

THEORY AND TOOL SUPPORT FOR THE FORMAL VERIFICATION OF CRYPTOGRAPHIC PROTOCOLS

THÈSE N° 4007 (2008)

PRÉSENTÉE LE 25 JANVIER 2008

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS

Laboratoire de méthodes de programmation 1

SECTION D'INFORMATIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Sébastien BRIAIS

DEA de programmation, Université de Paris VII, France
et de nationalité française

acceptée sur proposition du jury:

Prof. P. lenne, président du jury
Prof. U. Nestmann, Prof. M. Odersky, directeurs de thèse
Dr D. Hirschkoff, rapporteur
Prof. V. Kuncak, rapporteur
M. D. Miller, rapporteur



ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Lausanne, EPFL

2008

Contents

1	Introduction	7
1.1	Cryptographic protocols	7
1.1.1	Security goals	7
1.1.2	Cryptographic primitives	8
1.1.3	A malicious agent : the intruder	12
1.2	Models and techniques	15
1.2.1	Two views of cryptography	15
1.2.2	The Dolev-Yao model	16
1.2.3	BAN logic	17
1.2.4	Model checking	17
1.2.5	The inductive approach	18
1.2.6	Rewrite systems, Horn clauses	18
1.2.7	Process calculi	18
1.3	Outline of the thesis	20
1.3.1	Contributions	21
1.3.2	Overview	21
2	The Pi Calculus	25
2.1	Introduction to the pi calculus	25
2.1.1	Syntax	25
2.1.2	Labelled transition system	30
2.1.3	Structural congruence	34
2.2	Several notions of bisimilarity	36
2.2.1	Ground bisimulation	37
2.2.2	Early bisimulation	38
2.2.3	Late bisimulation	39
2.2.4	Open bisimulation	40
2.2.5	A word on weak equivalences	47
2.3	Open bisimulation, revisited	49
2.3.1	A type-aware variant of open bisimulation	49
2.3.2	A knowledge-aware variant of open bisimulation	53

2.3.3	About congruence properties	62
3	The Spi Calculus	71
3.1	Syntax	71
3.1.1	The language of messages	71
3.1.2	Expressions	73
3.1.3	Processes	74
3.1.4	Example	77
3.2	Labelled Transitions System	77
3.2.1	Names, free names, bound names, α -conversion	77
3.2.2	Substitutions	79
3.2.3	Late Semantics	81
3.2.4	Semantics with Constraints	83
3.3	Equivalences	87
3.3.1	Structural congruence	87
3.3.2	Testing equivalence, barbed equivalence	89
3.3.3	Bisimulations	91
4	Representing environment knowledge as hedges	95
4.1	Synthesis	95
4.1.1	Definition	95
4.1.2	Comparing hedges power	97
4.2	Analysis	98
4.2.1	Weak analysis	98
4.2.2	Analysis	99
4.3	Irreducible hedges	107
4.3.1	Reducing hedges	107
4.3.2	Irreducible hedges	109
4.3.3	Characterisation of irreducible hedges	112
4.4	Consistent hedges	113
4.4.1	Inversing hedges	113
4.4.2	Consistency	114
4.4.3	Properties	117
5	Open Bisimulation for the Spi Calculus	119
5.1	Late hedged bisimulation	119
5.2	Open hedged bisimulation	121
5.2.1	S-environments	121
5.2.2	Open hedged bisimulation	134
5.2.3	Soundness of open hedged bisimulation	135
5.3	Up to techniques	138

5.3.1	Up to structural congruence	139
5.3.2	Up to bijective renamings	140
5.3.3	Up to respectful substitutions	141
5.3.4	Open hedged bisimulation is an extension of K-open bisimulation	141
5.4	Symbolic Characterisation	146
5.4.1	Symbolic transition system	146
5.4.2	Properties of the symbolic transition system	150
5.4.3	Symbolic open hedged bisimulation	154
5.4.4	Towards mechanisation	159
6	A Formalisation in coq	171
6.1	The formalisation	172
6.1.1	Representation of binders	172
6.1.2	The de Bruijn representation	175
6.1.3	Abstracting from the semantics	183
6.2	Proof sketches	187
6.2.1	Proof of Theorem 4	187
6.2.2	Proof of Theorem 13	189
6.2.3	Proof of Theorem 14	193
6.2.4	Late hedged bisimilarity in coq	202
7	From Protocol Narrations to Spi Calculus	211
7.1	Introduction	211
7.1.1	The setting	211
7.1.2	The challenge	212
7.1.3	Our approach	213
7.1.4	Tool support.	214
7.2	Extending protocol narrations	214
7.3	Compiling protocol narrations	216
7.3.1	Target syntax.	216
7.3.2	Evaluation of expressions and formulae.	218
7.3.3	Knowledge representation.	219
7.3.4	Generating checks.	222
7.3.5	Reducing knowledge sets.	224
7.3.6	The compilation.	225
7.4	A detailed example: the ASW protocol	228
7.4.1	The protocol	228
7.4.2	Compilation of the ASW protocol	229
7.4.3	The ASW protocol and the pattern-matching spi cal- culus	234

7.5	Executing protocol narrations	235
7.5.1	Binders and α -conversion.	236
7.5.2	Reordering.	237
7.5.3	Labelled transitions.	238
7.6	Rewriting protocol narrations ... into spi calculus	239
7.7	<code>spyer</code>	242
7.7.1	Simplifying formula.	243
7.7.2	The Wide-Mouthed Frog Protocol	245
7.7.3	The Otway-Rees protocol	246
7.8	Related work and future work	248
A	Curriculum Vitæ	269

List of Tables

1.1	Encryption function	9
1.2	Alice communicates M to Bob via their shared symmetric key k	9
1.3	Alice communicates M to Bob via Bob's public key $\text{pub}(k_B)$.	10
2.1	The processes of the pi calculus	27
2.2	The bound names of a process	28
2.3	The free names of a process	28
2.4	Notation for actions	30
2.5	The late semantics of the pi calculus	32
2.6	The process contexts of the pi calculus	35
2.7	Axioms of structural congruence	36
2.8	Relationships among equivalences	46
3.1	The messages of the spi calculus	72
3.2	The expressions of the spi calculus	73
3.3	Concrete evaluation of expressions	75
3.4	The processes of the spi calculus	76
3.5	The guards of the spi calculus	76
3.6	Evaluation of guards	77
3.7	Notation for actions.	83
3.8	The late semantics of the spi calculus	84
3.9	The late semantics of the spi calculus (with constraints) . . .	86
3.10	The process contexts of the spi calculus	88
3.11	Axioms of structural congruence	88
5.1	Abstract evaluation of expressions	148
5.2	The symbolic semantics of the spi calculus	149
5.3	Evaluation of symbolic actions	150
5.4	Evaluation of constraints	150
5.5	Type constraints of constraints	150

5.6	Definition of \Longrightarrow	162
6.1	Representation of $a(x).[Dec_k^x:M](vl)\bar{b}\langle l \rangle.0$ using de Bruijn notation	176
6.2	Definition of $mem_d(i, P)$ lifted on processes	177
6.3	Parallel composition of an abstraction and a process using de Bruijn representation	178
6.4	Architecture of the coq formalisation (de Bruijn operations)	183
6.5	Formalisation of the labelled semantics in coq	185
6.6	Implementation of actions for the standard labelled semantics of the spi calculus	186
6.7	Definition of $x \triangleleft E$	195
6.8	Definition of $\sharp(E)$	196
7.1	Wide-Mouthed Frog protocol	212
7.2	Protocol narrations	215
7.3	Wide-Mouthed Frog protocol, with formal declarations	216
7.4	Syntax of executable narrations	217
7.5	Evaluation of expressions (can fail, in particular if $v(E) \neq \emptyset$) and formulae	218
7.6	Synthesis	220
7.7	Analysis	221
7.8	Definition of $\mathcal{X}[\cdot]$	226
7.9	Substitution	236
7.10	Reordering	237
7.11	Labelled semantics of executable narrations	238
7.12	Definition of $\mathcal{A}(\cdot), R(\cdot)$	240
7.13	Definition of $\cdot \uparrow$	240
7.14	Correspondence of abstract and <code>spyer</code> syntax for expressions and formulae	242

Résumé

Les *protocoles cryptographiques* forment un ingrédient essentiel des communications réseau. S'ils semblent de prime abord relativement simples en comparaison d'autres algorithmes distribués, ils sont néanmoins réputés pour être difficile à élaborer. Cette difficulté s'explique par le fait qu'ils doivent garantir des propriétés de sécurité dans n'importe quel environnement, incluant potentiellement des intrus hostiles. Durant les dernières décennies sont apparus plusieurs modèles ainsi que des outils permettant d'étudier la correction des protocoles cryptographiques. Parmi ces modèles figure le spi calcul : un calcul de processus dérivé du pi calcul incorporant des primitives cryptographiques. Les calculs de processus tels que le spi calcul permettent de décrire de façon claire et concise des algorithmes distribués tels que les protocoles cryptographiques. De plus, le spi calcul permet de modéliser de manière élégante certaines propriétés de sécurité via des *équivalences observationnelles*.

Quand cette thèse a commencé, cette approche manquait cruellement d'outils. Inspirés par la situation existante dans le pi calcul, nous proposons une nouvelle notion d'équivalence observationnelle qui est proche de pouvoir être implémentée. En outre, nous proposons une formalisation en `Coq` de nos travaux qui non seulement valide notre étude théorique mais constituera aussi à terme la base d'un outil certifié qui automatisera les tests d'équivalence dans le spi calcul.

Pour compléter la suite d'outils, nous proposons également une sémantique formelle pour une notation informelle utilisée couramment pour décrire les protocoles cryptographiques : les narrations de protocole. Nous donnons finalement une manière rigoureuse de traduire les narrations de protocole en spi calcul, constituant ainsi les fondements de notre outil automatique de traduction *spyer*.

Mots-clé : protocoles cryptographiques, calculs de processus, spi calcul, bisimulations, assistants de preuve.

Abstract

Cryptographic protocols are an essential component of network communications. Despite their relatively small size compared to other distributed algorithms, they are known to be error-prone. This is due to the obligation to behave robustly in the context of unknown hostile attackers who might want to act against the security objectives of the jointly interacting entities. The need for techniques to verify the correctness of cryptographic protocols has stimulated the development of new frameworks and tools during the last decades. Among the various models is the spi calculus: a process calculus which is an extension of the pi calculus that incorporates cryptographic primitives. Process calculi such as the spi calculus offer the possibility to describe in a precise and concise way distributed algorithms such as cryptographic protocols. Moreover, spi calculus offers an elegant way to formalise some security properties of cryptographic protocols via *behavioural equivalences*.

At the time this thesis began, this approach lacked tool support. Inspired by the situation in the pi calculus, we propose a new notion of behavioural equivalence for the spi calculus that is close to an algorithm. Besides, we propose a `coq` formalisation of our results that not only validates our theoretical developments but also will eventually be the basis of a certified tool that would automate equivalence checking of spi calculus terms.

To complete the toolchain, we propose a formal semantics for an informal notation to describe cryptographic protocols, so called protocol narrations. We give a rigorous procedure to translate protocol narrations into spi calculus terms; this constitutes the foundations of our automatic translation tool `spyex`.

Keywords: cryptographic protocols, process calculi, spi calculus, bisimulations, proof assistants.

Remerciements

J'aimerais remercier Uwe Nestmann pour avoir accepté de me prendre en thèse sous sa direction. Même si son départ à Berlin n'a pas rendu la chose facile tous les jours, je crois que nous avons finalement réussi à bien travailler ensemble et je le remercie encore pour avoir répondu présent aux moments critiques. Je remercie également Martin Odersky qui a accepté de jouer le rôle de co-directeur de thèse lorsque Uwe a quitté l'EPFL. Je le remercie encore pour m'avoir ouvert les portes du LAMP, et ainsi donc avoir initié mon aventure helvète, alors que j'étais à la recherche d'un stage de maîtrise « à l'étranger ».

J'aimerais exprimer toute ma gratitude aux membres de mon jury de thèse. Merci à Paolo lenne pour avoir accepté de présider mon jury. Merci à Dale Miller, Daniel Hirschkoﬀ et Viktor Kuncak pour avoir accepté si gentiment de lire et évaluer ma thèse.

Je remercie Daniel Bünzli pour tous les moments que nous avons passés à nous exposer mutuellement nos problèmes, moments au cours desquels j'ai pu apprécié sa grande rigueur scientifique. Je le remercie également, ainsi que le reste de la bande « des vrais potes » (à savoir Michel Schinz, Philippe Altherr, Vincent Cremet, Gilles Dubochet et Rachele Fuzati) pour tous les bons moments que nous avons pu partager pendant et en dehors du travail. Je remercie aussi les autres LAMPions pour leur bonne humeur quotidienne ; je pense à Johannes Borgström pour la collaboration scientifique que nous avons menée ainsi qu'à Stéphane Micheloud pour les échanges que nous avons pu avoir pendant la pause de midi. Je remercie très chaleureusement Christophe Gensoul pour avoir fait son projet de semestre sous ma direction mais aussi pour tous les très bons moments que nous avons pu passer par la suite.

Je remercie François-Régis Sinot avec qui j'ai passé un très bon séjour à San Francisco lors de CONCUR'05.

Je remercie Benjamin Gandon pour son amitié sans faille.

Finalement, je remercie toute ma famille pour m'avoir supporté pendant toutes ces années. J'aurais aimé être encore plus près de vous quand nous avons du affronter certaines épreuves.

Chapter 1

Introduction

In this chapter, we explicate the practical motivations of this thesis and give an overview of related works. Based on this exposition, we explain the approach chosen here, and give an outline of the thesis.

1.1 Cryptographic protocols

The democratisation of the Internet has deeply changed the nature of human interactions. For instance, e-commerce offers the ability to buy almost anything while staying at home, with a simple mouse click. These new ways of interacting, through computers over an insecure network, require some technological support in order to guarantee security of communications; this is precisely the *raison d'être* of cryptographic protocols, also called security protocols.

According to [95], a *cryptographic protocol* is “a distributed algorithm defined by a sequence of steps precisely specifying the actions required of two or more entities to achieve a specific security objective”.

1.1.1 Security goals

The security objectives that are usually emphasised in the use of cryptographic protocols are the following:

1. *Confidentiality* or *secrecy* or *privacy*. It requires the content of messages or other information to be accessible only to the authorised parties.
2. *Data integrity* ensures that data has not been corrupted or altered, e.g. by an unauthorised party.

3. *Authentication* is related to identification. It is usually subdivided into two major classes: *entity authentication* and *data origin authentication*. Entity authentication means that one entity, often called a *principal*, is indeed who it claims to be. Data origin authentication means that the source of information is corroborated; it implicitly provides data integrity.
4. *Non-repudiation* prevents an entity from denying previous commitments or actions. When an entity denies having taken certain actions, a dispute arises. A procedure involving a trusted third party is needed to resolve it.

A more comprehensive overview of security goals can be found in [124]. The formal meaning of these goals is not always completely clear; modal logics have thus emerged to express them in a rigorous way [89].

1.1.2 Cryptographic primitives

Cryptographic protocols make use of cryptographic primitives in order to reach their security goals. We will have a very abstract view of cryptography and cryptographic primitives, nonetheless we discuss quickly the toolbox usable by a protocol designer.

Cipher algorithms

Cryptography has a long and fascinating history [88]. Greece and ancient Rome already made use of it for military or diplomatic purposes. More recently, one also remembers that it played a crucial role in the outcome of both world wars. It is only during the 1970s that the use of cryptography was extended to the civil field with the elaboration of the Data Encryption Standard [107]. In 1976, cryptography made its revolution with the introduction of the notion of public-key cryptography by Diffie and Hellman [66].

A cipher algorithm is a reversible cryptographic primitive. The *encryption function* is usually parametrised by an *encryption key* and it takes a message—the *plaintext*—to produce the *ciphertext* (see Table 1.1). To get back the plaintext, a *decryption function*, which is parametrised by a *decryption key*, is used.

We speak of *symmetric key* cryptography if for each pair (d, e) of decryption key and encryption key, it is easy to determine d knowing only e and to determine e from d ,

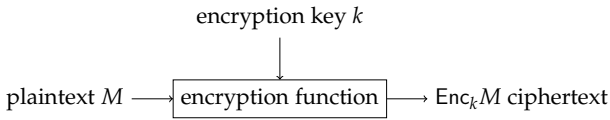
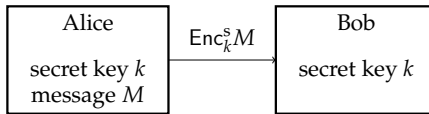


Table 1.1: Encryption function

In most practical symmetric-key encryption schemes, we have $e = d$. In this case, two parties which want to communicate have to *share* the same key k ; we thus sometimes speak of *shared key* cryptography (see Table 1.2). One of the major issues with symmetric-key systems is to find an efficient method to agree upon and exchange keys securely: this is the *key distribution problem*.

Table 1.2: Alice communicates M to Bob via their shared symmetric key k

In contrast, if for each pair (d, e) , given an encryption key e and a random cyphertext c it is computationally infeasible to find a plaintext m such that its encryption with e is c then we speak of *public key* cryptography. This implies that computing e from d or d from e is computationally infeasible.

In this setting, a two-party communication between Alice and Bob has the following form. Bob selects a key pair (d, e) and sends the encryption key e (called the *public key*) to Alice but keeps the decryption key d secure and secret. Alice may then send a message to Bob by applying the encryption function determined by Bob's public key. Bob can decrypt the cyphertext c by applying the decryption function using its private key d (see Table 1.3).

In public-key systems, it is needed to have a mean to authenticate public keys because otherwise an intruder may impersonate an honest principal and defeat the system without breaking the encryption scheme. Authentication is usually provided by a public key infrastructure (PKI) through *certificates*.

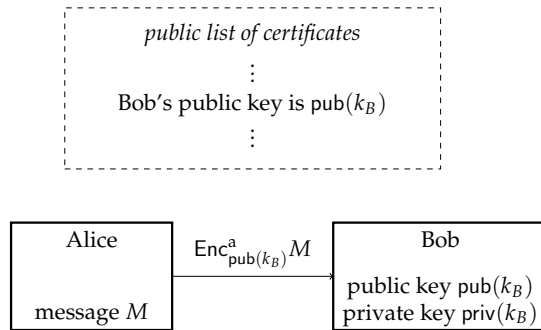


Table 1.3: Alice communicates M to Bob via Bob's public key $\text{pub}(k_B)$

We will not enter into the details of implementation of cipher algorithms or cipher modes of operation since it is beyond the scope of this thesis. A comprehensive and technical account of cipher algorithms can be found in [95]. We just mention as examples of shared-key cryptosystems the one-time pad (which ensure perfect secrecy), the *data encryption standard* [107] and its successor the *advanced encryption standard* [108]. Instances of public-key cryptosystems are for example RSA (named after its inventors Rivest-Shamir-Adleman) [119] or ElGamal [73].

Cryptographic hash functions

Quoting [95], a *hash function* "is a computationally efficient function mapping binary strings of arbitrary length to binary strings of some fixed length, called *hash values*". Hash functions can be used to detect data corruption. Indeed, assume that the hash value of a data M is $H(M)$ and that M happens to be corrupted into M' , then the hash value $H(M')$ will almost certainly be different from $H(M)$ and thus we will detect the corruption. Of course, we are assuming that the range of the hash function is large enough so that collisions are rare. It is also desirable that a small change to a data results in a large change of the hash value so that data corruption can be detected even if the hash value has also been corrupted.

To detect accidental corruption of data, a hash function such as a Cyclic Redundancy Check [117] is sufficient. However, to avoid malicious alteration of data, the choice of the hash function should be more careful. Indeed, if the hash function is simple, it can be fairly easy for someone to

choose an alteration of the data that have the same hash value. To prevent this, it is required that cryptographic hash functions have the following properties:

- Knowing H and given a hash value y , it is extremely difficult to find a data x such that $H(x) = y$.
- Knowing H and given an input x and its hash value $H(x)$, it is extremely difficult to find another input $x' \neq x$ such that $H(x') = H(x)$.
- Knowing H , it is extremely difficult to find two distinct inputs x, x' such that $H(x) = H(x')$.

Again, we will not enter into the details of cryptographic hash functions and will refer the reader to [95]. As candidates for cryptographic hash functions, we just mention MD5 [118] or preferably SHA-1 [109] or RIPEMD [67].

Digital signatures

Digital signatures are supposed to prove the origin and the authenticity of the data to which they are bound. They are the digital counterpart of handwritten signatures or seals. Digital signatures are usually implemented using public-key cryptography. A signature algorithm takes a document and a private key and produces a signed document. A signature verification algorithm takes a signed document and a public key and answers “yes” or “no”. If the answer is “yes”, we are entitled to conclude that the message has been signed with the matching private key and that the data has not been corrupted.

A digital signature scheme is the following:

- *Signature generation* Assume an entity A has private key $\text{priv}(k_A)$ and public key $\text{pub}(k_A)$. To sign a message M , it produces the concatenation of its identity A , the message M and the encryption of the hash value $H(M)$ of M with its own private key. In short, the signature is

$$A, M, \text{Enc}_{\text{priv}(k_A)}^a H(M)$$

- *Signature verification* To check a signed message, an entity B should read A 's identity, compute the hash value of the second part of the signature and accept the signature only if this hash value is equal to the decryption with A 's authentic public key $\text{pub}(k_A)$ of the third part of the signature.

This method of authentication relies on the assumption that the secret private key $\text{priv}(k_A)$ of A is never compromised to avoid that someone else sign a document with A 's private key.

Once again, we refer to [95] for a more comprehensive overview of digital signature algorithms.

Nonces and timestamps

To provide uniqueness or timeliness guarantees, time-variant parameters such as *nonces* or *timestamps* may be used in the elaboration of cryptographic protocols.

A nonce can be understood in the context of cryptographic protocols as a fresh random value. They are usually used to establish causal relationships between messages; typically in challenge-response protocols to ensure *freshness* of data. Nonces can be constrained by a *timeout period* to fix a relative point in time for the parties involved.

