_{k,k'}(x)) : \mathit{value\_of}(\{t\}) = \{M'\} \wedge$
$\mathit{untag}(M') = (\forall x \in \mathit{bnv}(M) : M[\alpha_{k,k'}(x)/x]))$

$\Rightarrow \quad (\exists M \in \mathit{Term} : \varphi_{\mathcal{E}}(\phi, M) \in \phi(x) \ \Rightarrow \ \exists t \in \phi'(\alpha_{k,k'}(x)) : \mathit{value\_of}(\{t\}) = \{M'\} \wedge$
$\mathit{untag}(M') = (\forall x \in \mathit{bnv}(M) : M[\alpha_{k,k'}(x)/x]))$

The proof proceeds by mathematical induction on sets of environments.

The base case: n=0

$(\phi = \bigcup_{i=1\ldots 0}^{\phi} \phi_i = \phi_0) \ \wedge \ (\phi' = \bigcup_{i=1\ldots 0}^{\phi} \phi'_i = \phi_0)$

$\Rightarrow \quad \exists M \in \mathit{Term} : \varphi_{\mathcal{E}}(\phi_0, M) \in \phi_0(x) \ \Rightarrow \ \exists t \in \phi'_0(\alpha_{k,k'}(x)) : \mathit{value\_of}(\{t\}) = \{M'\} \wedge$
$\mathit{untag}(M') = (\forall x \in \mathit{bnv}(M) : M[\alpha_{k,k'}(x)/x])$

Since, $\forall x \in \mathcal{N} : \phi_0(x) = \phi'_0(\alpha_k(x)) = \emptyset$.

The induction step. First, assume the hypothesis is true for a union of n environments:
$(\phi = \bigcup_{i=1\ldots n}^{\phi} \phi_i) \ \wedge \ (\phi' = \bigcup_{i=1\ldots n}^{\phi} \phi'_i) \ \wedge$
$(\exists M \in \mathit{Term} : \varphi_{\mathcal{E}}(\phi_i, M) \in \phi_i(x) \ \Rightarrow \ \exists t \in \phi'_i(\alpha_{k,k'}(x)) : \mathit{value\_of}(\{t\}) = \{M'\} \wedge$
$\mathit{untag}(M') = (\forall x \in \mathit{bnv}(M) : M[\alpha_{k,k'}(x)/x]))$

$\Rightarrow \quad (\exists M \in \mathit{Term} : \varphi_{\mathcal{E}}(\phi, M) \in \phi(x) \ \Rightarrow \ \exists t \in \phi'(\alpha_{k,k'}(x)) : \mathit{value\_of}(\{t\}) = \{M'\} \wedge$
$\mathit{untag}(M') = (\forall x \in \mathit{bnv}(M) : M[\alpha_{k,k'}(x)/x]))$

Proving the induction step demands that the above statement is true for a union of $n+1$ environments. Assuming that, $\phi = \bigcup_{i=1\ldots n}^{\phi} \phi_i$ and $\phi' = \bigcup_{i=1\ldots n}^{\phi} \phi'_i$, as above, we can show that:
$\bigcup_{i=1\ldots n+1}^{\phi} \phi_i = \phi \cup_\phi \phi_{n+1}$
$\bigcup_{i=1\ldots n+1}^{\phi} \phi'_i = \phi' \cup_\phi \phi'_{n+1}$

From the definition of $\cup_\phi$ in Section 4.2.1, we have:

$(\phi \cup_\phi \phi_{n+1})(x) = \phi(x) \cup \phi_{n+1}(x)$
$(\phi' \cup_\phi \phi'_{n+1})(x) = \phi'(x) \cup \phi'_{n+1}(x)$

We obtain $\forall y \in \phi(x), y' \in \phi_{n+1}(x) \ \Rightarrow \ y, y' \in (\phi(x) \cup \phi_{n+1}(x))$
and, $\forall t \in \phi'(\alpha_k(x)), t' \in \phi'_{n+1}(\alpha_k(x)) \ \Rightarrow \ t, t' \in (\phi'(\alpha_k(x)) \cup \phi'_{n+1}(\alpha_k(x)))$

By the inductive hypothesis, we have that:
$(\exists M \in \mathit{Term} : \varphi_{\mathcal{E}}(\phi, M) \in \phi(x) \ \Rightarrow \ \exists t \in \phi'(\alpha_{k,k'}(x)) : \mathit{value\_of}(\{t\}) = \{M'\} \wedge$
$\mathit{untag}(M') = (\forall x \in \mathit{bnv}(M) : M[\alpha_{k,k'}(x)/x]))$

and with the assumption that the new added environments are also safe:
$(\exists M \in \mathit{Term} : \varphi_{\mathcal{E}}(\phi_{n+1}, M) \in \phi_{n+1}(x) \ \Rightarrow \ \exists t \in \phi'_{n+1}(\alpha_{k,k'}(x)) : \mathit{value\_of}(\{t\}) = \{M'\} \wedge$
$\mathit{untag}(M') = (\forall x \in \mathit{bnv}(M) : M[\alpha_{k,k'}(x)/x]))$

Then by the properties of set union, it is possible to arrive at the following result:
$(\exists M \in \mathit{Term} : \varphi_{\mathcal{E}}((\phi(x) \ \cup \ \phi_{n+1}(x)), M) \in (\phi(x) \ \cup \ \phi_{n+1}(x))$

$\Rightarrow \quad \exists t \in (\phi'(\alpha_{k,k'}(x)) \ \cup \ \phi'_{n+1}(\alpha_{k,k'}(x))) : \mathit{value\_of}(\{t\}) = \{M'\} \wedge$
$\mathit{untag}(M') = (\forall x \in \mathit{bnv}(M) : M[\alpha_{k,k'}(x)/x]))$

□

A.4 Safety of the abstract semantics for the spi calculus (Theorem 8)

$\forall P, \rho, \phi_{\mathcal{E}}, \phi_{\mathcal{A}}$:
$(\mathcal{E}^{spi}(\llbracket P \rrbracket)\ \rho\ \phi_{\mathcal{E}} = (p, \phi'_{\mathcal{E}}))\ \wedge\ (\mathcal{A}^{spi}(\llbracket P \rrbracket)\ \rho\ \phi_{\mathcal{A}} = \phi'_{\mathcal{A}})\ \wedge$
$(\exists M \in Term : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) \in \phi_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\}\ \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

$\Rightarrow\quad (\exists M \in Term : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, M) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi'_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\}\ \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

The proof is by structural induction on P (in what follows, we have that $fst(x_1, x_2) = x_1$ and $snd(x_1, x_2) = x_2$).

The base case: 0

$\mathcal{A}^{spi}(\llbracket \mathbf{0} \rrbracket)\ \rho\ \phi_{\mathcal{A}} = \phi_{\mathcal{A}}$ by $(\mathcal{A}^{spi}1)$

$\mathcal{E}^{spi}(\llbracket \mathbf{0} \rrbracket)\ \rho\ \phi_{\mathcal{E}} = (\emptyset, \phi_{\mathcal{E}})$ by $(\mathcal{E}^{spi}1)$

Here we have from the antecedent:
$\mathcal{A}^{spi}(\llbracket P \rrbracket)\ \rho_0\ \phi_{\mathcal{A}} = \phi'_{\mathcal{A}}$
$\mathcal{E}^{spi}(\llbracket P \rrbracket)\ \rho_0\ \phi_{\mathcal{E}} = (p, \phi'_{\mathcal{E}})$

Since $\phi'_{\mathcal{E}} = \phi_{\mathcal{E}}$ and $\phi'_{\mathcal{A}} = \phi_{\mathcal{A}}$, this satisfies the safety requirement as follows:
$(\exists M \in Term : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) \in \phi_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\}\ \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

$\Rightarrow\quad (\exists M \in Term : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, M) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi'_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\}\ \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

The induction step.