Another approach is to agree on a global notion of time and use timestamps instead. A timestamp is simply a way of attaching the current value of time to a message. It can be used as an alternative to nonces: the freshness of a message is simply given by the subtraction of the timestamp to the current time. As noted in [95], usage of timestamps result generally in simpler protocols. The main drawback is the requirement of maintaining secure, synchronised distributed timeclocks. Thus timestamps are typically replaced in protocols by a nonce challenge plus a return message.

1.1.3 A malicious agent: the intruder

Although surprisingly small in extent compared to other distributed algorithms, cryptographic protocols are known to be error-prone, and "bugs are routinely found in well-known protocols years after they were first published" [14]. This is due to the obligation to behave robustly in presence of a hostile opponent who might want to act against the security objectives of the jointly interacting entities.

An example of attack

An example that illustrates the difficulty of designing and analysing cryptographic protocols is certainly the Needham-Schroeder protocol [105]. It was believed to be secure during 17 years until Gavin Lowe found a flaw with an automatic tool [92, 93]. We now discuss briefly this historical example following Gavin Lowe's articles.

A cryptographic protocol is defined by the sequence of messages exchanged between the involved entities. It is usually [58] expressed as a linear scenario of message exchanges, called *protocol narrations* [2] or *narrations* in short, that can be interpreted as the intended trace of ideal execution of the protocol.

The Needham-Schoeder protocol can be represented as follows:

1. $A \rightarrow S : A, B$
2. $S \rightarrow A : \text{Enc}_{\text{priv}(k_S)}^a(\text{pub}(k_B), B)$
3. $A \rightarrow B : \text{Enc}_{\text{pub}(k_B)}^a(n_A, A)$
4. $B \rightarrow S : B, A$
5. $S \rightarrow B : \text{Enc}_{\text{priv}(k_S)}^a(\text{pub}(k_A), A)$
6. $B \rightarrow A : \text{Enc}_{\text{pub}(k_A)}^a(n_A, n_B)$
7. $A \rightarrow B : \text{Enc}_{\text{pub}(k_B)}^a n_B$

This protocol involves three entities (A , B and S), called *participants* or *principals* or *agents*. Each line of the narration represents a message exchange between two participants: $A \rightarrow B : M$ should be read as “ A sends the message M to B ”. Messages are either *atomic* (like A , n_A , ...) or can be constructed by *concatenation* (like (A, B) , (n_A, A) , ...) or by ciphering (like $\text{Enc}_{\text{pub}(k_B)}^a(n_A, A)$, ...). The cipher algorithm used in this protocol is a public-key cryptography algorithm (the a in $\text{Enc}_k^a M$ stands for “asymmetric”). A pair (d, e) of public/private key is represented by a key k whose public part is $\text{pub}(k)$ and private part is $\text{priv}(k)$. The meaning of certain part of the messages is implicit. For instance, n_A is meant to be a nonce freshly generated by A . Likewise, k_A represents the pair of public/private key of A .

This protocol describes how an *initiator* A may establish a session with a *responder* B with the help of a *trusted key server* S . In step 1, A sends a message to the server S , requesting B 's public key. In step 2, S delivers the B 's public key along with B 's identity, encrypted using S 's private key (to assure authenticity of this message). In step 3, A seeks to establish a connection with B by selecting a nonce n_A and sending it along with its identity to B , encrypted with B 's public key. When B receives this message, it requests A 's public key to the server S (step 4) who delivers it in step 5. At step 6, B returns the nonce n_A along with a new nonce n_B to A , encrypted with A 's public key. When A receives this message, he returns the nonce n_B to B , encrypted with B 's public key. When receiving message 6, A should be assured he is talking to B since only B can decrypt message

3. Likewise, when B receives message 7, he should be assured that he is talking to A since only A can decrypt message 6.

As pointed out by Lowe, this protocol has a flaw [92]. To explain the attack, we assume that each agent initially knows each other's public key. We thus make abstraction of the key distribution machinery implemented by steps 1, 2, 4 and 5 and focus our attention on the following protocol narration:

$$\begin{aligned} 3. \quad A \rightarrow B &: \text{Enc}_{\text{pub}(k_B)}^a(n_A, A) \\ 6. \quad B \rightarrow A &: \text{Enc}_{\text{pub}(k_A)}^a(n_A, n_B) \\ 7. \quad A \rightarrow B &: \text{Enc}_{\text{pub}(k_B)}^a n_B \end{aligned}$$

The attack on the protocol allows an intruder I to impersonate another agent A to set up a false session with B . The attack involves two simultaneous runs of the protocol: in run 1, A establishes a valid session with I ; in run 2, I impersonates A to establish a fake session with B . In the attack below, we write $I(A)$ for I impersonating A . We also write for instance 1.3 to mean step 3 of run 1. The attack works as follows:

$$\begin{aligned} 1.3. \quad A \rightarrow I &: \text{Enc}_{\text{pub}(k_I)}^a(n_A, A) \\ 2.3. \quad I(A) \rightarrow B &: \text{Enc}_{\text{pub}(k_B)}^a(n_A, A) \\ 2.6. \quad B \rightarrow I(A) &: \text{Enc}_{\text{pub}(k_A)}^a(n_A, n_B) \\ 1.6. \quad I \rightarrow A &: \text{Enc}_{\text{pub}(k_A)}^a(n_A, n_B) \\ 1.7. \quad A \rightarrow I &: \text{Enc}_{\text{pub}(k_I)}^a n_B \\ 2.7. \quad I(A) \rightarrow B &: \text{Enc}_{\text{pub}(k_B)}^a n_B \end{aligned}$$

In step 1.3, A starts to establish a normal session with I , sending a nonce n_A . In step 2.3, I impersonates A and starts a fake session with B by forwarding the nonce n_A learnt in the previous message. In response, B creates a new nonce n_B and tries to return it to A along with n_A . The intruder intercepts this message and forwards it to A (it cannot decrypt this message since it is encrypted with A 's public key) in step 1.6. Then A decrypts this message to obtain n_B and since it is of the expected form, it returns n_B to I in step 1.7. The intruder I can then decrypt this message to get n_B and returns it to B . Hence B believes that A has correctly established a session with it. The consequences are that I may continue to impersonate A to send messages to B during the session, which is bad.

Gavin Lowe proposed a fix to the Needham-Schroeder protocol, which is now known to be the Needham-Schroeder-Lowe protocol. The fix con-

sists in changing step 6 of the protocol into:

$$6. B \rightarrow A : \text{Enc}_{\text{pub}(k_A)}^a(B, n_A, n_B)$$

This fix prevents the intruder from replaying this message in another run of the protocol.

Adversaries

In order to study security properties of cryptographic protocols, we assume the presence of an unauthorised “third” party which is given many names such as *adversary*, *intruder*, *attacker*, *opponent*, ... Thus, it is typically assumed that protocol messages are transmitted over an *unprotected* network.

The first considered intruders were *passive*: they were only able to record communications on the network and analyse the messages to find a breach (for instance decrypt a message with a compromised key).

Nowadays, the attackers considered are *active*. They are able to intercept any message on the network, analyse messages, construct new messages and inject messages.

Classical attacks on protocols include *man-in-the-middle attack* where the intruder imposes himself between the communications between two principals, *replay attack* where information from a previous run of a protocol is used, *oracle* where the intruder tricks an honest agent into inadvertently revealing some information, *interleave* exemplified by Lowe’s attack on Needham-Schroeder protocol.

1.2 Models and techniques

Even if it is possible to devise some prudent engineering practises [10] for designing cryptographic protocols, there is a need of rigorous techniques to verify their correctness. We give below a brief (and thus necessarily not comprehensive) overview of the different approaches and techniques that have been developed during the last decades.

1.2.1 Two views of cryptography

There exist basically two ways to model cryptographic primitives that are used in security protocols:

- The *formal* (or symbolic) approach models cryptographic messages with a term algebra (an algebraic datatype in a programming language). For instance, the language of cryptographic messages we have used so far is defined by:

$$M, N ::= a | \text{pub}(M) | \text{priv}(M) | H(M) | \text{Enc}_N^s M | \text{Enc}_N^a M$$

The security properties of cryptographic primitives are also formally modeled. It is typically assumed that the cryptographic primitives are *perfect*; a ciphered message *cannot* be decrypted, it may only be deciphered with the appropriate decryption key.

Algebraic properties of cryptographic primitives may be simply ignored or might be modelled within an equational theory.

For instance, for the RSA cryptosystem, we have the following property:

$$\text{Enc}_{\text{priv}(k)}^a \text{Enc}_{\text{pub}(k)}^a M = M = \text{Enc}_{\text{pub}(k)}^a \text{Enc}_{\text{priv}(k)}^a M$$

- The *computational* approach opens the “black-boxes” of the formal approach. Cryptographic messages are modelled by string of bits and cryptographic primitives are simply functions handling string of bits.

Security properties are defined in terms of the probability and computational complexity of successful attacks.

The gap between these two views of cryptography is being bridged. In [11], it is shown, under some conditions, that secrecy in the formal model implies secrecy in the computational model. [18] provides a secure implementation of the formal model as an abstract library of cryptographic primitives.

However one should keep in mind that the more abstract the model is, the more flaws go undetected. For instance, [113] shows that a certain protocol [52] is correct whereas [125] shows that this same protocol is incorrect when *exclusive or* is used to cipher messages. The counterpart can be roughly summarised by the slogan: the more abstract the model is, the easier the proofs are.

In the sequel, we adopt the formal view of cryptography.

1.2.2 The Dolev-Yao model

The 1983 paper of Dolev and Yao [68] is probably one of the most influential work in the domain.

- Cryptographic primitives are idealised and assumed to be perfect.
- A basic model for the intruder, which is still very popular, was introduced. The intruder has the control over the network: it can listen, replay, fake, delete, redirect, ... messages limited only by the cryptographic constraints.
- Security of protocols is expressed as a word problem.

1.2.3 BAN logic

The BAN logic [53] is a logic of authentication based on the notion of *beliefs*. The basic idea is to reason about the state of beliefs of the honest principals. BAN logic defines a deduction system that models how the beliefs evolve as new information is received. It has been very successful for finding bugs in protocols but has also its own limit. Thus, as the name of the article suggests, it only deals with authentication properties. This notion of authentication is not well defined and it has appeared that it was actually a rather weak form of authentication as suggests the example of Lowe's attack on the Needham-Schroeder public key protocol. The problem of such logics is that they adopt the point of view of honest principals and the goal is often to show that a certain property holds for a security protocol. We tend to think that the opposite point of view, where instead it is tried to show that there is a flaw in a protocol (and thus adopting the point of view of a dishonest principal), is better suited in the context of security protocols.

Nevertheless, BAN-like logics are interesting because reasoning is usually simple enough to be carried out by hand for small examples. These are also often decidable [104].

1.2.4 Model checking

The model checking approach consists in checking whether a given finite model satisfies a given logical formula.

For instance, Gavin Lowe's approach [93] can be summarised as follows:

- Model a protocol in CSP [83];
- Model the most general intruder that can interact with the system;
- Model the specification of the protocol as a (set of) formula in a temporal logic;

- Check with FDR [122] whether the CSP system satisfies the formula. The result is either *yes* or an attack of the protocol.

1.2.5 The inductive approach

Paulson's *inductive method* [114, 112] is a typical illustration of how theorem provers can help conduct study of cryptographic protocols. Reasoning takes place in the general purpose tool Isabelle [136] and involve inductive definitions and classical proofs by induction. At the core of this study is the *history*, which is roughly the set of emitted messages. A protocol is simply composed of steps that makes evolve the history. Two operators *Synth*, resp. *Analz* compute, given a set of messages, the set of messages obtained by synthesis, resp. analysis. For instance, if $X \in H$ and $K \in H$ then $Enc_K M \in Synth H$. This approach features several important analysis of protocols like Kerberos [23] and SSL/TLS [115] for instance.

Compared to model checking, theorem proving provides a finer analysis, but at a greater cost, because it requires substantial human assistance and effort. Nevertheless, it is viewed as a good complement to model checking, also because it is not restricted by the requirement of a finite-state model.

1.2.6 Rewrite systems, Horn clauses

Several models represent protocols and the attacker as a set of rewrite rules [56, 123]. An interesting variant of this model is Horn clauses, used for instance in [27, 60]. The advantage of the latter is that classical resolution methods can be applied. One difficulty with these two models is the handling of nonces.

1.2.7 Process calculi

Process calculi offer the possibility to describe protocols in a precise algebraic way close to an implementation. For instance, pi calculus [101] has been successfully used to model and study several communication protocols such as the GSM protocols [110]. In the field of security protocols, this approach has first proven its worth with Gavin Lowe's attack on Needham-Schroeder Public Key protocol using CSP for encoding and analysing this protocol [93]. Since then, numerous other calculi have been used for modelling and analysing security protocols. Among them is the

spi calculus [9] which is an extension of the pi calculus including cryptographic primitives.

The spi calculus is a particularly attractive candidate for modelling cryptographic protocols:

1. the underlying pi calculus provides the essence of what is needed to formulate distributed algorithms in a compact form;
2. the *restriction operator* inherited from the pi calculus can be used to model the creation of fresh, unguessable cryptographic keys and can also be used to model the creation of fresh nonces. Moreover, communication of secrets is conveniently modelled by *scope extrusion*, contrasting from CSP;
3. contrary to CSP, unwanted manipulations of cryptographic terms are semantically impossible;
4. there is no need to explicitly model the attacker, contrary to, for instance, the CSP/FDR approach. In the spi calculus approach, the attacker is implicitly present as any environment that can be formulated in the pi calculus;
5. important security objectives can be formulated as equivalences on spi calculus terms, but also standard programming language technology can be exploited, e.g., through the use of static analysis and reachability analysis.

Behavioural equivalences are an essential ingredient of process calculi theory. In [9], Abadi and Gordon suggest to formulate secrecy or authenticity properties as equations between spi calculus terms.

Thus a protocol P maintains secrecy if two instantiations of this protocol, one with a value M and the other with a value M' , are indistinguishable to the environment, i.e. $P(M) \sim P(M')$ for a certain notion of equivalence \sim .

As for authentication, it is asserted that a protocol satisfies the authentication property if $P \sim P_{\text{spec}}$ where P_{spec} is a specification process in which the receiver *magically* knows what message to receive from the transmitter process.

Spi calculus has some derivatives such as, for instance, the pattern-matching spi calculus [78] or LYSA [30]. The underlying idea has been generalised: the applied pi calculus [6] is essentially the pi calculus plus a general notion of terms. The term algebra is equipped with an equational theory to describe the interdependency of various function symbols; it it

thus possible to model “exclusive or” or RSA algebraic properties. The spi calculus can be seen as a fragment of the applied pi calculus that focuses on a particular choice of cryptographic primitives. “Hard-wiring” the constructs and the meaning of terms in the calculus (as in the spi calculus) leads, of course, to a less general framework but this has the advantage to simplify the analysis of the calculus.

1.3 Outline of the thesis

In this thesis, we have chosen the spi calculus framework to model and study cryptographic protocols because we found this approach rather elegant and general as explained previously. We address basically two problems.

Translation of cryptographic protocols The cryptographic protocols are usually described using the informal notation of protocol narrations (see Section 1.1.3). One problem that arises when modelling protocols is to be sure that the model corresponds closely to the studied protocol. Since protocol narrations have no formal meanings (or *semantics*), it is difficult to make the link between the informal notation and the formal model (in spi calculus for instance). To overcome this difficulty, we have defined an explicit operational semantics for protocol narrations and given a rigorous way to translate protocol narrations into spi calculus.

Behavioural equivalences The suggested notion of equivalence for formulating security properties is usually testing equivalence because it intuitively requires the two sides of the equation to behave indifferent in any —potentially hostile— context. However, its verification is rather difficult due to the use of an universal quantification over arbitrary processes. Thus, several *bisimulation* methods [8, 36, 40] have been developed to prove testing equivalence. Although they remove a universal quantification over arbitrary processes and thus are in principle easier to employ than testing equivalence, there is no way to automate proofs using these notions of bisimulation because they are still defined using several levels of quantification over infinite domains. Inspired by the pi calculus framework, where several tools exist to automatically check bisimulation, we propose a new notion of bisimulation for the spi calculus which we argue will be easier to mechanise. This new notion of bisimulation is an extension of open bisimulation [127] of the pi calculus and reuses some ideas of

hedged bisimulation [40] of the spi calculus. The involved definitions and theorems being rather technical and subtle, we have validated part of our work within the `coq` proof assistant.

1.3.1 Contributions

The major contributions of this thesis are listed below.

- We have given a formal semantics for protocol narrations and have defined a rigorous translation of protocol narrations into spi calculus. This procedure has been implemented in the tool `spyer` [44].
- We have proposed two alternative definitions of open bisimulation in the pi calculus that have helped to formulate more precise congruence properties than the ones that were first stated by Sangiorgi in [127].
- Based on our pi calculus alternative definitions of open bisimulation, we have proposed an open-style definition of bisimulation for the spi calculus named *open hedged bisimulation*. We have shown that this notion is an extension of open bisimulation of the pi calculus and that it is a sound proof technique for showing late hedged bisimilarity. We have also given a symbolic characterisation of this new notion of bisimulation, which is a promising step towards its mechanisation.
- We have formalised in the `coq` proof assistant the pi calculus and the spi calculus. We have formalised the theory of *hedges*, which are an essential data structure to reason about the knowledge of the attacker in hedged bisimulations. We have also validated the main results about the various labelled transitions systems involved in our definitions. This provides a framework to reason formally about hedged bisimulations in `coq`.

1.3.2 Overview

We now give a brief overview of the work presented in this thesis.

Chapter 2. We present the pi calculus of Milner, Parrow and Walker. We present its syntax, its semantics and four very popular notions of bisimulation used in the pi calculus: ground, early, late and open bisimulation. We then focus on open bisimulation and argue that it is an attractive candidate notion of bisimulation for the pi calculus. We revisit this notion of

bisimulation and propose two alternative definitions that are better suited for being smoothly extended to the spi calculus framework. Thanks to the more precise structure involved in these alternative definitions, we formulate—and show—more precise congruence properties than the ones that were first stated by Sangiorgi.

This work is a revised version of the workshop paper [48].

Chapter 3. We present a variant of the spi calculus of Abadi and Gordon, which includes shared-key cryptography, public-key cryptography and hashing. We present its syntax and semantics. We discuss how behavioural equivalences can be used to formulate security properties and we motivate the notion of *environment-sensitive* bisimulation.

Chapter 4. We present the structure of hedges, which is a central notion used in hedged bisimulation of Borgström and Nestmann to represent the knowledge of the attacker. We present the basic notions (such as *synthesis*, *analysis*, ...) and results. This part is a generalisation of the result of [40] for a rich language of cryptographic messages.

All the results of this chapter have been formalised in the `coq` proof assistant. A first version of the formalisation has been released as an official `coq` contribution [43]. An improved version is part of our spi formalisation [45].

Chapter 5. We recall the definition of late hedged bisimulation as defined by Borgström and Nestmann. We present our proposal for an open-style definition of bisimulation for the spi calculus: open hedged bisimulation. We show that it is an extension of open bisimulation of the pi calculus. We show that it is a sound proof technique for showing late hedged bisimilarity. We then define a symbolic semantics for the spi calculus and state the symbolic characterisation theorem. We then briefly discuss why it is an important step towards mechanisation of open hedged bisimilarity.

Part of this work is a revised version of the journal paper [50] and the working paper [46]. The symbolic semantics has been inspired by our conference paper [39].

In collaboration with Borgström, we have written a prototype tool [38] for another notion of bisimulation in the spi calculus. The proof of correctness of this tool is still ongoing work but some of the results should be reusable for open hedged bisimulation.

Chapter 6. We quickly sketch the underlying ideas of our coq formalisation of the π and the spi calculus. We discuss in particular the representation of binders. We also discuss the abstractions we have found to limit the amount of work required when defining a new semantics. We then conclude by giving the proof sketches of the most important results of Chapter 3 and Chapter 5.

We have recently made public this formalisation. The details can be found in [45]. The goal of this formalisation is not only to validate our theoretical results but also to be able to eventually extract a certified bisimulation checker.

Chapter 7. We make the formal link between security protocols and the spi calculus in this last chapter. We thus present a formal semantics for protocol narrations and give a precise and rigorous procedure to translate a protocol narration into a spi calculus term.

This work is a revised version of the workshop paper [47] and the journal paper [49].

Chapter 2

The Pi Calculus

We present in this chapter the pi calculus of Milner, Parrow and Walker [101]. We describe its syntax, its labelled late transitions system and several popular notions of bisimilarity; for a more gentle or detailed introduction to the pi calculus, we refer the reader to [101], [100], [111] and [129]. Then, we focus on open bisimilarity and give two alternate definitions, following [48].

2.1 Introduction to the pi calculus

The pi calculus is a mathematical model of communicating and mobile systems. In the late 1970's, Robin Milner introduced CCS (Calculus of Communicating Systems [98]) in order to model concurrent systems. In CCS, a concurrent system is characterised by the communications that can happen between its components. Then, in the late 1980's, Milner, Parrow and Walker introduced the pi calculus, an extension of CCS, as a model for concurrent and mobile systems. In the pi calculus, the notion of link is essential: the location of a process is characterised by the neighbours it is linked to. Since communication links can be transferred between two processes, the neighbourhood of a process, i.e. its location, may change over time.

2.1.1 Syntax

In the pi calculus, links are represented by names. Names are used by processes to interact. A name can be passed by a process to one another; the receiver may use this passed name for further interactions. We shall

assume to have a countably-infinite set of names N ranged over by lower-case letters $a, b, c, \dots, x, y, z, \dots$

Processes of the pi calculus are built upon names. Informally, they are composed of:

- An *empty* process $\mathbf{0}$ that can do nothing.
- A *silent* prefix $\tau.P$. It can evolve invisibly to P .
- An *input* prefix $a(x).P$. It can receive a name z along the *communication channel* a and then continue as P where x is substituted by z .
- An *output* prefix $\bar{a}\langle z \rangle.P$. It can send the name z along the channel a and then continue as P .
- A *match* $[x = y]P$. It can continue as P if and only if the two names x and y are equal. Otherwise, it is stuck.
- A *parallel composition* $P \mid Q$. The processes P and Q can evolve independently or may interact via shared communication channels.
- A *sum* $P + Q$. It can behave either as P or Q . The choice is triggered by the environment or by internal computations of P or Q .
- A *restriction* $(\nu z)P$. A new fresh name z is created, whose scope is restricted to P . It can then evolve as P . The components of P can use the name z for interacting together but not for interacting with other processes: the channel z is private to P . However, since links can be transferred between two processes, the scope of z may vary over time as a result of an interaction between processes: the channel z may become public for some other processes.
- A *replication* $!P$. It can be thought as an infinite parallel composition $P \mid P \mid \dots$.

The syntax of the pi calculus processes P is summarised in Table 2.1.

We have included a sum (or choice) operator as it is sometimes useful for specification purpose.

Some presentations requires that operands of a summation are guarded processes (as e.g. in [129, 99]); we have chosen here to allow arbitrary processes to be sum operands as in the original presentation of the pi calculus [102]. This has the advantage of simplifying the syntax (as noted in [111]) but care should be taken when dealing with weak equivalences.

$P, Q ::=$	$\mathbf{0}$	inactive process
	$\tau.P$	silent prefix
	$a(x).P$	input prefix
	$\bar{a}\langle z \rangle.P$	output prefix
	$[x = y]P$	match
	$P \mid Q$	parallel composition
	$P + Q$	sum
	$(\nu z)P$	restriction
	$!P$	replication

Table 2.1: The processes of the pi calculus

We have preferred the replication operator instead of allowing recursive definitions because it is technically simpler.

Finally, we have not included a *mismatch* $[x \neq y]P$ which can behave like P if and only if the two names x and y are different. The main reason is that it breaks the following monotonicity property: application of a name substitution to a process does not diminish its capabilities for action. This property is useful for reasoning with open bisimilarity.

In the sequel, we will refer to this particular calculus as *the* pi calculus. As already mentioned, it is essentially the same calculus as the one of [102].

We adopt the following precedence among the syntactic forms:

$$\left. \begin{array}{l} \text{Restriction} \\ \text{Silent prefix} \\ \text{Input prefix} \\ \text{Output prefix} \\ \text{Match} \\ \text{Replication} \end{array} \right\} > \text{Parallel composition} > \text{Sum}$$

Thus $(\nu x)P \mid a(x).Q + R$ means $((\nu x)P) \mid (a(x).Q) + R$.

In the following, we use metavariables P, Q or parametrised metavariables $A(x), B(y, z)$ for designing pi calculus processes.

Bound names, free names

In $a(x).P$ or $(\nu x)P$, the name x is said to be *binding* with *scope* P . An occurrence of a name in a process is said to be *bound* if it is, or it lies within the scope of, a binding occurrence of a name. Otherwise, it is said to be *free*. More precisely, the *bound names* $\text{bn}(P)$ and the *free names* $\text{fn}(P)$ of a process P are defined inductively as stated in Table 2.2 and Table 2.3.

$$\begin{aligned}
\text{bn}(\mathbf{0}) &:= \emptyset \\
\text{bn}(\tau.P) &:= \text{bn}(P) \\
\text{bn}(a(x).P) &:= \{x\} \cup \text{bn}(P) \\
\text{bn}(\bar{a}\langle z \rangle.P) &:= \text{bn}(P) \\
\text{bn}([x=y]P) &:= \text{bn}(P) \\
\text{bn}(P \mid Q) &:= \text{bn}(P) \cup \text{bn}(Q) \\
\text{bn}(P + Q) &:= \text{bn}(P) \cup \text{bn}(Q) \\
\text{bn}((\nu z)P) &:= \{z\} \cup \text{bn}(P) \\
\text{bn}(!P) &:= \text{bn}(P)
\end{aligned}$$

Table 2.2: The bound names of a process

$$\begin{aligned}
\text{fn}(\mathbf{0}) &:= \emptyset \\
\text{fn}(\tau.P) &:= \text{fn}(P) \\
\text{fn}(a(x).P) &:= \{a\} \cup (\text{fn}(P) \setminus \{z\}) \\
\text{fn}(\bar{a}\langle z \rangle.P) &:= \{a, z\} \cup \text{fn}(P) \\
\text{fn}([x=y]P) &:= \{x, y\} \cup \text{fn}(P) \\
\text{fn}(P \mid Q) &:= \text{fn}(P) \cup \text{fn}(Q) \\
\text{fn}(P + Q) &:= \text{fn}(P) \cup \text{fn}(Q) \\
\text{fn}((\nu z)P) &:= \text{fn}(P) \setminus \{z\} \\
\text{fn}(!P) &:= \text{fn}(P)
\end{aligned}$$

Table 2.3: The free names of a process

For example, with $P := a(x).(vb) \bar{x}\langle b \rangle . \bar{c}\langle y \rangle . \mathbf{0}$, the bound names of P are $\text{bn}(P) = \{x, b\}$ and its free names are $\text{fn}(P) = \{a, c, y\}$.

We introduce the notion of substitution to express that a process can use names it receives.

Definition 1 (substitution).

A *substitution* σ is a function on names that is the identity except on a finite set.

If $\sigma : N \rightarrow N$ is a substitution, the finite set $\text{supp}(\sigma) := \{x \mid \sigma(x) \neq x\}$ is the *support* of σ and the finite set $\text{cosupp}(\sigma) := \{\sigma(x) \mid x \in \text{supp}(\sigma)\}$ is the *co-support* of σ . The names of σ are $\text{n}(\sigma) := \text{supp}(\sigma) \cup \text{cosupp}(\sigma)$.

We generally use the postfix notation and write $x\sigma$ for σ applied to x . Thus $x\sigma$ means $\sigma(x)$. We write $[y_1/x_1, \dots, y_n/x_n]$ for the substitution σ such that $x_i\sigma = y_i$ for $1 \leq i \leq n$ and $x\sigma = x$ otherwise (i.e. $x \notin \{x_1, \dots, x_n\}$, where x_1, \dots, x_n are assumed to be pairwise distincts). If X is a set of names, we write $X\sigma$ for $\{x\sigma \mid x \in X\}$. We sometimes use ϵ for the substitution that has an empty support.

The next thing to define is the effect of applying a substitution σ to a process P . Intuitively, it replaces each free occurrence of each name x in P by $x\sigma$. However, some care must be taken to avoid unintended name captures by a binder.

To avoid name captures, we identify processes that are said *α -equivalent* (or *α -convertible*). Two processes P and Q are *α -convertible*, $P =_\alpha Q$, if they *only* differ by a change of bound names.

For example, $a(x).(vb) \bar{x}\langle b \rangle . \bar{c}\langle y \rangle . \mathbf{0} =_\alpha a(x).(vz) \bar{x}\langle z \rangle . \bar{c}\langle y \rangle . \mathbf{0}$.

In the following, we identify *α -equivalent* processes. In particular, we assume that in a process bound names are different from each other and from the free names.

We can now define application of a substitution to a process:

Definition 2 (application of substitution).

Substitutions are applied to processes according to the following inductive definition:

$$\begin{aligned}
 (\mathbf{0})\sigma &:= \mathbf{0} \\
 (\tau.P)\sigma &:= \tau.(P\sigma) \\
 (a(x).P)\sigma &:= a\sigma(x).P\sigma && \text{if } x \notin \text{n}(\sigma) \\
 (\bar{a}(z).P)\sigma &:= \bar{a}\sigma(z\sigma).P\sigma \\
 ([x=y]P)\sigma &:= [x\sigma=y\sigma]P\sigma \\
 ((vz)P)\sigma &:= (vz)P\sigma && \text{if } z \notin \text{n}(\sigma) \\
 (P|Q)\sigma &:= P\sigma|Q\sigma \\
 (P+Q)\sigma &:= P\sigma+Q\sigma \\
 (!P)\sigma &:= !P\sigma
 \end{aligned}$$

μ	$\text{fn}(\mu)$	$\text{bn}(\mu)$	$\text{n}(\mu)$	$\mu\sigma$
τ	\emptyset	\emptyset	\emptyset	τ
$\bar{a}u$	$\{a, u\}$	\emptyset	$\{a, u\}$	$\bar{a}\sigma u\sigma$
$a(x)$	$\{a\}$	$\{x\}$	$\{a, x\}$	$a\sigma(x)$
$\bar{a}vx$	$\{a\}$	$\{x\}$	$\{a, x\}$	$\bar{a}\sigma vx$

Table 2.4: Notation for actions

To satisfy the side conditions for input prefix or restriction, we might first need to rename the bound names of P . This is possible because we have identified α -equivalent processes.

For example, consider $P := a(x).(vb)\bar{x}\langle b\rangle.\bar{c}\langle y\rangle.\mathbf{0}$ and $\sigma := \{^b/y\}$.

Then $P\sigma = a(x).(vz)\bar{x}\langle z\rangle.\bar{c}\langle b\rangle.\mathbf{0}$. Note in particular the change of the bound name b to the name z so to avoid the capture of the free name b , resulting from the application of σ to y , by the restriction operator.

2.1.2 Labelled transition system

The observable behaviours of processes are described by a *labelled transition system*. Labelled transitions take the form of $P \xrightarrow{\mu} P'$ for some set of actions ranged over by μ . In the π calculus, there are four kinds of actions:

- the *internal* action τ .
- the (free) *output* action $\bar{a}u$
- the *input* action $a(x)$
- the *bound output* action $\bar{a}vx$

An input transition $P \xrightarrow{a(x)} Q$ means that P can receive some name u along the channel a and then evolve to $Q\{^u/x\}$.

A bound output transition $P \xrightarrow{\bar{a}vx} Q$ (where $x \neq a$) means that P can emit a fresh local name x along the channel a and then evolve to Q .

If μ is an action, we define the set of its free names, its bound names, its names and the effect of applying a substitution to it. This is summarised in Table 2.4.

The labelled transition semantics is given by the rules of Table 2.5 enriched by the symmetric variants of COMM-L, CLOSE-L and SUM-L.

For instance, the symmetric variant of COMM-L is obtained by swapping the roles of P and Q . This gives:

$$\text{COMM-R} \frac{P \xrightarrow{\bar{a}u} P' \quad Q \xrightarrow{a(x)} Q'}{P | Q \xrightarrow{\tau} P' | Q' \{u/x\}}$$

Note that for the reasons explained in [129], we have chosen to have three rules dealing with replication instead of the single rule

$$\text{REP} \frac{P | !P \xrightarrow{\mu} P'}{!P \xrightarrow{\mu} P'}$$

The rule ALPHA allows to α -rename processes so that the side conditions of the CLOSE rules and the PAR rules can be satisfied.

Note also that the bound names of μ when $P \xrightarrow{\mu} Q$ are binding occurrences in Q , thus yielding a notion of α -equivalence for transitions.

An example of a transition derivation

Consider $P := (\nu z) (\bar{a}\langle z \rangle. \mathbf{0} | z(y). \mathbf{0}) | a(x). \bar{x}\langle b \rangle. \mathbf{0}$

P can evolve silently to $Q := (\nu z) ((\mathbf{0} | z(y). \mathbf{0}) | \bar{z}\langle b \rangle. \mathbf{0})$, i.e. we have $P \xrightarrow{\tau} Q$ according to the following derivation:

$$\frac{\frac{\frac{\bar{a}\langle z \rangle. \mathbf{0} \xrightarrow{\bar{a}z} \mathbf{0}}{\bar{a}\langle z \rangle. \mathbf{0} | z(y). \mathbf{0} \xrightarrow{\bar{a}z} \mathbf{0} | z(y). \mathbf{0}}}{(\nu z) (\bar{a}\langle z \rangle. \mathbf{0} | z(y). \mathbf{0}) \xrightarrow{\bar{a}\nu z} \mathbf{0} | z(y). \mathbf{0}} \quad \frac{a(x). \bar{x}\langle b \rangle. \mathbf{0} \xrightarrow{a(x)} \bar{x}\langle b \rangle. \mathbf{0}}{a(x). \bar{x}\langle b \rangle. \mathbf{0} \xrightarrow{a(x)} \bar{x}\langle b \rangle. \mathbf{0}}}{P \xrightarrow{\tau} (\nu z) ((\mathbf{0} | z(y). \mathbf{0}) | \bar{z}\langle b \rangle. \mathbf{0})}}$$

In this example, the transition $(\nu z) (\bar{a}\langle z \rangle. \mathbf{0} | z(y). \mathbf{0}) \xrightarrow{\bar{a}\nu z} \mathbf{0} | z(y). \mathbf{0}$ is derived. Since $(\nu z) (\bar{a}\langle z \rangle. \mathbf{0} | z(y). \mathbf{0}) =_{\alpha} (\nu u) (\bar{a}\langle u \rangle. \mathbf{0} | u(y). \mathbf{0})$, this transition is α -equivalent to $(\nu z) (\bar{a}\langle z \rangle. \mathbf{0} | z(y). \mathbf{0}) \xrightarrow{\bar{a}\nu u} \mathbf{0} | u(y). \mathbf{0}$.

Substitutions and labelled transitions

It is important to understand the link between substitutions and transitions, since behaviour is described with the help of substitutions.

As we announced in the prelude, application of a substitution to a process does not diminish its capabilities for action.

$$\begin{array}{c}
\text{SILENT } \frac{}{\tau.P \xrightarrow{\tau} P} \quad \text{INPUT } \frac{}{a(x).P \xrightarrow{a(x)} P} \quad \text{OUTPUT } \frac{}{\bar{a}\langle u \rangle.P \xrightarrow{\bar{a}u} P} \\
\\
\text{RES } \frac{P \xrightarrow{\mu} P'}{(vz)P \xrightarrow{\mu} (vz)P'} \quad z \notin \text{fn}(\mu) \quad \text{OPEN } \frac{P \xrightarrow{\bar{a}z} P'}{(vz)P \xrightarrow{\bar{a}vz} P'} \quad z \neq a \\
\\
\text{COMM-L } \frac{P \xrightarrow{a(x)} P' \quad Q \xrightarrow{\bar{a}u} Q'}{P|Q \xrightarrow{\tau} P'\{u/x\}|Q'} \\
\\
\text{CLOSE-L } \frac{P \xrightarrow{a(x)} P' \quad Q \xrightarrow{\bar{a}vz} Q'}{P|Q \xrightarrow{\tau} (vz)(P'\{z/x\}|Q')} \quad z \notin \text{fn}(P) \quad \text{MATCH } \frac{P \xrightarrow{\mu} P'}{[x=x]P \xrightarrow{\mu} P'} \\
\\
\text{PAR-L } \frac{P \xrightarrow{\mu} P'}{P|Q \xrightarrow{\mu} P'|Q} \quad \text{bn}(\mu) \cap \text{fn}(Q) = \emptyset \quad \text{SUM-L } \frac{P \xrightarrow{\mu} P'}{P+Q \xrightarrow{\mu} P'} \\
\\
\text{REP-ACT } \frac{P \xrightarrow{\mu} P'}{!P \xrightarrow{\mu} P' | !P} \quad \text{REP-COMM } \frac{P \xrightarrow{a(x)} P' \quad P \xrightarrow{\bar{a}u} P''}{!P \xrightarrow{\tau} (P'\{u/x\}|P'') | !P} \\
\\
\text{REP-CLOSE } \frac{P \xrightarrow{a(x)} P' \quad P \xrightarrow{\bar{a}vz} P''}{!P \xrightarrow{\tau} ((vz)(P'\{z/x\}|P'')) | !P} \quad z \notin \text{fn}(P) \\
\\
\text{ALPHA } \frac{P =_{\alpha} Q \quad Q \xrightarrow{\mu} Q' \quad Q' =_{\alpha} P'}{P \xrightarrow{\mu} P'}
\end{array}$$

Table 2.5: The late semantics of the pi calculus

Lemma 1:

If $P \xrightarrow{\mu} P'$ and $\text{fn}(\sigma) \cap \text{bn}(\mu) = \emptyset$ then $P\sigma \xrightarrow{\mu\sigma} P'\sigma$. \square

There is a converse result when considering injective substitutions on $\text{fn}(P)$.

Lemma 2:

If σ is injective on $\text{fn}(P)$, there exists a bijective substitution θ that agrees with σ on $\text{fn}(P)$ (and so $P\sigma = P\theta$) and for any such θ , we have that if $P\theta \xrightarrow{\mu'} Q$ then $P \xrightarrow{\mu} P'$ with $\mu\theta = \mu'$ and $P'\theta = Q$. \square

Alternative presentation in terms of abstractions and concretions

When dealing with the spi calculus, we use another presentation for the labelled transition system that was introduced by Milner for the polyadic π calculus [99].

This alternative presentation has the advantage to not mix binders between actions and derivatives of transition.

Thus an input transition $P \xrightarrow{a(x)} P'$ is written $P \xrightarrow{a} (x)P'$, where $(x)P'$ is called an *abstraction* and is seen as a function from name to process, an output transition $P \xrightarrow{\bar{a}z} P'$ is written $P \xrightarrow{\bar{a}} \langle z \rangle P'$, a bound output action $P \xrightarrow{\bar{a}vz} P'$ is written $P \xrightarrow{\bar{a}} (vz) \langle z \rangle P'$ where $\langle z \rangle P'$ and $(vz) \langle z \rangle P'$ are called *concretions* and are the counterpart of abstractions.

The administrative work of renaming required by the rules CLOSE and PAR can be separated from the semantics, simplifying greatly its presentation.

Thus the rule CLOSE-L becomes

$$\text{CLOSE-L} \quad \frac{P \xrightarrow{a} F \quad Q \xrightarrow{\bar{a}} C}{P | Q \xrightarrow{\tau} F \bullet C}$$

The cost of this alternative presentation is that several auxiliary operations have to be defined like the *pseudo application* $F \bullet C$ of an abstraction F and a concretion C .

For the monadic π calculus, we think that this alternative presentation was too heavy and prefer to use the classic presentation.

Early semantics

The labelled transition system we have given previously defines the *late* semantics of the π calculus. It is qualified to be late in the sense that the

transmitted name is substituted at the moment of a communication (rules COMM and CLOSE).

However, there is an alternative way to treat the semantics for input. The *early* semantics replaces the rule INPUT by

$$\text{INPUT} \frac{u \in \mathcal{N}}{a(x).P \xrightarrow{au} P\{u/x\}}$$

A new kind of action au , called *free input* action, is used. The input transition $P \xrightarrow{au} Q$ means that P receives the name u along the channel a and continue as Q .

The rules for communication need to be modified since the input action in the premise's prefix refers to the transmitted name. For example, the rule COMM-L becomes

$$\text{COMM-E} \frac{P \xrightarrow{au} P' \quad Q \xrightarrow{\bar{a}u} Q'}{P | Q \xrightarrow{\tau} P' | Q'}$$

This semantics is early in the sense that the transmitted name is substituted at the moment of an input action.

One can show that there is an early input transition $P \xrightarrow{au} P'$ if and only if there is a late input transition $P \xrightarrow{a(x)} P''$ such that $P' = P''\{u/x\}$.

For the other kind of transitions, it can be shown that the two semantics are equivalent; if μ is a non-input action then there is an early transition $P \xrightarrow{\mu} P'$ if and only if there is a late transition $P \xrightarrow{\mu} P'$.

As noted by Parrow in [111], "it could be argued that the early semantics more closely follows an operational intuition since, after all, a [process] performs an input action only when it actually receives a particular value". However, the choice of the late semantics allows a wider spectrum of behavioural equivalences as we shall see in Section 2.2. The late semantics is also better suited for writing automatic tools because of its more symbolic nature.

2.1.3 Structural congruence

Structural congruence is an equivalence relation between processes that allows to manipulate the structure of terms. It aims at identifying processes that intuitively represent the same entity by just looking at their structure. The use of such a relation has been suggested by the Chemical Abstract Machine of Berry and Boudol [25].

We first define the notions of *context* and *congruence*.

Definition 3 (process contexts).

A process context $C[\cdot]$ is a process with a *hole* somewhere. The contexts are described by the grammar of Table 2.6.

$$\begin{array}{l}
C[\cdot] ::= [\cdot] \\
\quad | \tau.C[\cdot] \\
\quad | a(x).C[\cdot] \mid \bar{a}(u).C[\cdot] \\
\quad | [x=y]C[\cdot] \\
\quad | C[\cdot] + P \mid P + C[\cdot] \\
\quad | C[\cdot] \mid P \mid P \mid C[\cdot] \\
\quad | (vz)C[\cdot] \\
\quad | !C[\cdot]
\end{array}$$

Table 2.6: The process contexts of the pi calculus

If $C[\cdot]$ is a context and P is a process, we write $C[P]$ for the process obtained by replacing the hole $[\cdot]$ in $C[\cdot]$ by P . Note that some occurrences of free names of P may be bound in $C[P]$.

For example let $C[\cdot] := a(x).\cdot$ and $P := x(z).\mathbf{0} \mid \bar{y}(y).\mathbf{0}$. Then $C[P] = a(x).(x(z).\mathbf{0} \mid \bar{y}(y).\mathbf{0})$. The name x has been captured by the input prefix of the context $C[\cdot]$.

Definition 4 (congruence).

An equivalence relation \mathcal{R} on processes is a *congruence* if $(P, Q) \in \mathcal{R}$ implies $(C[P], C[Q]) \in \mathcal{R}$ for every context $C[\cdot]$.

We can now define the notion of *structural congruence*.

Definition 5 (structural congruence).

Structural congruence, \equiv , is the smallest congruence on processes that satisfies the axioms of Table 2.7.

For example, we have that $P := (vz)(\bar{a}(z).\mathbf{0} \mid z(y).\mathbf{0}) \mid a(x).\bar{x}(b).\mathbf{0}$ is structurally congruent to $Q := (vz)((\bar{a}(z).\mathbf{0} \mid a(x).\bar{x}(b).\mathbf{0}) \mid z(y).\mathbf{0})$. Note how the scope of z has been extended so that the communicants $\bar{a}(z).\mathbf{0}$ and $a(x).\bar{x}(b).\mathbf{0}$ are now in juxtaposition.