Case 1: $x(y).P$

$\mathcal{A}^{spi}(\llbracket M(x).P \rrbracket)\ \rho\ \phi_{\mathcal{A}}\ =\ \phi_{\mathcal{A}}$ by $(\mathcal{A}^{spi}2)$

$\mathcal{E}^{spi}(\llbracket M(x).P \rrbracket)\ \rho\ \phi_{\mathcal{E}}\ =\ (\{\!|in(\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M), \lambda x.p')|\!\}, \phi_{\mathcal{E}})$
where, $(p', \phi'_{\mathcal{E}}) = \mathcal{R}^{spi}(\llbracket \{\!| P |\!\}_{\rho} \rrbracket)\ \phi_{\mathcal{E}}$ and, $\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) \in N$ by $(\mathcal{E}^{spi}2)$

From the antecedent, we have that:
$\mathcal{A}^{spi}(\llbracket P \rrbracket)\ \rho_0\ \phi_{\mathcal{A}} = \phi'_{\mathcal{A}}$
$\mathcal{E}^{spi}(\llbracket P \rrbracket)\ \rho_0\ \phi_{\mathcal{E}} = (p, \phi'_{\mathcal{E}})$

Since $\phi'_{\mathcal{E}} = \phi_{\mathcal{E}}$ and $\phi'_{\mathcal{A}} = \phi_{\mathcal{A}}$, this satisfies the induction step and the safety requirement as follows:

$(\exists M \in Term : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) \in \phi_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\} \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

$\Rightarrow\quad (\exists M \in Term : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, M) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi'_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\} \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

Case 2: $\overline{M}\langle N \rangle.P$
$\mathcal{A}^{spi}(\llbracket \overline{M}\langle L^t \rangle.P \rrbracket)\ \rho\ \phi_{\mathcal{A}}\ =\ (\bigcup_{\substack{\phi \\ M'(z).P' \in \rho}} \phi'_{\mathcal{A}})\ \cup_{\phi}\ \phi_{\mathcal{A}}$
if, $untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, M)) \cap untag(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, M')) \cap \mathcal{N} \neq \{\}$
where, $\phi'_{\mathcal{A}} = \mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_{\rho} \uplus_{\rho} \rho[P'/M'(z).P'] \rrbracket)\ \phi''_{\mathcal{A}}$
and, $\phi''_{\mathcal{A}} = \phi_{\mathcal{A}}[\alpha_{k,k'}(z) \mapsto\ \phi_{\mathcal{A}}(\alpha_{k,k'}(z)) \cup \{\alpha_{k,k'}(t)\}]$ by $(\mathcal{A}^{spi}3)$

$\mathcal{E}^{spi}(\llbracket \overline{M}\langle L \rangle.P \rrbracket)\ \rho\ \phi_{\mathcal{E}}\ =$
$(\biguplus_{M'(z).P' \in \rho} \{\!|tau(p')|\!\}\ \uplus\ \{\!|out(\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M), \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, L), p'')|\!\}, \bigcup_{\substack{\phi \\ M'(z).P' \in \rho}} \phi'_{\mathcal{E}}\ \cup_{\phi}\ \phi_{\mathcal{E}})$
if, $\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) = \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M') \in N$
where, $(p', \phi'_{\mathcal{E}}) = \mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_{\rho} \uplus_{\rho} \rho[P'/M'(z).P'] \rrbracket)\ \phi_{\mathcal{E}}[z \mapsto \{\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, L)\}]$
and, $(p'', \phi''_{\mathcal{E}}) = \mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_{\rho} \rrbracket)\ \phi_{\mathcal{E}}$ by $(\mathcal{E}^{spi}3)$

We only consider the safety of the first $\phi'_{\mathcal{A}}$ environment (the safety of the second $\phi_{\mathcal{A}}$ environment follows from the base case). Extending $\phi'_{\mathcal{A}}$ and $(p', \phi'_{\mathcal{E}})$ in the above rules yields:
$\phi'_{\mathcal{A}} = \bigcup_{\substack{\phi \\ P \in \rho'}} \mathcal{A}^{spi}(\llbracket P \rrbracket)\ \rho[P' \backslash M'(z).P']\ \phi''_{\mathcal{A}}$
$(p', \phi'_{\mathcal{E}}) = (\biguplus_{P \in \rho} p''', \bigcup_{\substack{\phi \\ P \in \rho}} \phi'''_{\mathcal{E}})$
where, $(p''', \phi'''_{\mathcal{E}}) = \mathcal{E}^{spi}(\llbracket P \rrbracket)\ \rho[P' \backslash M'(z).P']\ \phi_{\mathcal{E}}[z \mapsto \{\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, L)\}]$

From the antecedent, we have that:
$(\exists M \in Term : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) \in \phi_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\} \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

From the above values of $\phi'''_{\mathcal{E}}$ and $\phi''_{\mathcal{A}}$ we have that:

$(\exists M \in Term : \varphi_{\mathcal{E}}(\phi'''_{\mathcal{E}}, L) \in \phi'''_{\mathcal{E}}(z) \ \Rightarrow \ \exists t \in \phi''_{\mathcal{A}}(\alpha_{k,k'}(z)) : value\_of(\{t\}) = \{L'\} \wedge$
$untag(L') = (\forall y \in bnv(L) : L[\alpha_{k,k'}(y)/y]))$

$\Rightarrow \quad (\exists M \in Term : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, M) \in \phi'_{\mathcal{E}}(x) \ \Rightarrow \ \exists t \in \phi'_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\} \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$ (From the antecedent)

By the induction hypothesis for P:
$(\mathcal{E}^{spi}(\llbracket P \rrbracket)\ \rho[P'\backslash M'(z).P']\ \phi_{\mathcal{E}}[z \mapsto \{\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, L)\}] = (p''', \phi'''_{\mathcal{E}})) \wedge$
$(\mathcal{A}^{spi}(\llbracket P \rrbracket)\ \rho[P'\backslash M'(z).P']\ \phi''_{\mathcal{A}} = \phi'''_{\mathcal{A}}) \wedge$
$(\exists M \in Term : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}[z \mapsto \{\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, L)\}], L) \in \phi_{\mathcal{E}}[z \mapsto \{\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, L)\}](z) \ \Rightarrow$
$\exists t \in \phi''_{\mathcal{A}}(\alpha_{k,k'}(z)) : value\_of(\{t\}) = \{L'\} \ \wedge untag(L') = (\forall y \in bnv(L) : L[\alpha_{k,k'}(y)/y]))$

By the safety of the $\cup_\phi$ relation (Lemma 2), we arrive at the following result (where p''', $\phi'''_{\mathcal{E}}$ and $\phi'''_{\mathcal{A}}$ are given in the above induction hypothesis result for P):
$(\biguplus_{M'(z).P' \in \rho} \{\!|tau(\biguplus_{P \in \rho} p''')|\!\} \ , \ \bigcup_{\phi}\bigcup_{\phi} \phi'''_{\mathcal{E}}) = p, \phi \ \wedge \ (\bigcup_{\phi}\bigcup_{\phi} \phi'''_{\mathcal{A}}) = \phi' \ \wedge$
$(\exists L \in Term : \varphi_{\mathcal{E}}(\phi'''_{\mathcal{E}}, L) \in \phi'''_{\mathcal{E}}(z) \ \Rightarrow \ \exists t \in \phi'''_{\mathcal{A}}(\alpha_{k,k'}(z)) : value\_of(\{t\}) = \{L'\} \wedge$
$untag(L') = (\forall y \in bnv(L) : L[\alpha_{k,k'}(y)/y]))$

$\Rightarrow \quad (\exists L \in Term : \varphi_{\mathcal{E}}(\phi, L) \in \phi(z) \ \Rightarrow \ \exists t \in \phi'(\alpha_{k,k'}(z)) : value\_of(\{t\}) = \{L'\} \wedge$
$untag(L') = (\forall y \in bnv(L) : L[\alpha_{k,k'}(y)/y]))$

Which satisfies the induction step and the safety requirement.