An important result about structural congruence is that it is preserved by the labelled transition system, i.e. the following theorem holds:

Theorem 1:

If $P \equiv Q$ and $P \xrightarrow{\mu} P'$ then there exists Q' such that $Q \xrightarrow{\mu} Q'$ and $P' \equiv Q'$. \diamond

$$\begin{array}{c}
\text{ALPHA } \frac{P =_{\alpha} Q}{P \equiv Q} \quad \text{SUM-ZERO } \frac{}{P + 0 \equiv P} \quad \text{SUM-COMM } \frac{}{P + Q \equiv Q + P} \\
\\
\text{SUM-ASSOC } \frac{}{(P + Q) + R \equiv P + (Q + R)} \quad \text{PAR-ZERO } \frac{}{P | 0 \equiv P} \\
\\
\text{PAR-COMM } \frac{}{P | Q \equiv Q | P} \quad \text{PAR-ASSOC } \frac{}{(P | Q) | R \equiv P | (Q | R)} \\
\\
\text{NEW-SWAP } \frac{}{(\nu x) (\nu y) P \equiv (\nu y) (\nu x) P} \quad \text{NEW-ZERO } \frac{}{(\nu x) \mathbf{0} \equiv \mathbf{0}} \\
\\
\text{NEW-SCOPE } \frac{}{((\nu x) P) | Q \equiv (\nu x) (P | Q)} \quad x \notin \text{fn}(Q) \quad \text{BANG } \frac{}{!P \equiv P | !P} \\
\\
\text{MATCH-EQUAL } \frac{}{[x = x]P \equiv P}
\end{array}$$

Table 2.7: Axioms of structural congruence

The above theorem can be diagrammatically represented by

$$\begin{array}{ccc}
P & \xrightarrow{\mu} & P' \\
\equiv & & \equiv \\
Q & \dashrightarrow^{\mu} & Q'
\end{array}$$

The plain arrow represents a universal quantification over labelled transitions and the dashed arrow represents an existential quantification.

2.2 Several notions of bisimilarity

Hitherto we have given the pi calculus syntax and its labelled late semantics. We have defined structural congruence which identifies processes that intuitively represent the same entity by just looking at their structure. Structural congruence is an interesting notion of equivalence but it is too fine-grained to study the behaviour of terms. To achieve this goal, we define several notions of *behavioural equivalence* based on the notion of *bisimulation*.

2.2.1 Ground bisimulation

A bisimulation can be thought of as a game. It is composed of two players (two processes) P and Q and an external observer who tries to find a difference between P and Q . The observer proceeds as follows: if P is able to perform a move $P \xrightarrow{\mu} P'$ (resp. Q is able to perform a move $Q \xrightarrow{\mu} Q'$) then Q should be able to perform an equivalent move $Q \xrightarrow{\mu} Q'$ (resp. P should be able to perform an equivalent move) so that P' and Q' can also play the game. If at some point in the game, one of the two players cannot play a responding move, then P and Q are declared to not have the same behaviour.

This game can be diagrammatically represented by:

$$\begin{array}{ccc} P & \xrightarrow{\mu} & P' \\ \mathcal{R} & \xrightarrow{\mu} & \mathcal{R} \\ Q & \dashrightarrow & Q' \end{array} \qquad \begin{array}{ccc} Q & \xrightarrow{\mu} & Q' \\ \mathcal{R} & \xrightarrow{\mu} & \mathcal{R} \\ P & \dashrightarrow & P' \end{array}$$

This leads to the following formal definition:

Definition 6 (ground bisimulation).

A symmetric relation $\mathcal{R} \subset P \times P$ is a *ground bisimulation* if for all $(P, Q) \in \mathcal{R}$, whenever $P \xrightarrow{\mu} P'$ and $\text{bn}(\mu) \cap \text{fn}(P + Q) = \emptyset$, there exists Q' such that $Q \xrightarrow{\mu} Q'$ and $(P', Q') \in \mathcal{R}$.

The condition on $\text{bn}(\mu)$ ensures that the bound names of μ are *fresh*.

We say that P and Q are *ground bisimilar*, and write $P \sim_g Q$, if there exists a ground bisimulation \mathcal{R} such that $(P, Q) \in \mathcal{R}$.

In other words, ground bisimilarity \sim_g is the union of all the ground bisimulations. It is itself a ground bisimulation. One can show that \sim_g is an equivalence relation (see [129]).

Ground bisimilarity arguably does not discriminate enough.

Thus, for instance we have

$$x(z).(\bar{z}\langle z \rangle. \mathbf{0} \mid a(x). \mathbf{0}) \sim_g x(z).(\bar{z}\langle z \rangle. a(x). \mathbf{0} + a(x). \bar{z}\langle z \rangle. \mathbf{0})$$

However, the process $x(z).(\bar{z}\langle z \rangle. \mathbf{0} \mid a(x). \mathbf{0})$ may receive the name a along the channel x and evolves to $\bar{a}\langle a \rangle. \mathbf{0} \mid a(x). \mathbf{0}$ which can then perform an internal communication whereas the other process would evolve to $(\bar{a}\langle a \rangle. a(x). \mathbf{0} + a(x). \bar{a}\langle a \rangle. \mathbf{0})$ where no internal communication can happen.

Hence, ground bisimilarity can not be considered as an interesting notion of behavioural equivalence in the pi calculus.

2.2.2 Early bisimulation

The definition of ground bisimulation has been obtained directly by applying the idea of the bisimulation game with the late semantics. However, with the early semantics, this yields a different notion of bisimulation.

Definition 7 (early bisimulation).

A symmetric relation $\mathcal{R} \subset \mathbf{P} \times \mathbf{P}$ is an *early bisimulation* if for all $(P, Q) \in \mathcal{R}$, whenever $P \xrightarrow{\mu} P'$ and $\text{bn}(\mu) \cap \text{fn}(P + Q) = \emptyset$ we have

- if μ is not an input action then there exists Q' such that $Q \xrightarrow{\mu} Q'$ and $(P', Q') \in \mathcal{R}$.
- if μ is an input action $a(x)$ then for all $u \in N$, there exists Q' such that $Q \xrightarrow{\mu} Q'$ and $(P'\{u/x\}, Q'\{u/x\}) \in \mathcal{R}$.

As before, we define the corresponding notion of bisimilarity. Two processes P and Q are *early bisimilar*, written $P \sim_e Q$, if there exists an early bisimulation \mathcal{R} such that $(P, Q) \in \mathcal{R}$.

Note that it is possible to define an equivalent notion of early bisimulation but using the early labelled transition system. In this case, the definition looks the same as the definition of ground bisimulation given above.

Early bisimilarity is known to be an interesting notion of behavioural equivalence.

For instance,

$$x(z).(\bar{z}\langle z \rangle. \mathbf{0} \mid a(x). \mathbf{0}) \not\sim_e x(z).(\bar{z}\langle z \rangle.a(x). \mathbf{0} + a(x).\bar{z}\langle z \rangle. \mathbf{0})$$

This equivalence does not hold essentially for the reason explained previously.

However, we have that

$$\bar{z}\langle z \rangle. \mathbf{0} \mid a(x). \mathbf{0} \sim_e \bar{z}\langle z \rangle.a(x). \mathbf{0} + a(x).\bar{z}\langle z \rangle. \mathbf{0}$$

This example illustrates the interleaving nature of the concurrency in the pi calculus.

It also shows that \sim_e is not closed under input prefix. However, one can show that \sim_e is preserved by all the other operators of the pi calculus. Early bisimilarity is a *non-input congruence*: it is preserved by every process context in which the hole $[\cdot]$ does not occur under an input prefix.

2.2.3 Late bisimulation

In the definition of early bisimulation, the responding move for an input transition depends on the name u chosen in the universal quantification. It is possible to strengthen this definition by requiring that the move is the same for each name u .

This yields to the following definition of bisimulation, which is obtained from the one of early bisimulation by swapping the universal and existential quantification for the input clause.

Definition 8 (late bisimulation).

A symmetric relation $\mathcal{R} \subset P \times P$ is a *late bisimulation* if for all $(P, Q) \in \mathcal{R}$, whenever $P \xrightarrow{\mu} P'$ and $\text{bn}(\mu) \cap \text{fn}(P + Q) = \emptyset$ we have

- if μ is not an input action then there exists Q' such that $Q \xrightarrow{\mu} Q'$ and $(P', Q') \in \mathcal{R}$.
- if μ is an input action $a(x)$ then there exists Q' such that $Q \xrightarrow{\mu} Q'$ and for all $u \in N$, we have $(P'\{u/x\}, Q'\{u/x\}) \in \mathcal{R}$.

The corresponding notion of bisimilarity is defined as previously. P and Q are *late bisimilar*, written $P \sim_1 Q$, if there exists a late bisimulation \mathcal{R} such that $(P, Q) \in \mathcal{R}$.

Clearly, late bisimilarity is a stronger notion of bisimilarity than early bisimilarity because of the logical implication:

$$\exists y \forall x : P(x, y) \implies \forall x \exists y : P(x, y)$$

However, late and early bisimilarity do not coincide. Indeed, let

$$\begin{aligned} P &:= x(z). \mathbf{0} + x(z). \bar{z}\langle z \rangle. \mathbf{0} \\ Q &:= x(z). \mathbf{0} + x(z). \bar{z}\langle z \rangle. \mathbf{0} + x(z). [z = y] \bar{z}\langle z \rangle. \mathbf{0} \end{aligned}$$

One can show that P and Q are early bisimilar, i.e. $P \sim_e Q$.

However, P and Q are not late bisimilar. The reason is that P should respond to the move $Q \xrightarrow{x(z)} [z = y] \bar{z}\langle z \rangle. \mathbf{0}$. If the chosen response is $P \xrightarrow{x(z)} \mathbf{0}$ then the substitution $\{y/z\}$ allows us to distinguish the two processes because $[z = z] \bar{z}\langle z \rangle. \mathbf{0} \xrightarrow{\bar{z}z} \mathbf{0}$ whereas $\mathbf{0}\{z/y\}$ is stuck. Otherwise, if the chosen response is $P \xrightarrow{x(z)} \bar{z}\langle z \rangle. \mathbf{0}$, then the substitution ϵ allow to distinguish the two processes because $\bar{z}\langle z \rangle. \mathbf{0} \xrightarrow{\bar{z}z} \mathbf{0}$ whereas $[z = y] \bar{z}\langle z \rangle. \mathbf{0}$ is stuck.

Like early bisimilarity, late bisimilarity is a non-input congruence. The example we used to show that early bisimilarity is not closed under input prefix works also to show that late bisimilarity is not closed under input prefix.

2.2.4 Open bisimulation

We have seen that neither early bisimilarity nor late bisimilarity are full congruences. Instead, they are only non-input congruences. However, the notion of congruence is important because it says that two processes that are related by a congruence can be plugged interchangeably in any environment without being able to tell the difference.

To remedy this, one can define on top of early (resp. late) bisimilarity the notion of early (resp. late) congruence.

Definition 9 (early congruence, late congruence).

P and Q are *early congruent*, written $P \sim_e^c Q$, if for every substitution σ we have $P\sigma \sim_e Q\sigma$.

P and Q are *late congruent*, written $P \sim_l^c Q$, if for every substitution σ we have $P\sigma \sim_l Q\sigma$.

Early congruence (resp. late congruence) is the largest congruence contained in early bisimilarity (resp. late bisimilarity) (see e.g. [111]).

We have seen previously that

$$\bar{z}\langle z \rangle. \mathbf{0} \mid a(x). \mathbf{0} \sim_e \bar{z}\langle z \rangle. a(x). \mathbf{0} + a(x). \bar{z}\langle z \rangle. \mathbf{0}$$

However these two processes are not early congruent because the substitution $\{a/z\}$ makes them non bisimilar.

Instead, we have that

$$\bar{z}\langle z \rangle. \mathbf{0} \mid a(x). \mathbf{0} \sim_e^c \bar{z}\langle z \rangle. a(x). \mathbf{0} + a(x). \bar{z}\langle z \rangle. \mathbf{0} + [z=a]\tau. \mathbf{0}$$

This equation shows the crucial role played by the match operator in an expansion law that reduces a parallel composition to a sum.

The question that arises now is: is it possible to define a congruence directly through bisimulations instead of requiring bisimilarity under all substitutions of names?

The answer is yes: *open bisimulation* of Sangiorgi [127] fulfils this requirement.

In open bisimulation, the quantification over all substitutions is incorporated in the bisimulation clause, obeying basically to the following diagram:

$$\forall\sigma : \begin{array}{ccc} P\sigma & \xrightarrow{\mu} & P' \\ \mathcal{R} & \mu & \mathcal{R} \\ Q\sigma & \dashrightarrow & Q' \end{array} \quad \forall\sigma : \begin{array}{ccc} Q\sigma & \xrightarrow{\mu} & Q' \\ \mathcal{R} & \mu & \mathcal{R} \\ P\sigma & \dashrightarrow & P' \end{array}$$

However, a quantification over *all* substitutions would be too discriminating. To understand why, consider

$$\begin{aligned} P &:= (\nu z) \bar{a}\langle z \rangle. [a=z] \tau. \mathbf{0} \\ Q &:= (\nu z) \bar{a}\langle z \rangle. \mathbf{0} \end{aligned}$$

Since z is a freshly created name, the match $[a=z]$ can not be satisfied. Thus it is reasonable to equate P and Q .

However, if we stick to the above scheme for defining open bisimilarity, the bound output transition $P \xrightarrow{\bar{a}\nu z} [a=z] \tau. \mathbf{0}$ cannot be simulated by Q because $[a=z] \tau. \mathbf{0}$ and $\mathbf{0}$ are not bisimilar for the substitution $\{z/a\}$.

After the bound output transition, the name z has become free in the derivative; thus being vulnerable to substitutions. But intuitively, the substitutions that are to be considered should not allow to *fuse* a and z .

Distinctions

One way to restrict the set of substitutions considered is to use the auxiliary concept of *distinctions* [102], whose goal is to keep track of inequalities such as $a \neq z$ above. Formally, a distinction is defined as follows.

Definition 10 (distinction).

A distinction D is a finite irreflexive and symmetric relation on names. The set of distinctions is \mathcal{D} .

The names of a distinction D are $n(D) = \{a, b \mid (a, b) \in D\}$

A substitution σ is said to respect a distinction D if it does not fuse any pair of D .

Definition 11 (respectful substitution).

Let D be a distinction and σ a substitution. We say that σ respects D , and write $\sigma \triangleright D$, if $\forall (x, y) \in D : x\sigma \neq y\sigma$.

When D is a distinction and σ is a substitution, we write $D\sigma$ for the set $\{(x\sigma, y\sigma) \mid (x, y) \in D\}$. Clearly, if σ respects D then $D\sigma$ is a distinction.

We introduce some further notations for dealing with distinctions.

When A and B are two finite sets of names, we write $A \otimes B$ for the distinction defined by $\{(a, b), (b, a) \mid a \in A \wedge b \in B \wedge a \neq b\}$.

When C is a finite set of names, we write C^\neq for $C \otimes C$.

When D is a distinction, we write $D - x$ for $D \setminus (\{x\} \times N \cup N \times \{x\})$; it removes in D all the pairs containing x .

When D is distinction and M is a set of names, we write $D \upharpoonright M$ for $D \cap (M \times M) = D \cap M^2$.

Open bisimulation

An open bisimulation is a distinction-indexed family of binary relations on processes. Instead of using the classical formulation in terms of indexed family, we introduce the notion of D-relation.

A D-relation is a relation on $\mathcal{D} \times \mathbf{P} \times \mathbf{P}$. We say that a D-relation \mathcal{R} is *symmetric* if for all $(D, P, Q) \in \mathcal{R}$ we have $(D, Q, P) \in \mathcal{R}$.

We now define formally what an open bisimulation is.

Definition 12 (open bisimulation).

A symmetric D-relation \mathcal{R} is an *open bisimulation* if for all $(D, P, Q) \in \mathcal{R}$ and for every substitution σ that respects D , whenever $P\sigma \xrightarrow{\mu} P'$ and $\text{bn}(\mu) \cap (\text{fn}(P+Q) \cup \text{n}(D) \cup \text{n}(\sigma)) = \emptyset$ we have

- if μ is not a bound output action then there exists Q' such that $Q\sigma \xrightarrow{\mu} Q'$ and $(D\sigma, P', Q') \in \mathcal{R}$.
- if μ is a bound output action $\bar{a}vz$ then there exists Q' such that $Q\sigma \xrightarrow{\mu} Q'$ and $(D\sigma \cup (\{z\} \otimes (\text{fn}(P\sigma + Q\sigma) \cup \text{n}(D\sigma))), P', Q') \in \mathcal{R}$.

P and Q are *open D-bisimilar*, written $P \sim_{\circ}^D Q$, if there exists an open bisimulation \mathcal{R} such that $(D, P, Q) \in \mathcal{R}$. When $D = \emptyset$, we simply say that P and Q are *open bisimilar* and write $P \sim_{\circ} Q$.

In the definition of open bisimulation, observe that after any action but a bound output, the updated distinction requires that the substituted names remain distinct. In addition, after a bound output, the fresh name z is required to be distinct from every other free name currently mentioned.

Open D-bisimilarities can be related by the following lemma [127].

Lemma 3:

If $D' \subseteq D$, then $P \sim_{\circ}^{D'} Q$ implies $P \sim_{\circ}^D Q$. □

Since substitutions affects only the free names of a process, we also have that $P \sim_{\circ}^D Q$ implies $P \sim_{\circ}^{D \upharpoonright \text{fn}(P+Q)} Q$ (see [141]).

As a useful proof technique, we give an alternative notion of open bisimulation that induces the same notion of bisimilarity. It explicates the fact that the only names that are important in the distinctions are the free names of the process, as noted in [129].

Definition 13.

A symmetric D-relation \mathcal{R} is an *open' bisimulation* if for all $(D, P, Q) \in \mathcal{R}$ and for every substitution σ that respects D , whenever $P\sigma \xrightarrow{\mu} P'$ and $\text{bn}(\mu) \cap (\text{fn}(P + Q) \cup \text{n}(D) \cup \text{n}(\sigma)) = \emptyset$ we have

- if μ is not a bound output action then there exists Q' such that $Q\sigma \xrightarrow{\mu} Q'$ and $(D\sigma, P', Q') \in \mathcal{R}$.
- if μ is a bound output action $\bar{a}vz$ then there exist Q' and a finite set X such that $Q\sigma \xrightarrow{\mu} Q'$ and $(D\sigma \cup (\{z\} \otimes X\sigma), P', Q') \in \mathcal{R}$ with $\text{fn}(P + Q) \subseteq X$.

P and Q are *open' D-bisimilar*, written $P \sim_{\sigma}^D Q$ if there exists an open' bisimulation \mathcal{R} such that $(D, P, Q) \in \mathcal{R}$.

Lemma 4:

We have $P \sim_{\sigma}^D Q \iff P \sim_{\sigma}^D Q$.

PROOF

Clearly $P \sim_{\sigma}^D Q$ implies $P \sim_{\sigma}^D Q$ since an open bisimulation is also an open' bisimulation.

The other implication involves further work. The sketch of the proof is the following:

- \sim_{σ}^D is closed under bijective substitutions, i.e. if $P \sim_{\sigma}^D Q$ and θ is bijective then $P\theta \sim_{\sigma\theta}^D Q\theta$.
- If $A \subseteq N$ is finite, $D \upharpoonright A \subseteq D'$ and $\sigma' \triangleright D'$ then there exists σ that agrees with σ' on A such that $\sigma \triangleright D$ and $x\sigma \in A\sigma' \iff x \in A$. Moreover, for all such σ , we have $D\sigma \upharpoonright A\sigma' \subseteq D'\sigma'$.

Let $X := (\text{n}(D) \cup A\sigma') \setminus A$.

Since X is finite, there exists Y and a bijective substitution $\theta : X \rightarrow Y$ such that $Y \cap A\sigma' = \emptyset$.

Define σ by $x\sigma = x\sigma'$ if $x \in A$, $x\sigma = x\theta$ if $x \in X$ and otherwise, if $x \notin A \cup \text{n}(D) \cup A\sigma'$, $x\sigma = x$.

Clearly, σ agrees with σ' on A .

Moreover, we have $\sigma \triangleright D$. Indeed, let $(x, y) \in D$. If $x \in A$ and $y \in A$, then $(x, y) \in D'$, $x\sigma = x\sigma'$ and $y\sigma = y\sigma'$. Since $\sigma' \triangleright D'$, we have $x\sigma \neq y\sigma$. If $x \in A$ and $y \notin A$, then by definition $y\sigma = y\theta \in Y$ and $Y \cap A\sigma = \emptyset$ so $y\sigma \neq x\sigma$. If $x \notin A$ and $y \notin A$, then $x\sigma = x\theta$ and $y\sigma = y\theta$. Since θ is bijective and $x \neq y$, we have $x\sigma \neq y\sigma$.

Let x such that $x\sigma \in A\sigma'$. By definition, necessarily, $x \in A$. Conversely, if $x \in A$, then $x\sigma \in A\sigma = A\sigma'$.

We now show that $D\sigma[A\sigma \subseteq D'\sigma']$.

Let $(x, y) \in D\sigma[A\sigma$. There exists $(u, v) \in D$ such that $(x, y) = (u\sigma, v\sigma)$. Moreover, $x \in A\sigma$ and $y \in A\sigma$. Thus $u\sigma \in A\sigma$ and $v\sigma \in A\sigma$. So $u \in A$ and $v \in A$ because $A\sigma = A\sigma'$ and by definition of σ . So $(u, v) \in D[A$. Since $D[A \subseteq D'$, we have $(u, v) \in D'$. Since $u \in A$ and σ and σ' agrees on A , we have $u\sigma = u\sigma'$. Similarly $v\sigma = v\sigma'$. Hence $(x, y) \in D'\sigma'$.

Then, it is easy to show that

$$\left\{ (D', P, Q) \mid P \sim_o^D Q \text{ and } D[\text{fn}(P+Q) \subseteq D'] \right\}$$

is an open bisimulation. ■

Congruence properties of open bisimulation

As announced, open bisimilarity is a full congruence. More precisely, open D -bisimilarity is a D -congruence; that is a relation that is preserved by every context that respects the distinction D . A context $C[\cdot]$ *respects* a distinction D if the occurrence of the hole is not underneath an input prefix binding a name in D .