Case 3: $(\nu\, a)P$

$\mathcal{A}^{spi}(\llbracket(\nu a)P\rrbracket)\ \rho\ \phi_{\mathcal{A}} \ = \ \mathcal{R}^{spi}(\llbracket\{\!|P|\!\}_\rho \uplus_\rho \rho\rrbracket)\ \phi_{\mathcal{A}} = \bigcup_{P \in \rho'}{}_{\phi}\ \mathcal{A}^{spi}(\llbracket P\rrbracket)\ \rho\ \phi_{\mathcal{A}}$ (by $\mathcal{A}^{spi}4$)

$\mathcal{E}^{spi}(\llbracket(\nu a)P\rrbracket)\ \rho\ \phi_{\mathcal{E}} \ = \ (new(\lambda a.p'), \phi'_{\mathcal{E}})$

where, $(p', \phi'_{\mathcal{E}}) = \mathcal{R}^{spi}(\llbracket\{\!|P|\!\}_\rho \uplus_\rho \rho\rrbracket)\ \phi_{\mathcal{E}} = (\biguplus_{P \in \rho} p'', \bigcup_{P \in \rho}{}_{\phi}\ \phi''_{\mathcal{E}})$

and, $(p'', \phi''_{\mathcal{E}}) = \mathcal{E}^{spi}(\llbracket P\rrbracket)\ \rho\ \phi_{\mathcal{E}}$ (by $\mathcal{E}^{spi}4$)

By the induction hypothesis for P:
$(\mathcal{E}^{spi}(\llbracket P\rrbracket)\ \rho\ \phi_{\mathcal{E}} = (p, \phi'_{\mathcal{E}})) \ \wedge \ (\mathcal{A}^{spi}(\llbracket P\rrbracket)\ \rho\ \phi_{\mathcal{A}} = \phi'_{\mathcal{A}}) \wedge$
$(\exists M \in Term : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) \in \phi_{\mathcal{E}}(x) \ \Rightarrow \ \exists t \in \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\} \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

$\Rightarrow \quad (\exists M \in Term : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, M) \in \phi'_{\mathcal{E}}(x) \ \Rightarrow \ \exists t \in \phi'_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\} \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

By the safety of the $\cup_\phi$ relation (Lemma 2), we arrive at the following result:

$(new(\lambda a.\ \biguplus_{P\in\rho} fst(\mathcal{E}^{spi}(\llbracket P\rrbracket\ \rho\ \phi_\mathcal{E}))\ ,\ \bigcup_{\phi\ P\in\rho} snd(\mathcal{E}^{spi}(\llbracket P\rrbracket\ \rho\ \phi_\mathcal{E})) = (p,\phi)\ \wedge$

$(\bigcup_{\phi\ P\in\rho'} \mathcal{A}^{spi}(\llbracket P\rrbracket\ \rho\ \phi_\mathcal{A} = \phi')\ \wedge$

$(\exists M \in Term : \varphi_\mathcal{E}(\phi_\mathcal{E}, M) \in \phi_\mathcal{E}(x)\ \Rightarrow\ \exists t \in \phi_\mathcal{A}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\}\ \wedge$

$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

$\Rightarrow\quad (\exists M \in Term : \varphi_\mathcal{E}(\phi, M) \in \phi(x)\ \Rightarrow\ \exists t \in \phi'(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\}\ \wedge$

$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

Which satisfies the induction step and the safety requirement.

Case 4: $P \mid Q$

$\mathcal{A}^{spi}(\llbracket P \mid Q\rrbracket\ \rho\ \phi_\mathcal{A}\ =\ \mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \{\!|Q|\!\}_\rho \uplus_\rho \rho\rrbracket\ \phi_\mathcal{A} = \bigcup_{\phi\ P\in\rho'} \mathcal{A}^{spi}(\llbracket P\rrbracket\ \rho''\ \phi_\mathcal{A}$ (by $\mathcal{A}^{spi}5$)

$\mathcal{E}^{spi}(\llbracket P \mid Q\rrbracket\ \rho\ \phi_\mathcal{E}\ =\ \mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \{\!|Q|\!\}_\rho \uplus_\rho \rho\rrbracket\ \phi_\mathcal{E} = (\biguplus_{P\in\rho'} p', \bigcup_{\phi\ P\in\rho'} \phi'_\mathcal{E})$

where, $(p', \phi'_\mathcal{E}) = \mathcal{E}^{spi}(\llbracket P\rrbracket\ \rho''\ \phi_\mathcal{E}$

and, $\rho'' = \rho' \backslash \{\!|P|\!\}_\rho$ and $\rho' = \{\!|P|\!\}_\rho \uplus_\rho \{\!|Q|\!\}_\rho \uplus_\rho \rho$ (by $\mathcal{E}^{spi}5$)

By the induction hypothesis for P:

$(\mathcal{E}^{spi}(\llbracket P\rrbracket\ \rho''\ \phi_\mathcal{E} = (p', \phi'_\mathcal{E}))\ \wedge\ (\mathcal{A}^{spi}(\llbracket P\rrbracket\ \rho''\ \phi_\mathcal{A} = \phi'_\mathcal{A})\ \wedge$

$(\exists M \in Term : \varphi_\mathcal{E}(\phi_\mathcal{E}, M) \in \phi_\mathcal{E}(x)\ \Rightarrow\ \exists t \in \phi_\mathcal{A}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\}\ \wedge$

$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

$\Rightarrow\quad (\exists M \in Term : \varphi_\mathcal{E}(\phi'_\mathcal{E}, M) \in \phi'_\mathcal{E}(x)\ \Rightarrow\ \exists t \in \phi'_\mathcal{A}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\}\ \wedge$

$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

By the safety of the $\cup_\phi$ relation (Lemma 2), we arrive at the following result:

$(\biguplus_{P\in\rho'} fst(\mathcal{E}^{spi}(\llbracket P\rrbracket\ \rho''\ \phi_\mathcal{E}), \bigcup_{\phi\ P\in\rho'} snd(\mathcal{E}^{spi}(\llbracket P\rrbracket\ \rho''\ \phi_\mathcal{E})) = (p,\phi)\ \wedge\ (\bigcup_{\phi\ P\in\rho'} \mathcal{A}^{spi}(\llbracket P\rrbracket\ \rho''\ \phi_\mathcal{A}) = \phi'\ \wedge$

$(\exists M \in Term : \varphi_\mathcal{E}(\phi_\mathcal{E}, M) \in \phi_\mathcal{E}(x)\ \Rightarrow\ \exists t \in \phi_\mathcal{A}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\}\ \wedge$

$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

$\Rightarrow\quad (\exists M \in Term : \varphi_\mathcal{E}(\phi, M) \in \phi(x)\ \Rightarrow\ \exists t \in \phi'(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\}\ \wedge$

$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

Which satisfies the induction step and the safety requirement.

Case 5: $!P$

$\mathcal{A}^{spi}(\![!P]\!)\ \rho\ \phi_{\mathcal{A}}\ =\ \mathcal{F}^{spi}_{\mathcal{A}}(-1)$

where, $\mathcal{F}^{spi}_{\mathcal{A}}(n) = let\ \phi_1 = \mathcal{A}^{spi}(\![\ \prod_{i=1}^{n} ren(P,i)\ ]\!)\ \rho\ \phi_{\mathcal{A}}\ in$

$let\ \phi_2 = \mathcal{A}^{spi}(\![\ \prod_{i=1}^{n+1} ren(P,i)\ ]\!)\ \rho\ \phi_{\mathcal{A}}\ in$

$if\ \phi_1 = \phi_2\ then\ \phi_1\ else\ \mathcal{F}^{spi}(n+1)$

and, $\forall x \in bnv(P), t \in tags\_of(P) : ren(P,i) = (P[x_i/x])[t_i/t]$ (by $\mathcal{A}^{spi}6$)

$\mathcal{E}^{spi}(\![!P]\!)\ \rho\ \phi_{\mathcal{E}}\ =\ \mathcal{F}^{spi}(-1)_{\mathcal{E}}$

where, $\mathcal{F}^{spi}_{\mathcal{E}}(n) = let\ v_1 = \mathcal{E}^{spi}(\![\ \prod_{i=1}^{n} P[bnv_i(P)/bnv(P)]\ ]\!)\ \rho\ \phi_{\mathcal{E}}\ in$

$let\ v_2 = \mathcal{E}^{spi}(\![\ \prod_{i=1}^{n+1} P[bnv_i(P)/bnv(P)]\ ]\!)\ \rho\ \phi_{\mathcal{E}}\ in$

$if\ v_1 = v_2\ then\ v_1\ else\ \mathcal{F}^{spi}(n+1)$ (by $\mathcal{E}^{spi}6$)

The proof proceeds by mathematical induction, where we demonstrate that $\mathcal{F}^{spi}_{\mathcal{A}}(-1)$ is safe with respect to $\mathcal{F}^{spi}_{\mathcal{E}}(-1)$ (base case) and that if $\mathcal{F}^{spi}_{\mathcal{A}}(n)$ is safe with respect to $\mathcal{F}^{spi}_{\mathcal{E}}(n)$ then $\mathcal{F}^{spi}_{\mathcal{A}}(n+1)$ is safe with respect to $\mathcal{F}^{spi}_{\mathcal{E}}(n+1)$ (inductive step).

The base case: $n = -1$

$\mathcal{F}^{spi}_{\mathcal{A}}(-1) = let\ \phi_1 = \bot_{D^\sharp}\ in$

$let\ \phi_2 = \mathcal{A}^{spi}(\![\ \mathbf{0}\ ]\!)\ \rho\ \phi_{\mathcal{A}}\ in$

$if\ \phi_1 = \phi_2\ then\ \phi_1\ else\ \mathcal{F}^{spi}_{\mathcal{A}}(0)$

$\mathcal{F}^{spi}_{\mathcal{E}}(-1) = let\ v_1 = (\bot_{Pi_\bot}, \bot_D)\ in$

$let\ v_2 = \mathcal{E}^{spi}(\![\ \mathbf{0}\ ]\!)\ \rho\ \phi_{\mathcal{E}}\ in$

$if\ v_1 = v_2\ then\ v_1\ else\ \mathcal{F}^{spi}_{\mathcal{E}}(0)$

From the above two calculations, we have that $\bot_{D^\sharp} = \phi_{\mathcal{A}0}$ is a safe abstraction of $\bot_D = \phi_{\mathcal{E}0}$ since $\forall x \in dom(\phi_{\mathcal{E}0}) : \phi_{\mathcal{E}0}(x) = \phi_{\mathcal{A}0}(x) = \{\}$. Also, since we proved the case for $P = \mathbf{0}$ (The second computation of $\mathcal{F}^{spi}$) earlier, then we have that:

$(\mathcal{E}^{spi}(\![\mathbf{0}]\!)\ \rho\ \phi_{\mathcal{E}} = (p, \phi'_{\mathcal{E}}))\ \wedge\ (\mathcal{A}^{spi}(\![\mathbf{0}]\!)\ \rho\ \phi_{\mathcal{A}} = \phi'_{\mathcal{A}})\ \wedge$

$(\exists M \in Term : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) \in \phi_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\}\ \wedge$

$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

$\Rightarrow\quad (\exists M \in Term : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, M) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi'_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\}\ \wedge$

$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

The induction step:

We start by assuming the inductive hypothesis:
$(\mathcal{F}_{\mathcal{E}}^{spi}(n) = p', \phi'_{\mathcal{E}}) \ \wedge \ (\mathcal{F}_{\mathcal{A}}^{spi}(n) = \phi'_{\mathcal{A}}) \wedge$
$(\exists M \in \mathit{Term} : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) \in \phi_{\mathcal{E}}(x) \ \Rightarrow \ \exists t \in \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) : \mathit{value\_of}(\{t\}) = \{M'\} \wedge$
$\mathit{untag}(M') = (\forall y \in \mathit{bnv}(M) : M[\alpha_{k,k'}(y)/y]))$

$\Rightarrow \quad (\exists M \in \mathit{Term} : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, M) \in \phi'_{\mathcal{E}}(x) \ \Rightarrow \ \exists t \in \phi'_{\mathcal{A}}(\alpha_{k,k'}(x)) : \mathit{value\_of}(\{t\}) = \{M'\} \wedge$
$\mathit{untag}(M') = (\forall y \in \mathit{bnv}(M) : M[\alpha_{k,k'}(y)/y]))$

Now, we prove that the same safety relation holds for $\mathcal{F}_{\mathcal{A}}^{spi}(n+1)$ with respect to $\mathcal{F}_{\mathcal{E}}^{spi}(n+1)$:
$\mathcal{F}_{\mathcal{A}}^{spi}(n+1) = \mathit{let}\ \phi_1 = \mathcal{A}^{spi}(\![\ \prod_{i=1}^{n+1} \mathit{ren}(P,i)\ ]\!)\ \rho\ \phi_{\mathcal{A}}\ \mathit{in}$
$\mathit{let}\ \phi_2 = \mathcal{A}^{spi}(\![\ \prod_{i=1}^{n+2} \mathit{ren}(P,i)\ ]\!)\ \rho\ \phi_{\mathcal{A}}\ \mathit{in}$
$\mathit{if}\ \phi_1 = \phi_2\ \mathit{then}\ \phi_1\ \mathit{else}\ \mathcal{F}_{\mathcal{A}}^{spi}(n+2)$

$\mathcal{F}_{\mathcal{E}}^{spi}(n+1) = \mathit{let}\ v_1 = \mathcal{E}^{spi}(\![\ \prod_{i=1}^{n+1} P[bnv_i(P)/bnv(P)]\ ]\!)\ \rho\ \phi_{\mathcal{E}}\ \mathit{in}$
$\mathit{let}\ v_2 = \mathcal{E}^{spi}(\![\ \prod_{i=1}^{n+2} P[bn_i(P)/bn(P)]\ ]\!)\ \rho\ \phi_{\mathcal{E}}\ \mathit{in}$
$\mathit{if}\ v_1 = v_2\ \mathit{then}\ v_1\ \mathit{else}\ \mathcal{F}_{\mathcal{E}}^{spi}(n+2)$

From the induction hypothesis, we know that:
$(\mathcal{E}^{spi}(\![\ \prod_{i=1}^{n+1} P[bnv_i(P)/bnv(P)]\ ]\!)\ \rho\ \phi_{\mathcal{E}} = p', \phi'_{\mathcal{E}}) \ \wedge \ (\mathcal{A}^{spi}(\![\ \prod_{i=1}^{n+1} \mathit{ren}(P,i)\ ]\!)\ \rho\ \phi_{\mathcal{A}} = \phi'_{\mathcal{A}}) \wedge$
$(\exists M \in \mathit{Term} : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) \in \phi_{\mathcal{E}}(x) \ \Rightarrow \ \exists t \in \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) : \mathit{value\_of}(\{t\}) = \{M'\} \wedge$
$\mathit{untag}(M') = (\forall y \in \mathit{bnv}(M) : M[\alpha_{k,k'}(y)/y]))$

$\Rightarrow \quad (\exists M \in \mathit{Term} : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, M) \in \phi'_{\mathcal{E}}(x) \ \Rightarrow \ \exists t \in \phi'_{\mathcal{A}}(\alpha_{k,k'}(x)) : \mathit{value\_of}(\{t\}) = \{M'\} \wedge$
$\mathit{untag}(M') = (\forall y \in \mathit{bnv}(M) : M[\alpha_{k,k'}(y)/y]))$

The remaining case involves:
$\mathcal{A}^{spi}(\![\ \prod_{i=1}^{n+2} \mathit{ren}(P,i)\ ]\!)\ \rho\ \phi_{\mathcal{A}}$ and $\mathcal{E}^{spi}(\![\ \prod_{i=1}^{n+2} P[bnv_i(P)/bnv(P)]\ ]\!)\ \rho\ \phi_{\mathcal{E}}$.
We can rewrite these as:
$\mathcal{A}^{spi}(\![\ \prod_{i=1}^{n+2} \mathit{ren}(P,i)\ ]\!)\ \rho\ \phi_{\mathcal{A}} = \mathcal{A}^{spi}(\![\ \prod_{i=1}^{n+1} \mathit{ren}(P,i) \ \mid \ \mathit{ren}(P,n+2)\ ]\!)\ \rho\ \phi_{\mathcal{A}}$