For example, for $D := \{(x, y), (y, x)\}$, the context $C[\cdot] := a(x).C[\cdot]$ does not respect D because the name x is bound in the hole by an input prefix whereas the context $C'[\cdot] := a(z).[\cdot]$ respects D .

Actually, the following proposition summarises the congruence properties of open D -bisimilarity [127].

Proposition 1:

If $P \sim_o^D Q$ then

1. $\tau.P \sim_o^D \tau.Q$
2. $P \mid R \sim_o^D Q \mid R$ and $R \mid P \sim_o^D R \mid Q$

3. $P + R \sim_o^D Q + R$ and $R + P \sim_o^D R + Q$
4. $!P \sim_o^D !Q$
5. $[x=y]P \sim_o^D [x=y]Q$
6. $\bar{a}(z).P \sim_o^D \bar{a}(z).Q$
7. $(\nu z)P \sim_o^{D-z} (\nu z)Q$
8. if $x \notin n(D)$ then $a(x).P \sim_o^D a(x).Q$ ◇

The lazy flavour of open bisimulation

Open bisimilarity implies late congruence, i.e. we have

$$P \sim_o Q \implies P \sim_1^C Q$$

However, this inclusion is strict. To see why, consider P and Q (taken from [127]) defined by

$$\begin{aligned} P &:= c(x).(\tau.\mathbf{0} + \tau.\tau.\mathbf{0}) \\ Q &:= c(x).(\tau.\mathbf{0} + \tau.\tau.\mathbf{0} + \tau.[x=y]\tau.\mathbf{0}) \end{aligned}$$

Then $P \sim_1^C Q$. Indeed, for every substitution σ , after the initial input action, $Q\sigma$'s third summand becomes equal to $P\sigma$'s first summand or $P\sigma$'s second summand.

On the contrary, we have $P \not\sim_o Q$. Indeed, the instantiation of x can be *delayed* until it is used in the match $[x=y]$, reminiscent to a call-by-need style. Thus, the action $Q \xrightarrow{a(x)} \tau.[x=y]\tau.\mathbf{0}$ can be mimicked neither by $P \xrightarrow{a(x)} \tau.\mathbf{0}$ nor by $P \xrightarrow{a(x)} \tau.\mathbf{0}$ because in the first case we have $([x=y]\tau.\mathbf{0})\{y/x\} \xrightarrow{\tau} \mathbf{0}$ whereas $\mathbf{0}\{y/x\} \not\xrightarrow{\tau}$ and in the second case we have $[x=y]\tau.\mathbf{0} \not\xrightarrow{\tau}$ whereas $\tau.\mathbf{0} \xrightarrow{\tau} \mathbf{0}$.

We recapitulate in Table 2.8 the relationships among the several notions of equivalence we have seen so far. An arrow means strict inclusion and when it goes from left to right it also means "largest congruence in".

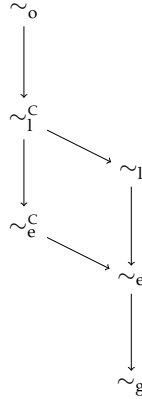


Table 2.8: Relationships among equivalences

An efficient symbolic characterisation

Open bisimilarity is attractive because it enjoys properties of congruence. However, the definition of open bisimulation may seem difficult to check in practise because of the infinite quantification over all respectful substitutions. This potential complexity actually exists only seemingly because it is possible to give an alternative definition of open bisimulation that removes this quantification. This alternative definition, called *symbolic characterisation*, has been proven useful for automating open bisimulation [141, 142, 42] on finite control pi calculus.

The symbolic characterisation relies on the definition of an alternative labelled transition system called the *symbolic open transition system*. The idea is to collect matches along the derivation rules so that it characterises the minimal substitution to be investigated for each transition. The essence of the symbolic open transition system is captured by the following communication rule, where M and N stand for *match sequences*.

$$\text{O-COMM-L} \frac{P \xrightarrow[M]{a(x)} P' \quad Q \xrightarrow[N]{\bar{b}u} Q'}{P \mid Q \xrightarrow[MN[a=b]]{\tau} P'\{u/x\} \mid Q'}$$

Then, a symbolic open bisimulation is defined in terms of the symbolic

open transition system. It resembles the definition of open bisimulation but instead of quantifying over all substitutions, it checks only for the *most general unifier* of each symbolic open transition $P \xrightarrow[M]{\mu} P'$. The proof of correspondence uses the result of Lemma 1.

We refer the reader to [127] for a more detailed presentation of the symbolic characterisation.

2.2.5 A word on weak equivalences

Hitherto, we have dealt with behavioural equivalences that treat visible actions and internal actions equally. These notions of equivalences are usually called *strong equivalences* in contrast to *weak equivalences* that abstract on internal actions, which is of practical use for applying pi calculus.

Weak bisimilarities use a different notion of moves, that essentially ignores τ transitions.

Definition 14 (weak transition relations).

1. \Longrightarrow is the reflexive and transitive closure of $\xrightarrow{\tau}$.
2. If μ is an action, $\xRightarrow{\mu}$ is $\Longrightarrow \xrightarrow{\mu} \Longrightarrow$.

We write $\xRightarrow{\hat{\mu}}$ for $\xRightarrow{\mu}$ if $\mu \neq \tau$ and for \Longrightarrow if $\mu = \tau$.

The intention is then to replace $\xrightarrow{\mu}$ by $\xRightarrow{\hat{\mu}}$ in the definitions of bisimulation.

However, a special treatment is required for τ transitions because they cannot be simply ignored. The reason is, as noted in [129], that τ actions can preempt other actions. For instance, the two processes $x(z).\mathbf{0} + y(z).\mathbf{0}$ and $\tau.x(z).\mathbf{0} + \tau.y(z).\mathbf{0}$ are not behaviourally equivalent.

Moreover, it is not necessary that the first player's moves involve more than single transitions due to the recursive nature of bisimulation. This observation makes easier proofs of weak equivalences.

Hence, a weak ground bisimulation is defined as follows.

Definition 15 (weak ground bisimulation).

A symmetric relation $\mathcal{R} \subset P \times P$ is a *weak ground bisimulation* if for all $(P, Q) \in \mathcal{R}$, whenever $P \xrightarrow{\mu} P'$ and $\text{bn}(\mu) \cap \text{fn}(P + Q) = \emptyset$, there exists Q' such that $Q \xRightarrow{\hat{\mu}} Q'$ and $(P', Q') \in \mathcal{R}$.

The formulation of weak early bisimulation is similar to the previous definition when using early semantics. With late semantics, input actions need a special treatment.

Definition 16 (weak early bisimulation).

A symmetric relation $\mathcal{R} \subset P \times P$ is a *weak early bisimulation* if for all $(P, Q) \in \mathcal{R}$, whenever $P \xrightarrow{\mu} P'$ and $\text{bn}(\mu) \cap \text{fn}(P + Q) = \emptyset$ we have

- if μ is not an input action then there exists Q' such that $Q \xRightarrow{\hat{\mu}} Q'$ and $(P', Q') \in \mathcal{R}$.
- if μ is an input action $a(x)$ then for all $u \in N$, there exists Q' and Q'' such that $Q \xRightarrow{\mu} Q'$, $Q'\{u/x\} \xRightarrow{\mu} Q''$ and $(P'\{u/x\}, Q'') \in \mathcal{R}$.

There are two plausible definitions for weak late bisimulation, as noted in [129], the most reasonable being the following.

Definition 17 (weak late bisimulation).

A symmetric relation $\mathcal{R} \subset P \times P$ is a *weak late bisimulation* if for all $(P, Q) \in \mathcal{R}$, whenever $P \xrightarrow{\mu} P'$ and $\text{bn}(\mu) \cap \text{fn}(P + Q) = \emptyset$ we have

- if μ is not an input action then there exists Q' such that $Q \xRightarrow{\hat{\mu}} Q'$ and $(P', Q') \in \mathcal{R}$.
- if μ is an input action $a(x)$ then there exists Q' such that $Q \xRightarrow{\mu} Q'$ and for all $u \in N$, there exists Q'' such that $Q'\{u/x\} \xRightarrow{\mu} Q''$ and $(P'\{u/x\}, Q'') \in \mathcal{R}$.

In contrast, the definition of weak open bisimulation is more regular.

Definition 18 (weak open bisimulation).

A symmetric D-relation \mathcal{R} is a *weak open bisimulation* if for all $(D, P, Q) \in \mathcal{R}$ and for every substitution σ that respects D , whenever $P\sigma \xrightarrow{\mu} P'$ and $\text{bn}(\mu) \cap (\text{fn}(P + Q) \cup \text{n}(D) \cup \text{n}(\sigma)) = \emptyset$ we have

- if μ is not a bound output action then there is Q' such that $Q\sigma \xRightarrow{\hat{\mu}} Q'$ and $(D\sigma, P', Q') \in \mathcal{R}$.
- if μ is a bound output action $\bar{a}vz$ then there is Q' such that $Q\sigma \xRightarrow{\hat{\mu}} Q'$ and $(D\sigma \cup (\{z\} \otimes (\text{fn}(P\sigma + Q\sigma) \cup D\sigma)), P', Q') \in \mathcal{R}$.

In the rest of this document, we mainly focus on strong equivalences because (1) the theory is generally simpler for the strong case, (2) the theory can be smoothly adapted to the weak case because in many respects, strong and weak cases only differ in details, and (3) they are often a useful proof technique for weak equivalences. However, it is important to note that the goal is weak equivalence.

2.3 Open bisimulation, revisited

As the previous section suggests, our favourite notion of behavioural equivalence for the pi calculus is open bisimulation because of its congruence properties and its easy implementability. For this reason, we are interested in generalising this notion to the case of the spi calculus, which is an extension of the pi calculus presented in the next chapter. To make this generalisation possible, we had to revisit open bisimulation of the pi calculus. The following developments were discussed in details in [48, 50].

2.3.1 A type-aware variant of open bisimulation

As we saw, substitutions are at the core of many notions of bisimulations for the pi calculus. This is due to the different treatments of simulated *symbolic* input transitions, e.g., when

$$\text{simulating } P \xrightarrow{a(x)} P' \quad \text{by} \quad Q \xrightarrow{a(x)} Q'.$$

The problem is that after the execution of a symbolic input on channel a , the “input variable” name x becomes free in the resulting continuation processes P' and Q' . Considering all possible instantiations of this name x by received name messages can be done either not at all (as in ground), or (as in early) before the simulating transition is chosen, or (as in late) right afterwards—or (as in open) considering all possible substitutions (not only affecting the just freed input variable) even before starting any bisimulation game. As we previously noticed, the latter case can also be seen as “very late” or “lazy” since all possible instantiations of the input variable will be checked the next time we try to continue with the bisimulation game with P' and Q' .

For the sake of clarity, when talking about a substitution $\{M/x\}$, let us use the terms *substitution subject* for x and *substitution object* for M .

What do we actually mean when we require *all possible instantiations* in a bisimulation game? More precisely: which set of substitutions shall be considered, and how do we characterise it? In other words: which entities are admissible as substitution subjects and objects, respectively?

By definition, only free names can ever be affected as substitution subjects. In a process, there are three kinds of free names. A free name may be free because:

1. either it was already initially free,

2. or it has become free after having done an input (or been substituted),
3. or it has become free after having been created as a local name, and afterwards output to some observing process.

In contrast to Sangiorgi [127], we argue that names of the latter kind are constant, i.e., they should not be considered as substitution *subjects*, because they were created freshly and thus appropriately chosen. In contrast, the first two kinds shall be considered.

This idea is implemented in the following notion of F-open bisimulation, which we qualify as *type-aware*. In our context, the word *type* does not refer to the type of names of some typed pi calculus but rather to the type of the substitution function. In lack of a better word, we may also have used the term *syntax-aware*.

The simple idea of F-open bisimulation is to prevent names that were previously (in the course of a bisimulation game) created freshly from being considered as permissible substitution subjects.

The knowledgeable reader may be reminded of the notion of *quasi-open* bisimulation, proposed by Sangiorgi and Walker [126], and later on revisited by Fu [72]. There, the use of distinctions as environments was adapted to the use of a simple set of names that were once freshly created and therefore deemed to remain constant. The resulting quasi-open bisimulation was recognised as being strictly weaker than open bisimulation. Sangiorgi and Walker intuitively summarised this difference as: “*In open bisimilarity, when a name z is sent in a bound-output action, the distinction is enlarged to ensure that z is never identified with any name that is free in the processes that send it. In quasi-open bisimilarity, in contrast, at no point after the scope of z is extruded can a substitution be applied that identifies z with any other name.*” [126].

Like quasi-open bisimulation, the following definition also explicitly keeps track of previously freshly created names. However, it does not use this information to prevent the fusion of such fresh names like quasi-open bisimulation does. It only uses this information to implement the idea that fresh names can be considered as constant names once chosen, such that they should afterwards never be used as substitution subjects. In fact, Lemma 6 and Lemma 7 show that this change still faithfully retains the equational power of open bisimulation.

Definition 19 (F-environment).

The pair (D, C) where D is a distinction and C is a finite subset of names is a *F-environment* if $C \not\subseteq D$. The set of all F-environments is written \mathcal{F} .

The distinction D plays the same role as in open bisimulation, while the set C indicates which names can be considered as constant names. It is used to refine the notion of respectfulness, as follows.

Definition 20 (respectful substitution).

A substitution σ *respects* a F-environment (D, C) , written $\sigma \blacktriangleright (D, C)$, if $\sigma \triangleright D$ and $\text{supp}(\sigma) \cap C = \emptyset$.

The following lemma states the link between the two previously seen notions of respectfulness.

Lemma 5:

Let (D, C) a F-environment and σ a substitution such that $\sigma \triangleright D$.

Then there exists a substitution σ' and a bijective substitution θ such that $\sigma' \blacktriangleright (D, C)$, $\sigma = \sigma'\theta$ and $\text{n}(\theta) \subseteq C \cup C\sigma$.

PROOF

We first prove that σ is injective on the finite set C .

Indeed, let $x, y \in C$ such that $x \neq y$. Since $C^\neq \subseteq D$, we have $(x, y) \in D$. Moreover, we have $\sigma \triangleright D$, so we have $x\sigma \neq y\sigma$. This proves that σ is injective on C .

According to Lemma 1.4.11 of [129], there exists a bijective substitution θ such that σ and θ agree on C . By construction, we also have that $\text{n}(\theta) \subseteq C \cup C\sigma$.

Let $\sigma' = \sigma\theta^{-1}$. Then σ' is a substitution such that $\sigma = \sigma'\theta$.

We now show that $\sigma' \blacktriangleright (D, C)$.

We have that $\sigma' \triangleright D$. Indeed, let $x, y \in D$. Since $\sigma \triangleright D$, we have that $x\sigma \neq y\sigma$. Now, since θ^{-1} is bijective, we get $x\sigma\theta^{-1} \neq y\sigma\theta^{-1}$, hence $x\sigma' \neq y\sigma'$ and $\sigma' \triangleright D$.

Moreover, we have $\text{supp}(\sigma') \cap C = \emptyset$. Indeed, let $x \in C$. Since σ and θ agree on C , we have $x\sigma = x\theta$. So $x\sigma' = x\sigma\theta^{-1} = x\theta\theta^{-1} = x$ and $x \notin \text{supp}(\sigma')$. Hence $\text{supp}(\sigma') \cap C = \emptyset$.

So $\sigma' \blacktriangleright (D, C)$. ■

Definition 21 (F-relation).

A F-relation \mathcal{R} is a subset of $\mathcal{F} \times \mathcal{P} \times \mathcal{P}$.

It is *symmetric* if for all $((D, C), P, Q) \in \mathcal{R}$, we have $((D, C), Q, P) \in \mathcal{R}$.

Definition 22 (F-open bisimulation).

A symmetric F-relation \mathcal{R} is a F-open bisimulation if for all $((D, C), P, Q) \in \mathcal{R}$ and for every substitution σ that respects (D, C) , whenever $P\sigma \xrightarrow{\mu} P'$ and $\text{bn}(\mu) \cap (\text{fn}(P + Q) \cup \text{n}(D) \cup \text{n}(\sigma)) = \emptyset$ we have

- if μ is not a bound output action then there exists Q' such that $Q\sigma \xrightarrow{\mu} Q'$ and $((D\sigma, C), P', Q') \in \mathcal{R}$.
- if μ is a bound output action $\bar{a}vz$ then there exists Q' such that $Q\sigma \xrightarrow{\mu} Q'$ and $((D\sigma \cup (\{z\} \otimes (\text{fn}(P\sigma + Q\sigma) \cup D\sigma)), C \cup \{z\}), P', Q') \in \mathcal{R}$.

The only two differences compared to open bisimulation are, first, that the notion of respectfulness is slightly modified such that it takes into account the constant names of a F-environment and, second, that the extruded names are being accumulated in the pool of constant names of F-environments.

Note that in the above definition, since $C^\neq \subseteq D$ and $\text{supp}(\sigma) \cap C = \emptyset$, we have $C^\neq \subseteq D\sigma$. Moreover, on bound actions, since $C \subseteq n(D\sigma)$, we have $(C \cup \{z\})^\neq \subseteq D\sigma \cup (\{z\} \otimes (\text{fn}(P\sigma + Q\sigma) \cup D\sigma))$. Note also that the freshness condition does not mention C because by assumption $C \subseteq n(D)$.

P and Q are F-open (D, C) -bisimilar, written $P \sim_{\text{F}}^{(D, C)} Q$, if there is a F-open bisimulation \mathcal{R} such that $((D, C), P, Q) \in \mathcal{R}$.

In the same spirit as Lemma 4, it is possible to define a F-open' bisimulation that yields the same notion of bisimilarity. In this case, X should include $C \cup \text{fn}(P + Q)$.

Open and F-open bisimilarity are equivalent in the following sense, as expressed by the combination of the statements of the Lemma 6 and Lemma 7.

Lemma 6:

If $P \sim_{\text{F}}^{(D, C)} Q$, then $P \sim_o^D Q$.

PROOF

Let \mathcal{R} be a F-open bisimulation such that $((D, C), P, Q) \in \mathcal{R}$.

Let $\mathcal{R}' := \{(D, P, Q) \mid ((D, C), P, Q) \in \mathcal{R}\}$.

Then \mathcal{R}' is an open bisimulation up to bijective substitutions (see [128] for more details about up to techniques).

Indeed, let $(D, P, Q) \in \mathcal{R}'$. There is C such that $((D, C), P, Q) \in \mathcal{R}$.

Let σ such that $\sigma \triangleright D$.

By Lemma 5, there exists a substitution σ' and a bijective substitution θ such that $\sigma = \sigma'\theta$ and $\sigma' \blacktriangleright (D, C)$ with $n(\theta) \subseteq C \cup C\sigma$.

We only show how bound output actions are mimicked since the other cases are similar but simpler.

Assume that $P\sigma \xrightarrow{\bar{a}vz} P'$ with $z \notin (\text{fn}(P + Q) \cup n(D) \cup n(\sigma))$.

Since $\sigma = \sigma'\theta$, we have $P\sigma'\theta \xrightarrow{\bar{a}vz} P'$. Since $z \notin n(D)$, we have $z \notin C$. Moreover we have $z \notin n(\sigma)$. So $z \notin C \cup C\sigma$. Thus $z \notin n(\theta)$.

Hence, we have $P\sigma' \xrightarrow{a\theta^{-1}vz} P'' := P'\theta^{-1}$ because θ is bijective.

Since \mathcal{R} is a F-open bisimulation, $\sigma' \blacktriangleright (D, C)$ and $z \notin (\text{fn}(P + Q) \cup \text{n}(D) \cup \text{n}(\sigma))$, there exists Q'' such that $Q\sigma' \xrightarrow{a\theta^{-1}vz} Q''$ and $((D'', C \cup \{z\}), P'', Q'') \in \mathcal{R}$, with, by definition, $D'' := D\sigma' \cup (\{z\} \otimes (\text{fn}(P\sigma' + Q\sigma') \cup D\sigma'))$. Thus $(D'', P'', Q'') \in \mathcal{R}'$.

Since θ is bijective, we have $Q\sigma = Q\sigma'\theta \xrightarrow{\bar{a}vz} Q' := Q''\theta$.

Let $D' = D\sigma \cup (\{z\} \otimes (\text{fn}(P\sigma + Q\sigma) \cup D\sigma))$. Clearly, since $\sigma = \sigma'\theta$ and θ is bijective, we have $D' = D''\theta$.

Thus, we have $(D'\theta^{-1}, P'\theta^{-1}, Q'\theta^{-1}) = (D'', P'', Q'') \in \mathcal{R}'$ with θ^{-1} being a bijective substitution.

So \mathcal{R}' is an open bisimulation up to bijective substitutions.

Hence the result. \blacksquare

Lemma 7:

If $P \sim_{\sigma}^D Q$ and $C \neq \subseteq D$ then $P \sim_{\text{F}}^{(D,C)} Q$.

PROOF

Since $\sigma \blacktriangleright (D, C)$ implies $\sigma \triangleright D$. \blacksquare

2.3.2 A knowledge-aware variant of open bisimulation

We have argued that not all substitution subjects shall be considered and defined on this idea a type-aware variant of open bisimulation. On the other hand, also not all substitution *objects* may be acceptable. More precisely: depending on the history of the ongoing bisimulation game, certain instantiations may sometimes be forbidden. There may be two different reasons for this.

The first reason concerns names of kind (1) or (2) (see p.49), say a , that were free in a process *before* another name, say b , got freshly created and extruded. Due to the freshness property, any subsequent substitution for subject a must not mention b as substitution object, so not to retrospectively invalidate this freshness property. In Sangiorgi's open bisimulation, distinctions precisely keep track of inequalities like $a \neq b$, as required above. In analogy to type-awareness, we may use the term *freshness-awareness* to characterise bisimulations using this sort of substitutions.