$\mathcal{E}^{spi}(\![\ \prod_{i=1}^{n+2} P[bnv_i(P)/bnv(P)]\ ]\!)\ \rho\ \phi_{\mathcal{E}} =$
$\mathcal{E}^{spi}(\![\ \prod_{i=1}^{n+1} P[bnv_i(P)/bnv(P)] \ \mid \ P[bnv_{n+2}(P)/bnv(P)]\ ]\!)\ \rho\ \phi_{\mathcal{E}}$

From the induction hypothesis, we have that:
$(\mathcal{E}^{spi}(\![\prod_{i=1}^{n+1} P[bnv_i(P)/bnv(P)]\,]\!)\ \rho\ \phi_{\mathcal{E}} = p', \phi'_{\mathcal{E}})\ \wedge\ (\mathcal{A}^{spi}(\![\prod_{i=1}^{n+1} ren(P,i)\,]\!)\ \rho\ \phi_{\mathcal{A}} = \phi'_{\mathcal{A}})\ \wedge$
$(\exists M \in Term : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) \in \phi_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\}\ \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

$\Rightarrow\quad (\exists M \in Term : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, M) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi'_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\}\ \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

And from the antecedent:
$(\mathcal{E}^{spi}(\![\ P[bnv_{n+2}(P)/bnv(P)]\ ]\!)\ \rho\ \phi_{\mathcal{E}} = p', \phi'_{\mathcal{E}})\ \wedge\ (\mathcal{A}^{spi}(\![\ ren(P, n+2)\ ]\!)\ \rho\ \phi_{\mathcal{A}} = \phi'_{\mathcal{A}})\ \wedge$
$(\exists M \in Term : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) \in \phi_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\}\ \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

$\Rightarrow\quad (\exists M \in Term : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, M) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi'_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\}\ \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

We have already proven the case for the parallel composition of two processes,
$\mathcal{A}^{spi}(\![\ P \mid Q\ ]\!)\ \rho\ \phi_{\mathcal{A}}$ and $\mathcal{E}^{spi}(\![\ P \mid Q\ ]\!)\ \rho\ \phi_{\mathcal{E}}$ (case 5). Hence, we can conclude that:
$(\mathcal{F}^{spi}_{\mathcal{E}}(n+1) = p', \phi'_{\mathcal{E}})\ \wedge\ (\mathcal{F}^{spi}_{\mathcal{A}}(n+1) = \phi'_{\mathcal{A}})\ \wedge$
$(\exists M \in Term : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) \in \phi_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\}\ \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

$\Rightarrow\quad (\exists M \in Term : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, M) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi'_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\}\ \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

From both the base case and the induction step, we can conclude that:
$(\mathcal{E}^{spi}(\![\ !P\ ]\!)\ \rho\ \phi_{\mathcal{E}} = p', \phi'_{\mathcal{E}})\ \wedge\ (\mathcal{A}^{spi}(\![\ !P\ ]\!)\ \rho\ \phi_{\mathcal{A}} = \phi'_{\mathcal{A}})\ \wedge$
$(\exists M \in Term : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) \in \phi_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\}\ \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

$\Rightarrow\quad (\exists M \in Term : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, M) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi'_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\}\ \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

Case 6: *if* $M = N$ *then* P *else* Q

$\mathcal{A}^{spi}(\![\mathit{if}\ M = L\ \mathit{then}\ P\ \mathit{else}\ Q]\!)\ \rho\ \phi_{\mathcal{A}} =$

$$\begin{cases} \mathcal{R}^{spi}(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)\ \phi_{\mathcal{A}}, & \text{if, } \mathit{untag}(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, M)) \cap \mathit{untag}(\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, L)) \neq \{\} \\ \mathcal{R}^{spi}(\![\{\!|Q|\!\}_\rho \uplus_\rho \rho]\!)\ \phi_{\mathcal{A}}, & \text{otherwise} \end{cases}$$

where, $\mathcal{R}^{spi}(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)_s\ \phi_{\mathcal{A}} = \bigcup_{P \in \rho'_P}^{\phi} \mathcal{A}^{spi}(\![P]\!)_s\ \rho''_P\ \phi_{\mathcal{A}}$

and, $\mathcal{R}^{spi}(\![\{\!|Q|\!\}_\rho \uplus_\rho \rho]\!)_s\ \phi_{\mathcal{A}} = \bigcup_{P \in \rho'_Q}^{\phi} \mathcal{A}^{spi}(\![Q]\!)_s\ \rho''_Q\ \phi_{\mathcal{A}}$ (by $\mathcal{A}^{spi}7$)

$\mathcal{E}^{spi}(\![\mathit{if}\ M = L\ \mathit{then}\ P\ \mathit{else}\ Q]\!)\ \rho\ \phi_{\mathcal{E}} =$

$$\begin{cases} \mathcal{R}^{spi}(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)\ \phi_{\mathcal{E}}, & \text{if } \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) = \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, L) \\ \mathcal{R}^{spi}(\![\{\!|Q|\!\}_\rho \uplus_\rho \rho]\!)\ \phi_{\mathcal{E}}, & \text{otherwise} \end{cases}$$

where, $\mathcal{R}^{spi}(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)_s\ \phi_{\mathcal{E}} = \bigcup_{P \in \rho'_P}^{\phi} \mathcal{E}^{spi}(\![P]\!)_s\ \rho''_P\ \phi_{\mathcal{E}}$

and, $\mathcal{R}^{spi}(\![\{\!|Q|\!\}_\rho \uplus_\rho \rho]\!)_s\ \phi_{\mathcal{E}} = \bigcup_{P \in \rho'_Q}^{\phi} \mathcal{E}^{spi}(\![Q]\!)_s\ \rho''_Q\ \phi_{\mathcal{E}}$ (by $\mathcal{E}^{spi}7$)

Where, $\rho'_P = \{\!|P|\!\}_\rho \uplus_\rho \rho$, $\rho'_Q = \{\!|Q|\!\}_\rho \uplus_\rho \rho$, $\rho''_P = \rho'_P \backslash \{\!|P|\!\}_\rho$ and $\rho''_Q = \rho'_Q \backslash \{\!|Q|\!\}_\rho$.

Hence, from the inductive hypothesis for P (the same applies to Q):

$(\mathcal{E}^{spi}(\![P]\!)\ \rho''_P\ \phi_{\mathcal{E}} = \phi'_{\mathcal{E}})\ \wedge\ (\mathcal{A}^{spi}(\![P]\!)\ \rho''_P\ \phi_{\mathcal{A}} = \phi'_{\mathcal{A}})\ \wedge$

$(\exists M \in \mathit{Term} : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) \in \phi_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) : \mathit{value\_of}(\{t\}) = \{M'\} \wedge$

$\mathit{untag}(M') = (\forall y \in \mathit{bnv}(M) : M[\alpha_{k,k'}(y)/y]))$

$\Rightarrow\quad (\exists M \in \mathit{Term} : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, M) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi'_{\mathcal{A}}(\alpha_{k,k'}(x)) : \mathit{value\_of}(\{t\}) = \{M'\} \wedge$

$\mathit{untag}(M') = (\forall y \in \mathit{bnv}(M) : M[\alpha_{k,k'}(y)/y]))$

By the safety of the $\cup_\phi$ relation (Lemma 2), we arrive at the following result:

$(\bigcup_{P \in \rho'}^{\phi} \mathcal{E}^{spi}(\![P]\!)\ \rho''_P\ \phi_{\mathcal{E}}) = \phi\quad \wedge\quad (\bigcup_{P \in \rho'}^{\phi} \mathcal{A}^{spi}(\![P]\!)\ \rho''_P\ \phi_{\mathcal{A}}) = \phi'\quad \wedge$

$(\exists M \in \mathit{Term} : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) \in \phi_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) : \mathit{value\_of}(\{t\}) = \{M'\} \wedge$

$\mathit{untag}(M') = (\forall y \in \mathit{bnv}(M) : M[\alpha_{k,k'}(y)/y]))$

$\Rightarrow\quad (\exists M \in \mathit{Term} : \varphi_{\mathcal{E}}(\phi, M) \in \phi(x)\ \Rightarrow\ \exists t \in \phi'(\alpha_{k,k'}(x)) : \mathit{value\_of}(\{t\}) = \{M'\} \wedge$

$\mathit{untag}(M') = (\forall y \in \mathit{bnv}(M) : M[\alpha_{k,k'}(y)/y]))$

Which satisfies the induction step and the safety requirement.

Case 7: *let* $(x_1,\ldots,x_n) = (M_1,\ldots,M_n)$ *in* P *else* Q

$\mathcal{A}^{spi}(\![let\ (x_1,\ldots,x_n) = M\ in\ P\ else\ Q]\!)\ \rho\ \phi_{\mathcal{A}} =$

$$\begin{cases} \bigcup_{\phi} \quad \mathcal{R}^{spi}(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)\ \phi'_{\mathcal{A}}, \\ (M_1^{t_1},\ldots,M_n^{t_n})\in\varphi_{\mathcal{A}}(\phi_{\mathcal{A}},M) \\ \text{if } \exists (M_1^{t_1},\ldots,M_n^{t_n}) \in \varphi_{\mathcal{A}}(\phi_{\mathcal{A}},M) \\ \text{where, } \phi'_{\mathcal{A}} = \phi_{\mathcal{A}}[\alpha_{k,k'}(x_1) \mapsto \phi_{\mathcal{A}}(\alpha_{k,k'}(x_1)) \cup \{\alpha_{k,k'}(t_1)\},\ldots, \\ \qquad\qquad \alpha_{k,k'}(x_n) \mapsto \phi_{\mathcal{A}}(\alpha_{k,k'}(x_n)) \cup \{\alpha_{k,k'}(t_n)\}] \\ \mathcal{R}^{spi}(\![\{\!|Q|\!\}_\rho \uplus_\rho \rho]\!)\ \phi_{\mathcal{A}}, \qquad\qquad \text{otherwise} \end{cases} \quad \text{by } (\mathcal{A}^{spi}8)$$

$\mathcal{E}^{spi}(\![let\ (x_1,\ldots,x_n) = M\ in\ P\ else\ Q]\!)\ \rho\ \phi_{\mathcal{E}} =$

$$\begin{cases} \mathcal{R}^{spi}(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)\ \phi'_{\mathcal{E}}, \quad \text{if } \varphi_{\mathcal{E}}(\phi_{\mathcal{E}},M) = (t_1,\ldots,t_n) \\ \text{where, } \phi'_{\mathcal{E}} = \phi_{\mathcal{E}}[x_1 \mapsto \{t_1\},\ldots,x_n \mapsto \{t_n\}] \\ \mathcal{R}^{spi}(\![\{\!|Q|\!\}_\rho \uplus_\rho \rho]\!)\ \phi_{\mathcal{E}}, \quad \text{otherwise} \end{cases} \quad \text{by } (\mathcal{E}^{spi}8)$$

We extend the following constructs:

$\mathcal{R}^{spi}(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)\ \phi'_{\mathcal{A}} = \bigcup_{\substack{\phi \\ P\in\rho'}} \mathcal{A}^{spi}(\![P]\!)\ \rho''\ \phi'_{\mathcal{A}}$

$\mathcal{R}^{spi}(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)\ \phi'_{\mathcal{E}} = (\biguplus_{P\in\rho'} fst(\mathcal{E}^{spi}(\![P]\!)\ \rho''\ \phi'_{\mathcal{A}})\ ,\ \bigcup_{\substack{\phi \\ P\in\rho'}} snd(\mathcal{E}^{spi}(\![P]\!)\ \rho''\ \phi'_{\mathcal{A}}))$

where, $\rho'' = \rho' \backslash \{\!|P|\!\}_\rho$ and $\rho' = \{\!|P|\!\}_\rho \uplus_\rho \rho$ (the same expansion applies to Q)

From the induction hypothesis, and $\forall N \in Term$, we have that:

$(\exists N \in Term : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}},N) \in \phi_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{N'\} \wedge$
$untag(N') = (\forall y \in bnv(N) : N[\alpha_{k,k'}(y)/y]))$

And from the above values of $\phi'_{\mathcal{E}}$ and $\phi'_{\mathcal{A}}$, we can arrive at:

$(\exists M_j \in Term : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}},M_j) \in \phi_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'_j\} \wedge$
$untag(M'_j) = (\forall y \in bnv(M_j) : M_j[\alpha_{k,k'}(y)/y]))$

$\Rightarrow \quad (\exists N \in Term : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}},N) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi'_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{N'\} \wedge$
$untag(N') = (\forall y \in bnv(N) : N[\alpha_{k,k'}(y)/y]))$

Hence, from the inductive hypothesis for P, we have that:

$(\mathcal{E}^{spi}(\![P]\!)\ \rho''\ \phi'_{\mathcal{E}} = (p'',\phi''_{\mathcal{E}}))\ \wedge\ (\mathcal{A}^{spi}(\![P]\!)\ \rho''\ \phi'_{\mathcal{A}} = \phi''_{\mathcal{A}})\ \wedge$
$(\exists M \in Term : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}},M) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi'_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\} \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

$\Rightarrow \quad (\exists M \in Term : \varphi_{\mathcal{E}}(\phi''_{\mathcal{E}},M) \in \phi''_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi''_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\} \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

By the safety of the $\cup_\phi$ relation (Lemma 2), we arrive at the following result:

$((\biguplus_{P\in\rho'} fst(\mathcal{E}^{spi}(\llbracket P\rrbracket)\ \rho''\ \phi'_{\mathcal{A}})\ \ ,\ \ \bigcup_{P\in\rho'}^{\phi} snd(\mathcal{E}^{spi}(\llbracket P\rrbracket)\ \rho''\ \phi'_{\mathcal{A}}))) = (p,\phi)\ \ \wedge$

$(\bigcup_{P\in\rho'}^{\phi} \mathcal{A}^{spi}(\llbracket P\rrbracket)\ \rho''\ \phi'_{\mathcal{A}}) = \phi'\ \ \wedge$

$(\exists M \in Term : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, M) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi'_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\} \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

$\Rightarrow\quad (\exists M \in Term : \varphi_{\mathcal{E}}(\phi, M) \in \phi(x)\ \Rightarrow\ \exists t \in \phi'(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\} \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

Which satisfies the induction step and the safety requirement.

Case 8: *case* $\{M\}_N$ *of* $\{x\}_N$ *in* P *else* Q

$\mathcal{A}^{spi}(\llbracket case\ \{M\}_N\ of\ \{x\}_N\ in\ P\ else\ Q\rrbracket)\ \rho\ \phi_{\mathcal{A}} =$

$$\begin{cases} \bigcup_{\{M^{tag}\}_n \in \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, \{M\}_N)}^{\phi} \mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \rho\rrbracket)\ \phi'_{\mathcal{A}}, & \text{if, } n \in \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, N) \\ \text{where, } \phi'_{\mathcal{A}} = \phi_{\mathcal{A}}[\alpha_{k,k'}(x) \mapsto \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) \cup \{\alpha_{k,k'}(tag)\}] & \text{by } (\mathcal{A}^{spi}9) \\ \mathcal{R}^{spi}(\llbracket \{\!|Q|\!\}_\rho \uplus_\rho \rho\rrbracket)\ \phi_{\mathcal{A}}, & \text{otherwise} \end{cases}$$

$\mathcal{E}^{spi}(\llbracket case\ \{M\}_N\ of\ \{x\}_N\ in\ P\ else\ Q\rrbracket)\ \rho\ \phi_{\mathcal{E}} =$

$$\begin{cases} \mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \rho\rrbracket)\ \phi'_{\mathcal{E}}, & \text{if } \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, \{M\}_N) = sec(t,k) \text{ and } \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, N) = k \\ \text{where, } \phi'_{\mathcal{E}} = \phi_{\mathcal{E}}[x \mapsto \{t\}] & \text{by } (\mathcal{E}^{spi}9) \\ \mathcal{R}^{spi}(\llbracket \{\!|Q|\!\}_\rho \uplus_\rho \rho\rrbracket)\ \phi_{\mathcal{E}}, & \text{otherwise} \end{cases}$$

Where, $\rho' = \{\!|P|\!\}_\rho \uplus_\rho \rho$ and $\rho'' = \rho' \backslash \{\!|P|\!\}_\rho$.