The second reason concerns only names of kind (2) and resides on the intuition that substitution objects represent messages that may be sent from the observer to the observed process. In the pi calculus, there is no

limitation beyond distinctions: the observer may send any name that it may have received earlier, or it may simply invent names on its own.

We implement this latter idea by defining a bisimulation that makes explicit an observer who plays against the two players P and Q involved in the bisimulation game. The knowledge of the observer is stored in K-environments of the form (O, V, \prec) . The set of names V represents all the substitutable free names (i.e. names of kind (1) or (2)). The set of names O contains all the messages that were emitted by P and Q , except the names of V . Finally, the relation \prec indicates for each substitutable name x the available knowledge $\{n \in O \mid n \prec x\}$ that had possibly been acquired by the observer at the moment the name x was input. Thus, the relation \prec constrains the messages that may possibly be or have been received at a particular moment from the observer.

Definition 23 (K-environment).

A K-environment is a triple (O, V, \prec) such that $O \cup V$ is a finite subset of N , $O \cap V = \emptyset$ and $\prec \subseteq O \times V$. The set of all K-environments is \mathcal{K} .

If $\mathit{pe} = (O, V, \prec)$ ¹ is a K-environment, the names $n(\mathit{pe})$ of pe are $O \cup V$.

If pe is a K-environment, and $n \in N$, it is possible to extend pe with n in two ways. Either n is meant to be an emitted name and it is added to the constant part of pe , or n is meant to be a received name and it is added to the variable part of pe and put in relation with all already emitted names. If n is already contained in pe , its addition to pe has no effect.

Definition 24 (extension of a K-environment).

Let $\mathit{pe} = (O, V, \prec)$ be a K-environment and $n \in N$. We define

1. $\mathit{pe} +_o n := (O', V, \prec)$ where $O' := O \cup \{n\}$ if $n \notin V$ and $O' := O$ otherwise.
2. if $n \notin O \cup V$, $\mathit{pe} +_i n := (O, V \cup \{n\}, \prec')$ where $\prec' := \prec \cup (O \times \{n\})$ and otherwise, $\mathit{pe} +_i n := \mathit{pe}$.

Since the knowledge of the observer only grows with time, an interesting class of K-environments is the class of *growing* K-environments.

Definition 25 (growing K-environment).

Let $\mathit{pe} = (O, V, \prec)$ be a K-environment. We say that pe is *growing* if there exists an injective mapping $z : \llbracket 1, n \rrbracket \rightarrow V$ (where $n = \text{card}(v)$) such that for all $1 \leq i < n$, we have $O_i \subseteq O_{i+1}$ where $O_i := \{n \in O \mid n \prec z(i)\}$.

¹ pe stands for π environment

Intuitively, a growing K-environment $\mathit{pe} = (O, V, \prec)$ can be written $O_1x_1O_2x_2 \cdots O_nx_nO_{n+1}$ where $V = \{x_1, \dots, x_n\}$, $O_i \subseteq O_{i+1}$, $O_{n+1} = O$ and $n \prec x_i \iff n \in O_i$.

By definition, if pe is growing, then $\mathit{pe} +_o n$ is growing and $\mathit{pe} +_i n$ is growing.

Growing K-environments are tightly related to *quantifier prefixes* of [139]. In this setting, a *quantifier prefix* is a list $Q_1x_1Q_2x_2 \dots Q_nx_n$ for some $n \geq 0$ and where Q_i is either ∇ or \forall , where ∇ is roughly a quantifier for fresh names and \forall is a quantifier for variable names.

Intuitively, a quantifier prefix corresponds to the growing K-environment (O, V, \prec) defined by $O := \{x_i \mid Q_i = \nabla\}$, $V := \{x_i \mid Q_i = \forall\}$ and for $x_i \in O$ and $x_j \in V$, $x_i \prec x_j \iff i < j$.

Conversely, a growing K-environment $\mathit{pe} = (O, V, \prec)$ corresponds to several quantifier prefixes. If pe is growing, then pe can be written $O_1x_1O_2x_2 \cdots O_nx_nO_{n+1}$ where $V = \{x_1, \dots, x_n\}$, $O_i \subseteq O_{i+1}$, $O_{n+1} = O$ and $n \prec x_i \iff n \in O_i$. The corresponding quantifier prefixes are of the form $\overline{\nabla}O_1\forall x_1\overline{\nabla}(O_2 \setminus O_1)\forall x_2 \cdots \overline{\nabla}(O_n \setminus O_{n-1})\forall x_n\overline{\nabla}(O_{n+1} \setminus O_n)$ where $\overline{\nabla}\{y_1, \dots, y_n\}$ stands for $\nabla y_1\nabla y_2 \dots \nabla y_n$. Note that it is almost always the case that ∇ quantifiers can be interchanged in $\text{FO}\lambda^{\nabla\Delta}$ [97].

Keeping in mind that a substitution represents the potential inputs the observer could have generated, we define the set of respectful substitutions. A substitution σ respects a K-environment $\mathit{pe} = (O, V, \prec)$ if it affects only substitutable names (those in V) and if for each $x \in V$, it takes only values that were possible to generate at the moment when x was input. This means that such a name x can use any name in V (this corresponds to fusing two substitutable names), or use any name in O that was known by the observer when x was input (this is indicated by the relation \prec) or use any new fresh name not contained in pe (this corresponds to the creation of free names by the observer).

Definition 26 (respectful substitution).

We say that a substitution σ respects a K-environment $\mathit{pe} = (O, V, \prec)$, and write $\sigma \blacktriangleright \mathit{pe}$, if:

1. $\text{supp}(\sigma) \subseteq V$
2. $\forall x \in V : x\sigma \in O \implies x\sigma \prec x$

Any K-environment $\mathit{pe} = (O, V, \prec)$ may, under the impact of some respectful substitution σ , be straightforwardly updated to pe^σ . In general, the knowledge contained in O should be updated to $O\sigma$. However, in the

π calculus, substitution deals only with names, and since $O \cap V = \emptyset$ and $\text{supp}(\sigma) \subseteq V$ we have $O\sigma = O$. The set V of substitutable names should keep all the names that were not affected by σ , and in addition list all the new names that were created by the observer, as visible in the substitution objects. The fact that we put the names created by the environment in the substitutable part gives a “lazy” flavour to our definition, because it allows the observer to uncover itself gradually. Particular care must be taken when computing the new relation \prec' because of the possibility that σ fuses two names of V . Fusing two names x and y (by $x\sigma = y\sigma$) corresponds to a voluntary loss of power of the observer: the only admissible values for the fused name are those that were admissible for *both* x and y .

Definition 27 (K-environment updating).

Let $pe = (O, V, \prec)$ be a K-environment and σ a substitution such that $\sigma \blacktriangleright pe$. The updated environment of pe by σ is $pe^\sigma := (O, V', \prec')$ where

$$\begin{aligned} V' &:= (V \setminus \text{supp}(\sigma)) \cup \{x\sigma \mid x \in \text{supp}(\sigma) \wedge x\sigma \notin O\} \\ \prec' &:= \{(n, x') \mid \forall x \in V : x' \in \mathfrak{n}(x\sigma) \implies n \prec x\} \end{aligned}$$

Growth of K-environments is preserved by updating.

Lemma 8:

Let $pe = (O, V, \prec)$ be a growing K-environment and σ a substitution such that $\sigma \blacktriangleright pe$. Then pe^σ is growing.

PROOF

We write $V = \{x_1, \dots, x_n\}$ and $O_i = \{n \in O \mid n \prec x_i\}$. We assume that for $1 \leq i < n$, we have $O_i \subseteq O_{i+1}$.

By definition, $pe^\sigma = (O, V', \prec')$ where

$$V' := (V \setminus \text{supp}(\sigma)) \cup \{x\sigma \mid x \in \text{supp}(\sigma) \wedge x\sigma \notin O\}$$

and for $n \in O$ and $x' \in V'$

$$n \prec' x' \iff (\forall x \in V : x' \in \mathfrak{n}(x\sigma) \implies n \prec x)$$

If $x' \in V'$, there exists $x \in V$ such that $x' \in \mathfrak{n}(x\sigma)$ (by case distinction on $x' \in V'$). Thus, if $x' \in V$, the set $A_{x'} := \{1 \leq i \leq n \mid x' \in \mathfrak{n}(x_i\sigma)\}$ is not empty. For $x' \in V'$, we note $\text{id}_x(x')$ the minimal element of $A_{x'}$.

Since $O_i \subseteq O_{i+1}$ for $1 \leq i < n$, we have $O_i \subseteq O_j$ when $i \leq j$. Hence for all $n \in O$ and $i \leq j$, we have $n \prec x_i$ implies $n \prec x_j$.

If $n \in O$ and $x' \in V'$, we thus have $n \prec' x' \iff n \prec x_{\text{id}_x(x')}$.

We thus sort in ascending order the elements x' of V' according to $\text{id}_x(x')$. Such an ordering proves that pe^σ is growing. \blacksquare

Definition 28 (K-relation).

A K-relation \mathcal{R} is a subset of $\mathcal{K} \times P \times P$ such that for all $(\rho e, P, Q) \in \mathcal{R}$, we have that ρe is growing and $\text{fn}(P + Q) \subseteq \text{n}(\rho e)$. \mathcal{R} is *symmetric* if for all $(\rho e, P, Q) \in \mathcal{R}$, we have $(\rho e, Q, P) \in \mathcal{R}$.

The new variant of open bisimulation is now defined. It simply keeps track of whether dynamically freed names are substitutable or not. If they are, then we explicitly state that previously created names may be used in future substitutions. Names that will be created later on—by the process itself—will not be permitted.

Definition 29 (K-open bisimulation).

A symmetric K-relation \mathcal{R} is a *K-open bisimulation*, if for all $(\rho e, P, Q) \in \mathcal{R}$ and for each substitution σ that respects ρe , whenever $P\sigma \xrightarrow{\mu} P'$ with $\text{bn}(\mu) \cap (\text{n}(\rho e) \cup \text{n}(\sigma)) = \emptyset$, there exists Q' such that $Q\sigma \xrightarrow{\mu} Q'$ and

- if $\mu = \tau$, then $(\rho e^\sigma, P', Q') \in \mathcal{R}$
- if $\mu = a(x)$ then $(\rho e^\sigma +_i x, P', Q') \in \mathcal{R}$
- if $\mu = \bar{a}vz$ or $\mu = \bar{a}z$ then $(\rho e^\sigma +_o z, P', Q') \in \mathcal{R}$

We see in this definition that O collects all the messages emitted by P and Q (but the addition $\rho e^\sigma +_o z$ has only effect when $\mu = \bar{a}vz$ since ρe contains all the free names of P and Q) and V collects all substitutable names.

P and Q are K-open ρe -bisimilar, written $P \sim_K^{\rho e} Q$, if there is a K-open bisimulation \mathcal{R} such that $(\rho e, P, Q) \in \mathcal{R}$.

It is possible to represent any K-environment by some F-environment. The idea is that all names in O should be kept pairwise distinct (they were fresh names) and for all $(n, x) \in O \times V$, if n cannot be used to generate x (i.e. $\neg(n \prec x)$), then n and x should be distinct ($n \neq x$).

Definition 30 (F-environment of a K-environment).

Let $\rho e = (O, V, \prec)$ be a K-environment. The F-environment induced by ρe is $\text{dist}_F(\rho e) := (D, O)$ where

$$D := O^\neq \cup \bigcup_{n \in O \wedge x \in V \wedge \neg(n \prec x)} \{(n, x), (x, n)\}$$

Note that if $\rho e \in \mathcal{K}$, then $\text{dist}_F(\rho e) \in \mathcal{F}$.

Let us come back to the analogy between growing K-environments and quantifier prefixes. A quantifier prefix $\mathcal{Q}x_1\mathcal{Q}x_2\dots\mathcal{Q}x_n$ generates the following distinction [139]:

$$\{(x_i, x_j), (x_j, x_i) \mid i \neq j, \mathcal{Q}_i = \mathcal{Q}_j = \nabla \text{ or } i < j, \mathcal{Q}_i = \forall, \mathcal{Q}_j = \nabla\}$$

We show that this distinction is the same as the one generated by the K-environment (O, V, \prec) where $O = \{x_i \mid \mathcal{Q}_i = \nabla\}$, $V = \{x_i \mid \mathcal{Q}_i = \forall\}$ and for $x_i \in O, x_j \in V, x_i \prec x_j \iff i < j$.

By definition, the generated distinction is

$$O^\neq \cup \{(n, x), (x, n) \mid n \in O, x \in V, \neg(n \prec x)\}$$

Clearly, $O^\neq = \{(x_i, x_j), (x_j, x_i) \mid i \neq j, \mathcal{Q}_i = \mathcal{Q}_j = \nabla\}$.

Moreover, if $x_i \in O$ and $x_j \in V$, then $\neg(x_i \prec x_j) \iff i > j$ so

$$\{(n, x), (x, n) \mid n \in O, x \in V, \neg(n \prec x)\} = \{(x_i, x_j), (x_j, x_i) \mid i < j, \mathcal{Q}_i = \forall, \mathcal{Q}_j = \nabla\}$$

Hence the two distinctions are the same.

Conversely, if $\mathfrak{pe} = (O, V, \prec)$ is written $O_1x_1O_2x_2\dots O_nx_nO_{n+1}$ where $V = \{x_1, \dots, x_n\}$, $O_i \subseteq O_{i+1}$, $O_{n+1} = O$ and $n \prec x_i \iff n \in O_i$.

Let $\mathcal{Q}_1y_1\dots\mathcal{Q}_py_p$ a quantifier prefix corresponding to \mathfrak{pe} . By definition, it is of the form $\overline{\nabla}O_1\forall x_1\overline{\nabla}(O_2 \setminus O_1)\forall x_2\dots\overline{\nabla}(O_n \setminus O_{n-1})\forall x_n\overline{\nabla}(O_{n+1} \setminus O_n)$.

The distinction generated by the quantifier prefix is

$$\{(y_i, y_j), (y_j, y_i) \mid i \neq j, \mathcal{Q}_i = \mathcal{Q}_j = \nabla \text{ or } i < j, \mathcal{Q}_i = \forall, \mathcal{Q}_j = \nabla\}$$

By definition, we have $\mathcal{Q}_i = \nabla$ if and only if $y_i \in O$ so

$$\{(y_i, y_j), (y_j, y_i) \mid i \neq j, \mathcal{Q}_i = \mathcal{Q}_j = \nabla\} = O^\neq$$

Moreover we have $i < j, \mathcal{Q}_i = \forall, \mathcal{Q}_j = \nabla$ if and only if $y_i \in V, y_j \in O$ and $y_j \in (O_{n+1} \setminus O_i) = O \setminus O_i$, i.e. $\neg(y_j \prec y_i)$. So

$$\{(x_i, x_j), (x_j, x_i) \mid i < j, \mathcal{Q}_i = \forall, \mathcal{Q}_j = \nabla\} = \{(n, x), (x, n) \mid n \in O, x \in V, \neg(n \prec x)\}$$

Hence the two distinctions are the same.

The next lemma gives a precise correspondence between respectfulness of a F-environment and respectfulness of a K-environment.

Lemma 9:

Let $\mathfrak{pe} = (O, V, \prec)$ be a K-environment and σ a substitution. Then

$$\sigma \blacktriangleright \mathfrak{pe} \iff \text{supp}(\sigma) \subseteq V \wedge \sigma \blacktriangleright \text{dist}_F(\mathfrak{pe})$$

PROOF

Let D such that $\text{dist}_{\mathbb{F}}(\rho e) = (D, O)$.

- First assume that $\sigma \blacktriangleright \rho e$.

By definition, we have $\text{supp}(\sigma) \subseteq V$ and $\forall x \in V : x\sigma \in O \implies x\sigma \prec x$.

Since $\text{supp}(\sigma) \subseteq V$ and $O \cap V = \emptyset$, we have $\text{supp}(\sigma) \cap O = \emptyset$.

Let $(x, y) \in D$. We have to show that $x\sigma \neq y\sigma$. There are four cases (according to the definition of D): either $x, y \in O$ with $x \neq y$, or $x \in O, y \in V$ and $\neg(x \prec y)$ or the two other symmetric cases.

By case distinction, assume that $x, y \in O$ and $x \neq y$. Since $\text{supp}(\sigma) \cap O = \emptyset$, we have $x\sigma = x, y\sigma = y$, hence $x\sigma \neq y\sigma$.

Now assume that $x \in O, y \in V$ and $\neg(x \prec y)$. Since $\text{supp}(\sigma) \cap O = \emptyset$, we have $x\sigma = x$. Assume by contradiction that $y\sigma = x\sigma = x$, then we have $y\sigma \in O$. Thus, we have $y\sigma \prec y$ which is equivalent to $x \prec y$ and thus leading to a contradiction. So $x\sigma \neq y\sigma$.

The two other symmetric cases are treated in the same way.

Hence $\sigma \blacktriangleright \text{dist}_{\mathbb{F}}(\rho e)$.

- Assume now that $\text{supp}(\sigma) \subseteq V \wedge \sigma \blacktriangleright \text{dist}_{\mathbb{F}}(\rho e)$.

We have then that $\sigma \triangleright D$.

By hypothesis, $\text{supp}(\sigma) \subseteq V$.

Let $x \in V$ and assume that $x\sigma \in O$. We have to show that $x\sigma \prec x$. Assume by contradiction that $\neg(x\sigma \prec x)$. Then, by definition of D , we have that $(x\sigma, x) \in D$. Since σ respects D , we have $x\sigma\sigma \neq x\sigma$, but since $x\sigma \in O$ and $\text{supp}(\sigma) \cap O = \emptyset$, we have $x\sigma\sigma = x\sigma$, obtaining a contradiction.

Hence $\sigma \blacktriangleright \rho e$. ■

The next lemma studies the updating of a K-environment.

Lemma 10:

Let $\rho e = (O, V, \prec)$ be a K-environment, D such that $\text{dist}_{\mathbb{F}}(\rho e) = (D, O)$ and σ a substitution such that $\sigma \blacktriangleright \rho e$. Then $\text{dist}_{\mathbb{F}}(\rho e^\sigma) = (D\sigma, O)$. □

PROOF

Let $(D', O) = \text{dist}_{\mathbb{F}}(\rho e^\sigma)$. We have to show that $D' = D\sigma$.

By definition, $D' = O^\neq \cup \bigcup_{n \in O \wedge x' \in V' \wedge \neg(n \prec' x')} \{(n, x'), (x', n)\}$ where $V' = (V \setminus \text{supp}(\sigma)) \cup \{x \in \text{supp}(\sigma) \wedge x\sigma \notin O\}$ and \prec' is defined by

$$n \prec' x' \iff \bigwedge_{x \in V \wedge x' \in \text{en}(x\sigma)} n \prec x$$

Let $(x', y') \in D'$. If $(x', y') \in O^\neq$ then $(x', y') \in D\sigma$ since $\text{supp}(\sigma) \cap O = \emptyset$. So, assume that $x' \in O, y' \in V'$ and $\neg(x' \prec' y')$. By definition, we have that there exists in $y \in V$ such that $y' \in n(y\sigma)$ and $\neg(x' \prec y)$. So, we have, by definition of D , $(x', y) \in D$ and since $x'\sigma = x'$ and $y\sigma = y'$, we have thus $(x', y') \in D\sigma$. So $D' \subseteq D\sigma$.

Let $(x', y') \in D\sigma$. By definition, there exists $(x, y) \in D$ such that $x' = x\sigma$ and $y' = y\sigma$. If $(x, y) \in O^\neq$, then $x' = x$ and $y' = y$ and thus $(x', y') \in D'$. Now assume that $x \in O, y \in V$ and $\neg(x \prec y)$. Since $\text{supp}(\sigma) \cap O = \emptyset$, we have $x' = x$. If $y' \in O$ then $(x', y') \in O^\neq$ and $(x', y') \in D'$. Assume that $y' \notin O$. Then, by definition of V' , $y' \in V'$. We have, since $y' = y\sigma$, $y' \in n(y\sigma)$ and since $\neg(x' \prec y)$, we have, by definition of \prec' , $\neg(x' \prec' y')$ and thus $(x', y') \in D'$. So $D\sigma \subseteq D'$. ■

Finally, the following lemma studies how the distinction corresponding to an environment evolves when a fresh name is added to the constant part.

Lemma 11:

Let $\mathbf{pe} = (O, V, \prec)$ be a K -environment and z a fresh name (i.e. neither in O , nor in V) and let $(D, O) = \text{dist}_{\mathbb{F}}(\mathbf{pe})$.

Then $\text{dist}_{\mathbb{F}}(\mathbf{pe} +_o z) = (D \cup \{z\} \otimes (O \cup V), O \cup \{z\})$. □

PROOF

Since z is fresh, we have $\mathbf{pe} +_o z = (O \cup \{z\}, V, \prec)$.

So, by definition, we have $\text{dist}_{\mathbb{F}}(\mathbf{pe} +_o z) = (D', O \cup \{z\})$ where the distinction D' has been defined to be

$$D' := (O \cup \{z\})^\neq \cup \bigcup_{n \in O \cup \{z\} \wedge x \in V \wedge \neg(n \prec x)} \{(n, x), (x, n)\}$$

Thus

$$\begin{aligned} D' &= O^\neq \cup \{z\} \otimes O \\ &\cup \bigcup_{n \in O \wedge x \in V \wedge \neg(n \prec x)} \{(n, x), (x, n)\} \\ &\cup \bigcup_{x \in V} \{(z, x), (x, z)\} \end{aligned}$$

because z does not appear in \prec and so for every $x \in V$ we have $\neg(z \prec x)$.

Hence $D' = D \cup \{z\} \otimes (O \cup V)$. ■

From this, it follows that K-open bisimilarity is sound with respect to F-open bisimilarity.

Lemma 12:

If $P \sim_K^{\rho\mathbf{e}} Q$ then $P \sim_F^{\text{distr}(\rho\mathbf{e})} Q$.

PROOF

Let $\mathcal{R} = \{(\text{distr}_F(\rho\mathbf{e}), P, Q) \mid P \sim_K^{\rho\mathbf{e}} Q\}$.

\mathcal{R} is a F-open' bisimulation.

Let $((D, C), P, Q) \in \mathcal{R}$ and σ such that $\sigma \blacktriangleright (D, C)$.