We extend the following constructs:

$\mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \rho\rrbracket)\ \phi'_{\mathcal{A}} = \bigcup_{P\in\rho'}^{\phi} \mathcal{A}^{spi}(\llbracket P\rrbracket)\ \rho''\ \phi'_{\mathcal{A}}$

$\mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \rho\rrbracket)\ \phi'_{\mathcal{E}} = (\biguplus_{P\in\rho'} fst(\mathcal{E}^{spi}(\llbracket P\rrbracket)\ \rho''\ \phi'_{\mathcal{A}})\ \ ,\ \ \bigcup_{P\in\rho'}^{\phi} snd(\mathcal{E}^{spi}(\llbracket P\rrbracket)\ \rho''\ \phi'_{\mathcal{A}}))$

From the induction hypothesis and $\forall N \in Term$:

$(\exists N \in Term : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, N) \in \phi_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{N'\} \wedge$
$untag(N') = (\forall y \in bnv(N) : N[\alpha_{k,k'}(y)/y]))$

Setting $\phi'_{\mathcal{E}} = \phi_{\mathcal{E}}[x \mapsto \{t\}]$ and $\phi'_{\mathcal{A}} = \phi_{\mathcal{A}}[\alpha_{k,k'}(x) \mapsto\ \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) \cup \{\alpha_{k,k'}(tag)\}]$, then:

$(\exists T \in Term : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, T) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow\ \exists tag \in \phi'_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{tag\}) = \{T'\} \wedge$
$untag(T') = (\forall y \in bnv(T) : T[\alpha_{k,k'}(y)/y]))$ Where $\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, T) = t$ in rule $(\mathcal{E}^{spi}9)$

$\Rightarrow \quad (\exists M \in Term : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, M) \in \phi'_{\mathcal{E}}(x) \ \Rightarrow \ \exists t \in \phi'_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\} \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

Hence, from the inductive hypothesis for P, we have:
$(\mathcal{E}^{spi}(\![P]\!)\ \rho''\ \phi'_{\mathcal{E}} = \phi''_{\mathcal{E}}) \ \wedge \ (\mathcal{A}^{spi}(\![P]\!)\ \rho''\ \phi'_{\mathcal{A}} = \phi''_{\mathcal{A}}) \ \wedge$
$(\exists M \in Term : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) \in \phi_{\mathcal{E}}(x) \ \Rightarrow \ \exists t \in \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\} \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

$\Rightarrow \quad (\exists M \in Term : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, M) \in \phi'_{\mathcal{E}}(x) \ \Rightarrow \ \exists t \in \phi'_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\} \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

By the safety of the $\cup_\phi$ relation (Lemma 2), we arrive at the following result:
$(\underset{P\in\rho'}{\uplus} fst(\mathcal{E}^{spi}(\![P]\!)\ \rho''\ \phi'_{\mathcal{A}}) \ , \ \underset{P\in\rho'}{\bigcup_\phi} snd(\mathcal{E}^{spi}(\![P]\!)\ \rho''\ \phi'_{\mathcal{A}})) = (p, \phi) \ \wedge$
$(\underset{P\in\rho'}{\bigcup_\phi} \mathcal{A}^{spi}(\![P]\!)\ \rho''\ \phi'_{\mathcal{A}}) = \phi' \ \wedge$

$(\exists M \in Term : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, M) \in \phi'_{\mathcal{E}}(x) \ \Rightarrow \ \exists t \in \phi'_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\} \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

$\Rightarrow \quad (\exists M \in Term : \varphi_{\mathcal{E}}(\phi, M) \in \phi(x) \ \Rightarrow \ \exists t \in \phi'(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\} \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

Which satisfies the induction step and the safety requirement.

Case 9: *case* $\{\!|M|\!\}_N$ *of* $\{\!|x|\!\}_N$ *in* P *else* Q

$\mathcal{A}^{spi}(\![case\ \{\!|M|\!\}_N\ of\ \{\!|x|\!\}_N\ in\ P\ else\ Q]\!)\ \rho\ \phi_{\mathcal{A}} =$

$$\begin{cases} \underset{\{\![M^{tag}]\!\}_{n^+}\in\varphi_{\mathcal{A}}(\phi_{\mathcal{A}},\{\![M]\!\}_N)}{\bigcup_\phi} \mathcal{R}^{spi}(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)\ \phi'_{\mathcal{A}}, & \text{if, } n^- \in \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, N) \\ \text{where, } \phi'_{\mathcal{A}} = \phi_{\mathcal{A}}[\alpha_{k,k'}(x) \mapsto \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) \cup \{\alpha_{k,k'}(tag)\}] & \text{by } (\mathcal{A}^{spi}9) \\ \mathcal{R}^{spi}(\![\{\!|Q|\!\}_\rho \uplus_\rho \rho]\!)\ \phi_{\mathcal{A}}, & \text{otherwise} \end{cases}$$

$\mathcal{E}^{spi}(\![case\ \{\!|M|\!\}_N\ of\ \{\!|x|\!\}_N\ in\ P\ else\ Q]\!)\ \rho\ \phi_{\mathcal{E}} =$

$$\begin{cases} \mathcal{R}^{spi}(\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!)\ \phi'_{\mathcal{E}}, & \text{if } \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, \{\![M]\!\}_N) = pub(t, k^+) \text{ and } \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, N) = k^- \\ \text{where, } \phi'_{\mathcal{E}} = \phi_{\mathcal{E}}[x \mapsto \{t\}] & \text{by} \\ \mathcal{R}^{spi}(\![\{\!|Q|\!\}_\rho \uplus_\rho \rho]\!)\ \phi_{\mathcal{E}}, & \text{otherwise} \end{cases}$$
$(\mathcal{E}^{spi}9)$

Where, $\rho' = \{\!|P|\!\}_\rho \uplus_\rho \rho$ and $\rho'' = \rho' \backslash \{\!|P|\!\}_\rho$.

We extend the following constructs:

$\mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi'_{\mathcal{A}} = \bigcup_{P \in \rho'}^{\phi} \mathcal{A}^{spi}(\llbracket P \rrbracket)\ \rho''\ \phi'_{\mathcal{A}}$

$\mathcal{R}^{spi}(\llbracket \{\!|P|\!\}_\rho \uplus_\rho \rho \rrbracket)\ \phi'_{\mathcal{E}} = (\biguplus_{P \in \rho'} fst(\mathcal{E}^{spi}(\llbracket P \rrbracket)\ \rho''\ \phi'_{\mathcal{A}})\ \ ,\ \ \bigcup_{P \in \rho'}^{\phi} snd(\mathcal{E}^{spi}(\llbracket P \rrbracket)\ \rho''\ \phi'_{\mathcal{A}}))$

From the induction hypothesis and $\forall N \in Term$:

$(\exists N \in Term : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, N) \in \phi_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{N'\} \wedge$
$untag(N') = (\forall y \in bnv(N) : N[\alpha_{k,k'}(y)/y]))$

Setting $\phi'_{\mathcal{E}} = \phi_{\mathcal{E}}[x \mapsto \{t\}]$ and $\phi'_{\mathcal{A}} = \phi_{\mathcal{A}}[\alpha_{k,k'}(x) \mapsto\ \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) \cup \{\alpha_{k,k'}(tag)\}]$, then:

$(\exists T \in Term : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, T) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow\ \exists tag \in \phi'_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{tag\}) = \{T'\} \wedge$
$untag(T') = (\forall y \in bnv(T) : T[\alpha_{k,k'}(y)/y]))$ Where $\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, T) = t$ in rule ($\mathcal{E}^{spi}9$)

$\Rightarrow\quad (\exists M \in Term : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, M) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi'_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\} \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

Hence, from the inductive hypothesis for P, we have:

$(\mathcal{E}^{spi}(\llbracket P \rrbracket)\ \rho''\ \phi'_{\mathcal{E}} = \phi''_{\mathcal{E}})\ \wedge\ (\mathcal{A}^{spi}(\llbracket P \rrbracket)\ \rho''\ \phi'_{\mathcal{A}} = \phi''_{\mathcal{A}})\ \wedge$
$(\exists M \in Term : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) \in \phi_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\} \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

$\Rightarrow\quad (\exists M \in Term : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, M) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi'_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\} \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

By the safety of the $\cup_\phi$ relation (Lemma 2), we arrive at the following result:

$(\biguplus_{P \in \rho'} fst(\mathcal{E}^{spi}(\llbracket P \rrbracket)\ \rho''\ \phi'_{\mathcal{A}})\ \ ,\ \ \bigcup_{P \in \rho'}^{\phi} snd(\mathcal{E}^{spi}(\llbracket P \rrbracket)\ \rho''\ \phi'_{\mathcal{A}})) = (p, \phi)\ \wedge$
$(\bigcup_{P \in \rho'}^{\phi} \mathcal{A}^{spi}(\llbracket P \rrbracket)\ \rho''\ \phi'_{\mathcal{A}}) = \phi'\ \wedge$

$(\exists M \in Term : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, M) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi'_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\} \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

$\Rightarrow\quad (\exists M \in Term : \varphi_{\mathcal{E}}(\phi, M) \in \phi(x)\ \Rightarrow\ \exists t \in \phi'(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\} \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

Which satisfies the induction step and the safety requirement.

Case: 10 *case* $[\![M]\!]_N$ *of* $[\![x]\!]_N$ *in* P *else* Q

$\mathcal{A}^{spi}([\![case\ [\![M]\!]_N\ of\ [\![x]\!]_N\ in\ P\ else\ Q]\!])\ \rho\ \phi_{\mathcal{A}} =$

$$\begin{cases} \bigcup_{\phi \atop [\![M^{tag}]\!]_{n^-} \in \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, [\![M]\!]_N)} \mathcal{R}^{spi}([\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!])\ \phi'_{\mathcal{A}}, & \text{if, } n^+ \in \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, N) \\ \text{where, } \phi'_{\mathcal{A}} = \phi_{\mathcal{A}}[\alpha_{k,k'}(x) \mapsto \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) \cup \{\alpha_{k,k'}(tag)\}] & \text{by } (\mathcal{A}^{spi}9) \\ \mathcal{R}^{spi}([\![\{\!|Q|\!\}_\rho \uplus_\rho \rho]\!])\ \phi_{\mathcal{A}}, & \text{otherwise} \end{cases}$$

$\mathcal{E}^{spi}([\![case\ [\![M]\!]_N\ of\ [\![x]\!]_N\ in\ P\ else\ Q]\!])\ \rho\ \phi_{\mathcal{E}} =$

$$\begin{cases} \mathcal{R}^{spi}([\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!])\ \phi'_{\mathcal{E}}, & \text{if } \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, [\![M]\!]_N) = sig(t, k^-) \text{ and } \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, N) = k^+ \\ \text{where, } \phi'_{\mathcal{E}} = \phi_{\mathcal{E}}[x \mapsto \{t\}] & \text{by } (\mathcal{E}^{spi}9) \\ \mathcal{R}^{spi}([\![\{\!|Q|\!\}_\rho \uplus_\rho \rho]\!])\ \phi_{\mathcal{E}}, & \text{otherwise} \end{cases}$$

Where, $\rho' = \{\!|P|\!\}_\rho \uplus_\rho \rho$ and $\rho'' = \rho' \backslash \{\!|P|\!\}_\rho$.

We extend the following constructs:

$\mathcal{R}^{spi}([\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!])\ \phi'_{\mathcal{A}} = \bigcup_{P \in \rho'}^{\phi} \mathcal{A}^{spi}([\![P]\!])\ \rho''\ \phi'_{\mathcal{A}}$

$\mathcal{R}^{spi}([\![\{\!|P|\!\}_\rho \uplus_\rho \rho]\!])\ \phi'_{\mathcal{E}} = (\ \biguplus_{P \in \rho'} fst(\mathcal{E}^{spi}([\![P]\!])\ \rho''\ \phi'_{\mathcal{A}})\ \ ,\ \ \bigcup_{P \in \rho'}^{\phi} snd(\mathcal{E}^{spi}([\![P]\!])\ \rho''\ \phi'_{\mathcal{A}}))$

From the induction hypothesis and $\forall N \in Term$:

$(\exists N \in Term : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, N) \in \phi_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{N'\} \wedge$
$untag(N') = (\forall y \in bnv(N) : N[\alpha_{k,k'}(y)/y]))$

Setting $\phi'_{\mathcal{E}} = \phi_{\mathcal{E}}[x \mapsto \{t\}]$ and $\phi'_{\mathcal{A}} = \phi_{\mathcal{A}}[\alpha_{k,k'}(x) \mapsto\ \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) \cup \{\alpha_{k,k'}(tag)\}]$, then:

$(\exists T \in Term : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, T) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow\ \exists tag \in \phi'_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{tag\}) = \{T'\} \wedge$
$untag(T') = (\forall y \in bnv(T) : T[\alpha_{k,k'}(y)/y]))$ Where $\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, T) = t$ in rule $(\mathcal{E}^{spi}9)$

$\Rightarrow\quad (\exists M \in Term : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, M) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi'_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\} \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

Hence, from the inductive hypothesis for P, we have:

$(\mathcal{E}^{spi}([\![P]\!])\ \rho''\ \phi'_{\mathcal{E}} = \phi''_{\mathcal{E}})\ \wedge\ (\mathcal{A}^{spi}([\![P]\!])\ \rho''\ \phi'_{\mathcal{A}} = \phi''_{\mathcal{A}})\ \wedge$
$(\exists M \in Term : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) \in \phi_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\} \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

$\Rightarrow\quad (\exists M \in Term : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, M) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi'_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\} \wedge$
$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

By the safety of the $\cup_\phi$ relation (Lemma 2), we arrive at the following result:

$(\biguplus_{P\in\rho'} fst(\mathcal{E}^{spi}(\llbracket P\rrbracket\ \rho''\ \phi'_{\mathcal{A}})\ \ ,\ \ \bigcup_{P\in\rho'}^{\phi} snd(\mathcal{E}^{spi}(\llbracket P\rrbracket\ \rho''\ \phi'_{\mathcal{A}})) = (p,\phi)\ \ \wedge$

$(\bigcup_{P\in\rho'}^{\phi} \mathcal{A}^{spi}(\llbracket P\rrbracket\ \rho''\ \phi'_{\mathcal{A}}) = \phi'\ \ \wedge$

$(\exists M \in Term : \varphi_{\mathcal{E}}(\phi'_{\mathcal{E}}, M) \in \phi'_{\mathcal{E}}(x)\ \Rightarrow\ \exists t \in \phi'_{\mathcal{A}}(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\} \wedge$

$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

$\Rightarrow \quad (\exists M \in Term : \varphi_{\mathcal{E}}(\phi, M) \in \phi(x)\ \Rightarrow\ \exists t \in \phi'(\alpha_{k,k'}(x)) : value\_of(\{t\}) = \{M'\} \wedge$

$untag(M') = (\forall y \in bnv(M) : M[\alpha_{k,k'}(y)/y]))$

Which satisfies the induction step and the safety requirement.

□