Assume that $P\sigma \xrightarrow{\mu} P'$ with $\text{bn}(\mu) \cap (\text{n}(D) \cup \text{fn}(P+Q) \cup \text{n}(\sigma)) = \emptyset$.

There exists $\rho\mathbf{e} = (C, V, \prec)$ such that $(D, C) = \text{distr}_F(\rho\mathbf{e})$ and $\text{bn}(\mu) \cap \text{n}(\rho\mathbf{e}) = \emptyset$ and $P \sim_K^{\rho\mathbf{e}} Q$ (because it is closed under bijective substitutions).

Let σ' the restriction of σ to V . Clearly, $\sigma \blacktriangleright (D, C)$ because $\text{n}(D) \subseteq C \cup V$ and $C \subseteq \neq D$. Moreover, since $\text{fn}(P+Q) \subseteq C \cup V$, σ and σ' agrees on $\text{fn}(P+Q)$. Note also that since $\text{n}(D) \subseteq C \cup V$, we have $D\sigma = D\sigma'$.

So by Lemma 9, we have $\sigma' \blacktriangleright \rho\mathbf{e}$. Since $P\sigma' \xrightarrow{\mu} P'$ and $\text{bn}(\mu) \cap (\text{n}(\rho\mathbf{e}) \cup \text{n}(\sigma')) = \emptyset$, there exists Q' such that $Q\sigma' \xrightarrow{\mu} Q'$.

If μ is not a bound output, it is easy to see that $((D\sigma, C), P', Q') \in \mathcal{R}$, thanks in particular to Lemma 10.

If $\mu = \bar{a}vz$, then $P' \sim_K^{\rho\mathbf{e}' + \circ z} Q'$. By Lemma 11 and Lemma 10, there is a finite set X that contains $C \cup \text{fn}(P+Q)$ such that $\text{distr}_F(\rho\mathbf{e}' + \circ z) = D\sigma \cup (\{z\} \otimes X\sigma) = D'$ and then $(D', P', Q') \in \mathcal{R}$.

Hence \mathcal{R} is a F-open' bisimulation. ■

Under the condition that the F-environment (D, C) is representable by a K-environment $\rho\mathbf{e}$, F-open (D, C) -bisimilarity is sound with respect to K-open $\rho\mathbf{e}$ -bisimilarity.

Lemma 13:

If $P \sim_F^{(D,C)} Q$ and $(D, C) = \text{distr}_F(\rho\mathbf{e})$ for some growing K-environment $\rho\mathbf{e}$ with $\text{fn}(P+Q) \subseteq \text{n}(\rho\mathbf{e})$ then $P \sim_K^{\rho\mathbf{e}} Q$.

PROOF

Let $\mathcal{R} = \{(\rho\mathbf{e}, P, Q) \mid P \sim_F^{\text{distr}_F(\rho\mathbf{e})} Q \wedge \text{fn}(P+Q) \subseteq \text{n}(\rho\mathbf{e})\}$.

We show that \mathcal{R} is a K-open bisimulation.

Let $(\rho\mathbf{e}, P, Q) \in \mathcal{R}$ and σ such that $\sigma \blacktriangleright \rho\mathbf{e}$.

Assume that $P\sigma \xrightarrow{\mu} P'$ with $\text{bn}(\mu) \cap (\text{n}(\rho\mathbf{e}) \cup \text{n}(\sigma)) = \emptyset$.

We have $P \sim_{\mathbb{F}}^{\text{dist}_{\mathbb{F}}(\rho e)} Q$. By Lemma 9, we have $\sigma \blacktriangleright \text{dist}_{\mathbb{F}}(\rho e)$.

Let $(D, C) = \text{dist}_{\mathbb{F}}(\rho e)$. By definition, we have $\mathfrak{n}(D) \subseteq \mathfrak{n}(\rho e)$ and $\text{fn}(P + Q) \subseteq \mathfrak{n}(\rho e)$, so $\text{bn}(\mu) \cap (\text{fn}(P + Q) \cup \mathfrak{n}(D) \cup \mathfrak{n}(\sigma)) = \emptyset$.

So there exists Q' such that $Q\sigma \xrightarrow{\mu} Q'$.

If μ is not a bound output, we have $P' \sim_{\mathbb{F}}^{(D\sigma, C)} Q'$.

So, if $\mu = \tau$, we have by Lemma 10 $(\rho e^\sigma, P', Q') \in \mathcal{R}$.

If $\mu = a(x)$, we have $\text{dist}_{\mathbb{F}}(\rho e^\sigma +_i x) = \text{dist}_{\mathbb{F}}(\rho e^\sigma)$, so by Lemma 10, $(\rho e^\sigma +_i x, P', Q') \in \mathcal{R}$.

If $\mu = \bar{a}z$, we have $\rho e^\sigma +_o z = \rho e^\sigma$ since z is already in ρe^σ , so by Lemma 10, $(\rho e^\sigma +_o z, P', Q') \in \mathcal{R}$.

If μ is a bound output $\bar{a}vz$, we have $P' \sim_{\mathbb{F}}^{D', C \cup \{z\}} Q'$ with $D' := (D\sigma \cup (\{z\} \otimes (\text{fn}(P\sigma + Q\sigma) \cup \mathfrak{n}(D\sigma))), C \cup \{z\})$.

So by Lemma 10 and Lemma 11, we have $(\rho e^\sigma +_o z, P', Q') \in \mathcal{R}$ because the only difference between the updated distinction above and the distinction of $\text{dist}_{\mathbb{F}}(\rho e^\sigma +_o z)$ is the presence of some irrelevant names for the bisimilarity; the important fact is that $\text{fn}(P + Q) \subseteq \mathfrak{n}(\rho e)$.

Hence \mathcal{R} is a \mathbb{K} -open bisimulation. \blacksquare

2.3.3 About congruence properties

Due to its richer underlying information structures, we may formulate stronger congruence properties for \mathbb{K} -open bisimilarity than for the original open bisimilarity. The following results were conjectured in [48] and proved in [50].

We prove with the help of \mathbb{K} -open bisimilarity that, under some conditions, open D -bisimilarity is a congruence for a bigger class of contexts than just D -respectful contexts.

The idea is, if $(D, O) = \text{dist}_{\mathbb{F}}(O, V, \prec)$, (1) to admit contexts that are D -respectful, and furthermore (2) to admit contexts where the hole occurs underneath an input prefix that binds a name x of V , but only if, in addition, every name of $\{n \in O \mid \neg(n \prec x)\}$ appears underneath a respective restriction on the “path” from the hole-binding input prefix for x to the hole. This corresponds to the fact that, in the bisimulation, a name n in O comes from a restriction and a name x from V comes from an input prefix and we have $n \prec x$ if n was disclosed before x was input. Before going deeper into the formal details, let us understand the intuition by means of a simple example.

Example 1

Let $P = \bar{x}(x).0 \mid y(z).0$ and $Q = \bar{x}(x).y(z).0 + y(z).\bar{x}(x).0$.

It is known and easily verifiable that $P \sim_0^D Q$ with $D = \{(x, y), (y, x)\}$.

Let $C = \{y\}$ and $V = \{x\}$, and note that $(D, C) = \text{dist}_F((C, V, \emptyset))$.

Observe that $P \sim_K^{(C, V, \emptyset)} Q$.

Now, consider the context $X[\cdot] = a(x).(vy)[\cdot]$.

Then $X[P] \sim_0^{\emptyset} X[Q]$, although $X[\cdot]$ is not considered by D -congruence.

However, $X[\cdot]$ follows our above informal rule of admissible contexts.

Finally, just note in passing that also $X[P] \sim_K^{(\emptyset, \{a\}, \emptyset)} X[Q]$. *

Definition 31.

Let $\mathit{pe} = (O, V, \prec) \in \mathcal{K}$ and $n \in N$.

We define $\mathit{pe} - n := (O \setminus \{n\}, V \setminus \{n\}, \prec \setminus (\{n\} \times N \cup N \times \{n\}))$

Note that if pe is a K-environment, then $\mathit{pe} - n$ is also a K-environment.

Note also that, in addition, if pe is growing, so is $\mathit{pe} - n$.

The following lemma states that, as for open bisimulation, only free names of processes are relevant in the sense that their consideration in environments matters.

Lemma 14:

Assume that $P \sim_K^{\mathit{pe}} Q$ and $n \notin \text{fn}(P + Q)$. Then $P \sim_K^{\mathit{pe} - n} Q$.

PROOF

Assume that $P \sim_K^{\mathit{pe}} Q$ with $\mathit{pe} = (O, V, \prec)$.

By Lemma 12, we have $P \sim_F^{\text{dist}_F(\mathit{pe})} Q$ where $\text{dist}_F(\mathit{pe}) = (D, O)$ and $D = O^{\neq} \cup \bigcup_{(n,x) \in O \times V} \{(n, x), (x, n) \mid \neg(n \prec x)\}$.

By Lemma 6, we have that $P \sim_0^D Q$.

Since $n \notin \text{fn}(P + Q)$, we have that $P \sim_0^{D-n} Q$.

By definition, $D - n = D \setminus (\{n\} \times N \cup N \times \{n\})$.

Let $O' = O \setminus \{n\}$, $V' = V \setminus \{n\}$ and $\prec' = \prec \setminus (\{n\} \times N \cup N \times \{n\})$.

We then have that

$$D - n = O'^{\neq} \cup \bigcup_{(n,x) \in O' \times V'} \{(n, x), (x, n) \mid \neg(n \prec' x)\}$$

Thus, by Lemma 7, we have that $P \sim_F^{(D-n, O')} Q$.

Moreover, since $(D - n, O') = \text{dist}_F(\mathit{pe} - n)$, we have by Lemma 13 that $P \sim_K^{\mathit{pe} - n} Q$. Hence the result. ■

Conversely, it is possible to add some fresh variables to a K-environment.

Lemma 15:

Assume that $P \sim_K^{pe} Q$. Then $P \sim_K^{pe+i^n} Q$.

PROOF

If $n \in n(pe)$, then $pe +_i n = pe$ and the result is obvious.

Otherwise, this follows from $\text{dist}_F(pe) = \text{dist}_F(pe +_i n)$. ■

Before rephrasing the congruence properties of open bisimilarity, as stated in Proposition 1, in terms of K-open bisimilarity, we extend the second part of Definition 24 to a finite set of names (this is because K-open bisimilarity requires the *initial* environment to mention every free name).

Definition 32.

Let $pe = (O, V, \prec)$ be a K-environment and $N = \{n_1, \dots, n_k\}$ a finite set of names.

We define $pe +_i N$ to be pe_k where

- $pe_0 = pe$
- $pe_{j+1} = pe_j +_i n_j$

Note that the previous definition does not depend on the order in which we add the elements of N .

The following result is obviously true:

Lemma 16:

Let $pe = (O, V, \prec)$ be a K-environment, $(D, O) := \text{dist}_F(pe)$ and N a finite set of names. Then $\text{dist}_F(pe +_i N) = (D, O)$. □

In analogy with congruence results for standard open bisimilarity as of Proposition 1, we state one for K-open bisimilarity.

Proposition 2:

Let P, Q two processes and $pe = (O, V, \prec)$ a K-environment with $P \sim_K^{pe} Q$. Then,

1. $\tau.P \sim_K^{pe} \tau.Q$
2. $\forall R : R \mid P \sim_K^{pe+i \text{fn}(R)} R \mid Q$ and $P \mid R \sim_K^{pe+i \text{fn}(R)} Q \mid R$
3. $\forall R : R + P \sim_K^{pe+i \text{fn}(R)} R + Q$ and $P + R \sim_K^{pe+i \text{fn}(R)} Q + R$
4. $!P \sim_K^{pe} !Q$
5. $[x=y]P \sim_K^{pe+i\{x,y\}} [x=y]Q$

6. $\bar{a}\langle z \rangle.P \sim_{\mathbf{K}}^{\text{pe}+i\{a,z\}} \bar{a}\langle z \rangle.Q$
7. $(\nu z)P \sim_{\mathbf{K}}^{\text{pe}-z} (\nu z)Q$
8. if $x \notin O \cup V$ then $a(x).P \sim_{\mathbf{K}}^{\text{pe}+i^a} a(x).Q$
9. if $O = \{x\}$ and $\{y \in V \mid \neg(x \prec y)\} = \emptyset$ then $a(x).P \sim_{\mathbf{K}}^{\text{pe}+i^a} a(x).Q$
10. if $x \in V$ and $\{n \in O \mid \neg(n \prec x)\} = \emptyset$ then $a(x).P \sim_{\mathbf{K}}^{\text{pe}+i^a} a(x).Q$

PROOF

Let $(D, O) = \text{dist}_{\mathbb{F}}(\text{pe})$. Since $P \sim_{\mathbf{K}}^{\text{pe}} Q$, by Lemma 12, we have $P \sim_{\mathbb{F}}^{(D, O)} Q$. So by Lemma 6, we have $P \sim_{\mathbb{O}}^D Q$.

By Proposition 1, we have $P \mid R \sim_{\mathbb{O}}^D Q \mid R$. So by Lemma 7, we have $P \mid R \sim_{\mathbb{F}}^{(D, O)} Q \mid R$.

Since $(D, O) = \text{dist}_{\mathbb{F}}(\text{pe}) = \text{dist}_{\mathbb{F}}(\text{pe} +_i \text{fn}(R))$ by Lemma 16 and by Lemma 13 we have that $P \mid R \sim_{\mathbf{K}}^{\text{pe} +_i \text{fn}(R)} Q \mid R$.

A similar reasoning applies to the summation operator, the replication operator, the match prefix, the silent prefix and the output prefix.

For the restriction operator, we have $(\nu z)P \sim_{\mathbb{O}}^{D-z} (\nu z)Q$. So, since $(O \setminus \{z\})^{\neq} \subseteq D - z$, we have $(\nu z)P \sim_{\mathbb{F}}^{(D-z, O \setminus \{z\})} (\nu z)Q$. Moreover, since $(D - z, O \setminus \{z\}) = \text{dist}_{\mathbb{F}}(\text{pe} - z)$, we thus have $(\nu z)P \sim_{\mathbf{K}}^{\text{pe}-z} (\nu z)Q$.

For the input prefix, if $x \notin \mathfrak{n}(D)$, then $a(x).P \sim_{\mathbb{O}}^D a(x).Q$.

By definition, we have

$$D := O^{\neq} \cup \bigcup_{n \in O \wedge x \in V \wedge \neg(n \prec x)} \{(n, x), (x, n)\}$$

So $\mathfrak{n}(D) \subseteq O \cup V$. Thus, if $x \notin O \cup V$, then $x \notin \mathfrak{n}(D)$. So $a(x).P \sim_{\mathbb{O}}^D a(x).Q$ and finally $a(x). \sim_{\mathbf{K}}^{\text{pe}+i^a} a(x).Q$.

Similarly, if $x \in V$ and $\{n \in O \mid \neg(n \prec x)\} = \emptyset$, then $x \notin \mathfrak{n}(D)$. Thus $a(x). \sim_{\mathbf{K}}^{\text{pe}+i^a} a(x).Q$.

Finally, if $O = \{x\}$ and $\{y \in V \mid \neg(x \prec y)\} = \emptyset$ then $x \notin \mathfrak{n}(D)$ and thus $a(x). \sim_{\mathbf{K}}^{\text{pe}+i^a} a(x).Q$. Note that the only case where O^{\neq} is empty is when $\text{card } O < 2$. ■

From the previous proposition, we can deduce a set of contexts that are safe concerning K-open bisimilarity; for such contexts $C[\cdot]$ we have that if $P \sim_{\mathbf{K}}^{\text{pe}} Q$, then there exists pe' such that $C[P] \sim_{\mathbf{K}}^{\text{pe}'} C[Q]$.

Definition 33.

Let $\rho\mathbf{e} = (O, V, \prec)$ be a K-environment. We define the set of $\rho\mathbf{e}$ -respectful contexts as the language generated by the grammar defined as follows. For each subset $N \subseteq O \cup V$, we define a non-terminal symbol $C_N[\cdot]$. The start symbol is $C_\emptyset[\cdot]$. The production rules are of the form:

$$\begin{array}{l}
C_N[\cdot] ::= [\cdot] \quad \text{if } N = \emptyset \\
\quad | \tau.C_N[\cdot] \\
\quad | P | C_N[\cdot] \\
\quad | C_N[\cdot] | P \\
\quad | P + C_N[\cdot] \\
\quad | C_N[\cdot] + P \\
\quad | !C_N[\cdot] \\
\quad | [x=y]C_N[\cdot] \\
\quad | (vz)C_{N \setminus \{z\}}[\cdot] \\
\quad | \bar{a}(z).C_N[\cdot] \\
\quad | a(x).C_N[\cdot] \quad \text{if } x \notin O \cup V \\
\quad | a(x).C_{N \cup N'}[\cdot] \quad \text{if } x \in V \text{ and } N' = \{n \in O \mid \neg(n \prec x)\} \\
\quad | a(x).C_{N \cup N'}[\cdot] \quad \text{if } O = \{x\} \text{ and } N' = \{y \in V \mid \neg(x \prec y)\}
\end{array}$$

The idea is simply that when a name x of V is bound by an input prefix, then according to Proposition 2, it is sufficient that every name $n \in O$ such that $\neg(n \prec x)$ is removed from the environment, which is done via restrictions. The index N of each non-terminal $C_N[\cdot]$ simply remembers all such names.

Example 2

Back to Example 1, we have in this case $\rho\mathbf{e} = (\{y\}, \{x\}, \emptyset)$.

The context $X[\cdot] = a(x).(vy)[\cdot]$ is obtained by applying the second rule for adding an input prefix (since $x \in \{x\}$) and at this point the name y is pushed in the set of names N . Then the rule for the restriction is used to remove y from the set N . Finally the hole is placed.

The derivation path for obtaining $X[\cdot]$ via the grammar of Definition 33 is

$$C_\emptyset[\cdot] \rightarrow a(x).C_{\{y\}}[\cdot] \rightarrow a(x).(vy)C_\emptyset[\cdot] \rightarrow a(x).(vy)[\cdot]$$

From the previous developments, we deduce the following proposition.

Proposition 3:

If $P \sim_K^{\rho\mathbf{e}} Q$ and $C[\cdot]$ is a $\rho\mathbf{e}$ -respectful context then there exists a K-environment $\rho\mathbf{e}'$ such that $C[P] \sim_K^{\rho\mathbf{e}'} C[Q]$ (and $\rho\mathbf{e}'$ is built according to rules given in Proposition 2).

PROOF

The proof is by induction on $C[\cdot]$.

More precisely, we prove that if $P \sim_K^{pe} Q$ with $pe = (O, V, \prec)$ then for all context $C[\cdot]$, for all $N \subseteq n(pe)$, if $C[\cdot] = C_N[\cdot]$ for some context $C_N[\cdot]$ built according to formation rules of Definition 33 then there exists $pe' = (O', V', \prec')$ such that $C[P] \sim_K^{pe'} C[Q]$ and

- $O' \subseteq O$
- $O' \cap N = \emptyset$
- $\forall x \in V' : (x \in V \implies \{n \in O' \mid \neg(n \prec' x)\} \subseteq \{n \in O \mid \neg(n \prec x)\})$
- $\forall x \in V' : (x \notin V \implies \{n \in O' \mid \neg(n \prec' x)\} = \emptyset$
- $\forall n \in O' : \{y \in V' \mid \neg(n \prec' y)\} \subseteq \{y \in V \mid \neg(n \prec y)\})$
- $\forall n \in O' : \{y \in V' \mid \neg(n \prec' y)\} \cap N = \emptyset$

We first show that if pe' satisfies the above condition then so do $pe' - z$ and $pe' +_i z$ for any name z .

The result is obvious for $pe' - z$ since it removes some more information. Moreover, it satisfies also the property for $N \cup \{z\}$ since z is removed.

If $z \in n(pe')$ then $pe' +_i z = pe'$ so the result is trivial.

If $z \notin n(pe')$, then $pe' +_i z = (O', V' \cup \{z\}, \prec' \cup O' \times \{z\})$.

So $\{n \in O' \mid \neg(n \prec'' z)\} = \emptyset$. Thus it is clear that the result holds.

We can now show the corollary by induction on $C[\cdot]$.

If $C[\cdot] = [\cdot]$, then necessarily $N = \emptyset$ and $pe' = pe$ satisfies the property.

If $C[\cdot] = C'[\cdot] \mid R$ for some $C'[\cdot]$ and R . Assume that $C[\cdot] = C_N[\cdot]$ for some $N \subseteq n(pe)$ and $C_N[\cdot]$. Then necessarily $C'[\cdot] = C'_N[\cdot]$ for some $C'_N[\cdot]$.

By induction, there is pe' such that $C'[P] \sim_K^{pe'} C'[Q]$. Then by Proposition 2, we have $C'[P] \mid R \sim_K^{pe' +_i \text{fn}(R)} C'[Q] \mid R$. Clearly $pe'' = pe' +_i \text{fn}(R)$ satisfies the property.

We now treat the non trivial cases.

If $C[\cdot] = (\nu z) C'[\cdot] = C_N[\cdot]$. Then $C'[\cdot] = C_{N \setminus \{z\}}[\cdot]$.

By induction, we have $C'[P] \sim_K^{pe'} C'[Q]$ for pe' satisfying the condition for $N \setminus \{z\}$. So by Proposition 2, we have $(\nu z) C'[P] \sim_K^{pe' - z} (\nu z) C'[Q]$ and $pe'' = pe' - z$ satisfies the property for N (by case distinction on $z \in N$ or not).

If $C[\cdot] = a(x).C'[\cdot] = C_N[\cdot]$. There are three cases:

1. if $x \notin O \cup V$ and $C'[\cdot] = C'_N[\cdot]$

By induction, we have $C'[P] \sim_K^{pe'} C'[Q]$.

We write $pe' = (O', V', \prec')$.

Since $x \notin O \cup V$ then $x \notin O$ so $x \notin O'$.

If $x \in V'$. Then since $x \notin V$ then $\{n \in O' \mid \neg(n \prec' x)\} = \emptyset$. So by Proposition 2, we have $a(x).C'[P] \sim_K^{pe'+i\{a\}} a(x).C'[Q]$ and $pe'' = pe' + i\{a\}$ satisfies the property.

If $x \notin V'$ then $x \notin O' \cup V'$.

By Proposition 2, we have $a(x).C'[P] \sim_K^{pe'+i\{a\}} a(x).C'[Q]$ and $pe'' = pe' + i\{a\}$ satisfies the property.

2. if $x \in V$ and $C'[\cdot] = C'_{N \cup N'}[\cdot]$ for $N' = \{n \in O \mid \neg(n \prec x)\}$.

By induction we have $C'[P] \sim_K^{pe'} C'[Q]$ with $pe' = (O', V', \prec')$ satisfying the property for $N \cup N'$.

Since $x \in V$ and $O \cap V = \emptyset$, we have $x \notin O$ so $x \notin O'$.

If $x \in V'$, then $\{n \in O' \mid \neg(n \prec' x)\} \subseteq N'$. Since $O' \cap (N \cup N') = \emptyset$, we have $\{n \in O' \mid \neg(n \prec' x)\} = \emptyset$.

So by Proposition 2, $a(x).C'[P] \sim_K^{pe'+i\{a\}} a(x).C'[Q]$. And we have that $pe'' = pe' + i\{a\}$ satisfies the property for N .

If $x \notin V'$, then by Proposition 2, $a(x).C'[P] \sim_K^{pe'+i\{a\}} a(x).C'[Q]$ and $pe' + i\{a\}$ satisfies the property for N .

3. if $O = \{x\}$ and $C'[\cdot] = C'_{N \cup N'}[\cdot]$ for $N' = \{y \in V \mid \neg(x \prec y)\}$.

By induction, we have $C'[P] \sim_K^{pe'} C'[Q]$ with $pe' = (O', V', \prec')$ satisfying the property for $N \cup N'$.

Since $x \in O$ and $O \cap V = \emptyset$, we have $x \notin V$.

If $x \in O'$ then since $O' \subseteq O = \{x\}$, we have $O' = \{x\}$.

By hypothesis, we have $\{y \in V' \mid \neg(x \prec' y)\} \subseteq N'$ and also that $\{y \in V' \mid \neg(x \prec' y)\} \cap (N \cup N') = \emptyset$. Thus $\{y \in V' \mid \neg(x \prec' y)\} = \emptyset$.

So by Proposition 2, $a(x).C'[P] \sim_K^{pe'+i\{a\}} a(x).C'[Q]$ and $pe' + i\{a\}$ satisfies the property for N .

If $x \notin O'$ then $O' = \emptyset$. So by Proposition 2, $a(x).C'[P] \sim_K^{pe'+i\{a\}} a(x).C'[Q]$ and $pe' + i\{a\}$ satisfies the property for N . ■

Definition 34.

Let $\rho\epsilon$ be a K-environment.

A relation $\mathcal{R} \subseteq \mathbf{P} \times \mathbf{P}$ is a $\rho\epsilon$ -congruence if for all $(P, Q) \in \mathcal{R}$ and for all $\rho\epsilon$ -respectful contexts $C[\cdot]$ we have $(C[P], C[Q]) \in \mathcal{R}$.

Lemma 17:

Let $\rho\epsilon$ be a K-environment and $(D, O) = \text{dist}_{\mathbb{F}}(\rho\epsilon)$.

Then, every $\rho\epsilon$ -congruence is also a D -congruence.

PROOF

We show that any D -respectful context is also $\rho\epsilon$ -respectful.

We write $\rho\epsilon = (O, V, \prec)$.

More precisely, we show that if $C[\cdot]$ respects D then $C[\cdot] = C_{\emptyset}[\cdot]$ for some $C_{\emptyset}[\cdot]$ built according to rules of Definition 33.

Without loss of generality, we can assume that the bound names of $C[\cdot]$ are pairwise distinct.

The only non trivial case is $C[\cdot] = a(x).C'[\cdot]$.

Since $C[\cdot]$ respects D , we have $x \notin n(D)$ and $C'[\cdot]$ also respects D .

By induction, $C'[\cdot] = C'_{\emptyset}[\cdot]$ for some context $C'_{\emptyset}[\cdot]$ satisfying the rules of Definition 33.

There are three (disjoint) cases: $x \in O$, $x \in V$ or $x \notin O \cup V$.

Recall first that

$$D = O^{\neq} \cup \bigcup_{n \in O \wedge x \in V \wedge \neg(n \prec x)} \{(n, x), (x, n)\}$$

If $x \in O$, then necessarily $O = \{x\}$ otherwise there is $y \in O$ with $y \neq x$ and $(x, y) \in O^{\neq}$. Since $x \notin n(D)$, we also have that $\{y \in V \mid \neg(x \prec y)\} = \emptyset$, otherwise x is in the second part of the union defining D . So $C[\cdot] = C_{\emptyset}[\cdot]$ with $C_{\emptyset}[\cdot] = a(x).C'_{\emptyset}[\cdot]$ satisfies the rules of Definition 33.

If $x \in V$, then necessarily $\{n \in O \mid \neg(n \prec x)\} = \emptyset$ since otherwise x is in the second part of the union defining D . So $C[\cdot] = C_{\emptyset}[\cdot]$ with $C_{\emptyset}[\cdot] = a(x).C'_{\emptyset}[\cdot]$ satisfies the rules of Definition 33.

Finally, if $x \notin O \cup V$, then clearly $C[\cdot] = C_{\emptyset}[\cdot]$ with $C_{\emptyset}[\cdot] = a(x).C'_{\emptyset}[\cdot]$ satisfies the rules of Definition 33. ■

The following theorem states that open D -bisimilarity has better congruence properties than those expressed by D -congruence.

Theorem 2:

Let $\rho\epsilon$ be a K-environment and $(D, O) = \text{dist}_{\mathbb{F}}(\rho\epsilon)$.

Then, open D -bisimilarity is a $\rho\epsilon$ -congruence.

PROOF

This follows from the previous observations. ■

Conclusion

In this chapter, we have presented the pi calculus, a process calculus of concurrent and mobile systems. We have described the behaviour of processes with the help of a labelled transition system. We then have presented several notions of behavioural equivalence, expressed in terms of bisimulation. In particular, we have explained that open bisimulation of Sangiorgi is an attractive candidate notion of bisimulation for the pi calculus, due to its congruence properties and its easy implementability. In a second part, we have revisited open bisimulation and defined two variants of open bisimulation that are more precise concerning their management of names than Sangiorgi's open bisimulation, where every name is potentially considered as a variable. In a certain way, F-open bisimulation, by making some kind of dynamic typing of names, reconciles two different presentations of the pi calculus: the standard one where there is only one kind of names and the presentation where two different syntactic categories are used for names and *variables*, as by Honda and Yoshida [84] or by Hennessy and Rathke [80]. We have been able to exploit the additional information contained in K-environments to improve the congruence properties of open bisimulation. We will see in the following chapters that the definition of K-open bisimilarity has other benefits, since it can be smoothly lifted to the spi calculus, a cryptographic extension of the pi calculus.

Chapter 3

The Spi Calculus

We present in this chapter the spi calculus of Abadi and Gordon [9]. We describe its syntax and its labelled late transitions system. We then define an alternative labelled transitions system that keeps track of some additional information. Finally, we discuss about process equivalences in the spi calculus and how they can be used to express security properties.

3.1 Syntax

The spi calculus is an extension of the pi calculus that permits the transmission of cryptographic messages. It was designed to formalise and study cryptographic protocols [9]. Contrary to the original presentation of Abadi and Gordon where the cryptographic primitives are embedded in the process syntax, we present a spi calculus which is parametrised by the language of cryptographic messages as e.g. in [61, 40, 39]. The resulting spi calculus is thus more easily extensible and resembles applied pi calculus [6] in this respect.

3.1.1 The language of messages

The messages we will consider are built according to the grammar of Table 3.1. The basic components of message are *names*. The set of messages is written M .

Given a message M , it is possible:

- to compute its *hashing* $H(M)$,
- to retrieve its associated *public key* $\text{pub}(M)$,

$M, N ::= a$		$H(M)$	name
		$\text{pub}(M)$	hashing
		$\text{priv}(M)$	public key
		$(M . N)$	private key
		$\text{Enc}_N^s M$	pair
		$\text{Enc}_N^a M$	shared-key encryption
			asymmetric encryption

Table 3.1: The messages of the spi calculus

- to retrieve its associated *private key* $\text{priv}(M)$.

We will denote by the meta-variable op operators in $\{\text{pub}, \text{priv}, H\}$. Given two messages M and N , it is possible:

- to construct the *pairing* of M and N yielding $(M . N)$,
- to encrypt with a *shared-key cryptography* algorithm the message M with the key N yielding $\text{Enc}_N^s M$,
- to encrypt with a *public-key cryptography* algorithm the message M with the key N yielding $\text{Enc}_N^a M$.

We assume that the decryption of the *cyphertext* $\text{Enc}_N^s M$ succeeds and yields M *if and only if* the message N is given as decryption key.

We assume that the decryption of the cyphertext $\text{Enc}_N^a M$ succeeds and yields M *if and only if* either

- $N = \text{pub}(N')$ for some message N' and $\text{priv}(N')$ is given as decryption key, or
- $N = \text{priv}(N')$ for some message N' and $\text{pub}(N')$ is given as decryption key.

To refer to the inverse key in case of asymmetric cryptography, we define a function $\text{inv}(\cdot) : \mathbf{M} \rightarrow \mathbf{M} \cup \{\perp\}$ such that for every message $M \in \mathbf{M}$, we have $\text{inv}(\text{pub}(M)) := \text{priv}(M)$, $\text{inv}(\text{priv}(M)) := \text{pub}(M)$ and otherwise (i.e. M is not of the form $\text{pub}(M')$ or $\text{priv}(M')$) $\text{inv}(M) := \perp$. This function is (almost) involutive.

Lemma 18:

Let $M, N \in \mathbf{M}$. If $\text{inv}(M) = N$ then $\text{inv}(N) = M$. □

$E, F ::= a$	name
$H(E)$	hashing
$\text{pub}(E)$	public key
$\text{priv}(E)$	private key
$\pi_1(E)$	first projection
$\pi_2(E)$	second projection
$(E.F)$	pair
$\text{Enc}_F^s E$	shared-key encryption
$\text{Enc}_F^a E$	asymmetric encryption
$\text{Dec}_F^s E$	shared-key decryption
$\text{Dec}_F^a E$	asymmetric decryption

Table 3.2: The expressions of the spi calculus

3.1.2 Expressions

Syntax

To dynamically manipulate messages, we introduce the set of expressions E . Expressions are built according to the grammar of Table 3.2.

As for messages, expressions are built from names.

Given an expression E , it is possible:

- to compute the hashing $H(E)$ of the message represented by expression E ,
- to retrieve the public key $\text{pub}(E)$ associated to the message represented by E ,
- to retrieve the private key $\text{priv}(E)$ associated to the message represented by E ,
- to try to obtain the first projection $\pi_1(E)$ of the message represented by E ,
- to try to obtain the second projection $\pi_2(E)$ of the message represented by E .

Given two expressions E and F , it is possible:

- to construct the pairing $(E.F)$ of the messages represented by E and F ,

- to construct the shared-key encryption $\text{Enc}_F^s E$ of the message represented by E with the message represented by F as encryption key,
- to construct the asymmetric encryption $\text{Enc}_F^a E$ of the message represented by E with the message represented by F as encryption key,
- to try to obtain the decryption $\text{Dec}_F^s E$ of the message represented by E with the message represented by F as decryption key with the shared-key decryption algorithm,
- to try to obtain the decryption $\text{Dec}_F^a E$ of the message represented by E with the message represented by F as decryption key with the public-key decryption algorithm.

Concrete Evaluation

According to previous developments, it is possible to form *arbitrary* expressions. However, it might be the case that the computation of some expressions fails. For example, this is the case when one tries to decrypt a cyphertext with the wrong decryption key. We formalise the notion of computation of an expression—or *concrete evaluation*—with the function $\mathbf{e}_c(\cdot) : E \rightarrow M \cup \{\perp\}$. If the concrete evaluation of an expression E is a message M , we say that the (concrete) evaluation of E succeeds and equals M ; otherwise, we say that it fails (and $\mathbf{e}_c(E) = \perp$).

The definition of $\mathbf{e}_c(\cdot)$ is given in Table 3.3. The case of failure clearly originates from the incorrect application of a deconstructor to a term.

Indeed, the concrete evaluation of an expression that does not contain any deconstructors (i.e. a message) always succeeds:

Lemma 19:

For all $M \in M$, we have $\mathbf{e}_c(M) = M$.

PROOF

By induction on M . ■

However, there are expressions with deconstructors whose evaluation succeeds. For example $E = \pi_1((a.b))$. Then $\mathbf{e}_c(E) = a$.

3.1.3 Processes

We give a brief and informal interpretation of the processes of the spi calculus which are composed of:

- An *empty* process 0 . It can do nothing.

$\mathbf{e}_c(a)$	$:= a$	if $a \in \mathbf{N}$
$\mathbf{e}_c((E_1 . E_2))$	$:= (M_1 . M_2)$	if $\mathbf{e}_c(E_1) = M_1 \in \mathbf{M}$ and $\mathbf{e}_c(E_2) = M_2 \in \mathbf{M}$
$\mathbf{e}_c(\text{Enc}_F^s E)$	$:= \text{Enc}_{M_2}^s M_1$	if $\mathbf{e}_c(E) = M_1 \in \mathbf{M}$ and $\mathbf{e}_c(F) = M_2 \in \mathbf{M}$
$\mathbf{e}_c(\text{Enc}_F^a E)$	$:= \text{Enc}_{M_2}^a M_1$	if $\mathbf{e}_c(E) = M_1 \in \mathbf{M}$ and $\mathbf{e}_c(F) = M_2 \in \mathbf{M}$
$\mathbf{e}_c(\text{op}(E))$	$:= \text{op}(M)$	if $\mathbf{e}_c(E) = M \in \mathbf{M}$ and $\text{op} \in \{\text{pub}, \text{priv}, \text{H}\}$
$\mathbf{e}_c(\pi_1(E))$	$:= M_1$	if $\mathbf{e}_c(E) = (M_1 . M_2) \in \mathbf{M}$
$\mathbf{e}_c(\pi_2(E))$	$:= M_2$	if $\mathbf{e}_c(E) = (M_1 . M_2) \in \mathbf{M}$
$\mathbf{e}_c(\text{Dec}_F^s E)$	$:= M_1$	if $\mathbf{e}_c(E) = \text{Enc}_{M_2}^s M_1 \in \mathbf{M}$ and $\mathbf{e}_c(F) = M_2$
$\mathbf{e}_c(\text{Dec}_F^a E)$	$:= M_1$	if $\mathbf{e}_c(E) = \text{Enc}_{M_2}^a M_1 \in \mathbf{M}$ and $\mathbf{e}_c(F) = \text{inv}(M_2) \in \mathbf{M}$
$\mathbf{e}_c(E)$	$:= \perp$	in all other cases

Table 3.3: Concrete evaluation of expressions

- A *silent prefix* $\tau.P$. It can evolve invisibly to P .
- An *input prefix* $E(x).P$. If the evaluation of E succeeds and equals $a \in \mathbf{N}$ then it can receive a message M along the channel a and continue as P with M substituted for x . Otherwise, it can do nothing.
- An *output prefix* $\bar{E}(F).P$. If the evaluation of E succeeds and equals $a \in \mathbf{N}$ and the evaluation of F succeeds and equals M then it can send the message M along the channel a and continue as P . Otherwise, it can do nothing.
- A *guard* ϕP . If the *guard* ϕ is true then it can evolve as P . Otherwise, it can do nothing.
- A *parallel composition* $P | Q$. The processes P and Q can evolve independently or may interact via shared communication channels.
- A *sum* $P + Q$. It can behave either as P or Q .
- A *restriction* $(vz)P$. A fresh name z , whose scope is restricted to P , is created and then it behaves like P .

- A *replication* $!P$. It can be thought as an infinite parallel composition $P|P|\dots$ of the process P .

The syntax of the spi calculus processes P is summarised in Table 3.4.

$P, Q ::=$	$\mathbf{0}$	inactive process
	$\tau.P$	silent prefix
	$E(x).P$	input prefix
	$\bar{E}\langle F \rangle.P$	output prefix
	ϕP	guard
	$P Q$	parallel composition
	$P+Q$	sum
	$(\nu z)P$	restriction
	$!P$	replication

Table 3.4: The processes of the spi calculus

The language of *guards* F is described by Table 3.5. A guard is either

- the matching $[E=F]$ of two expressions E and F which is true if both the evaluations of E and F succeed and equal the same message M , or
- the test of the predicate $[E:N]$ which is true if the evaluation of the expression E succeeds and equals a name. The presence of this capability is motivated by the fact that since communication channels are required to be names, it gives a mean to test whether an expression evaluates to a name by attempting to transmit on it. This ability is thus expressed by this guard.

$\phi ::=$	$[E=F]$	matching
	$[E:N]$	is a name

Table 3.5: The guards of the spi calculus

The evaluation function $\mathbf{e}(\cdot) : F \rightarrow \{\mathbf{true}, \mathbf{false}\}$, which is given in Table 3.6, defines the notion of truth for guards.

It is convenient to define a short cut—written $[E:M]$ —for the guard $[E=E]$. Note that $\mathbf{e}([E:M]) = \mathbf{true}$ if and only if the evaluation of E succeeds.

$$\begin{aligned}
\mathbf{e}([E=F]) &:= \mathbf{true} && \text{if } \mathbf{e}_c(E) = \mathbf{e}_c(F) = M \in \mathbf{M} \\
\mathbf{e}([E:N]) &:= \mathbf{true} && \text{if } \mathbf{e}_c(E) = a \in \mathbf{N} \\
\mathbf{e}(\phi) &:= \mathbf{false} && \text{in all other cases}
\end{aligned}$$

Table 3.6: Evaluation of guards

3.1.4 Example

We consider a simple cryptographic protocol consisting of two principals A and B taken from [9]. We assume that A and B share the key k_{AB} . The protocol is that A sends the message M encrypted with k_{AB} to B .

The corresponding protocol narration would be:

$$A \rightarrow B : \text{Enc}_{k_{AB}}^S M$$

In spi calculus, this gives the process $P(M)$ where

$$\begin{aligned}
A(M) &:= \overline{c_B}(\text{Enc}_{k_{AB}}^S M). \mathbf{0} \\
B &:= c_B(x). [\text{Dec}_{k_{AB}}^S x : \mathbf{M}] F(\text{Dec}_{k_{AB}}^S x) \\
P(M) &:= (\nu k_{AB}) (A(M) \mid B)
\end{aligned}$$

B is parametrised by its continuation $F(y)$. The parameter y is intended to represent the clear text of the transmission. Before performing this continuation, B checks whether the received message is indeed a cyphertext that has been encrypted with the key k_{AB} thanks to the guard $[\text{Dec}_{k_{AB}}^S x : \mathbf{M}]$ (recall that x will be substituted by the input message).

3.2 Labelled Transitions System

As in the pi calculus, the behaviour of processes is described by a *labelled transitions system*. Before giving the late semantics of the spi calculus, we briefly define auxiliary notions such as free names, bound names and substitutions. We then give an alternative labelled transitions system that collects some additional information about processes.

3.2.1 Names, free names, bound names, α -conversion

Definition 35 ($\mathbf{n}(M)$, $\mathbf{n}(E)$, $\mathbf{n}(\phi)$).

The *names* $\mathbf{n}(M)$ of a message M are the set of names that appear in M . It is defined inductively by:

$$\begin{aligned}
n(a) &:= \{a\} \\
n((M_1 . M_2)) &:= n(M_1) \cup n(M_2) \\
n(\text{Enc}_{M_2}^s M_1) &:= n(M_1) \cup n(M_2) \\
n(\text{Enc}_{M_2}^a M_1) &:= n(M_1) \cup n(M_2) \\
n(\text{op}(M)) &:= n(M) \quad \text{with op} \in \{\text{pub, priv, H}\}
\end{aligned}$$

The *names* $n(E)$ of an expression E are the set of names that appear in E . It is defined inductively by:

$$\begin{aligned}
n(a) &:= \{a\} \\
n((E_1 . E_2)) &:= n(E_1) \cup n(E_2) \\
n(\text{Enc}_{E_2}^s E_1) &:= n(E_1) \cup n(E_2) \\
n(\text{Enc}_{E_2}^a E_1) &:= n(E_1) \cup n(E_2) \\
n(\text{op}(E)) &:= n(E) \quad \text{with op} \in \{\text{pub, priv, H}\} \\
n(\pi_1(E)) &:= n(E) \\
n(\pi_2(E)) &:= n(E) \\
n(\text{Dec}_{E_2}^s E_1) &:= n(E_1) \cup n(E_2) \\
n(\text{Dec}_{E_2}^a E_1) &:= n(E_1) \cup n(E_2)
\end{aligned}$$

The *names* $n(\phi)$ of a guard ϕ are the set of names that appear in ϕ . It is defined inductively by:

$$\begin{aligned}
n([E=F]) &:= n(E) \cup n(F) \\
n([E:\mathbf{N}]) &:= n(E)
\end{aligned}$$

We extend this definition to set of messages, expressions, guards in the obvious way. We also write $n(E_1, \dots, E_n, M_1, \dots, M_l, \phi_1, \dots, \phi_k)$ as a shortcut for $\bigcup_{1 \leq i \leq n} n(E_i) \cup \bigcup_{1 \leq i \leq l} n(M_i) \cup \bigcup_{1 \leq i \leq k} n(\phi_i)$.

It is obvious that if the evaluation of an expression E succeeds and equals M then the names of M are also names of E .

Lemma 20:

Let $E \in E$. If $\mathbf{e}_c(E) = M \in \mathbf{M}$ then $n(M) \subseteq n(E)$. □

The converse result is not true. For instance, consider $E = \pi_1((a . b))$. Then $\mathbf{e}_c(E) = a$, $n(E) = \{a, b\}$ and $n(\mathbf{e}_c(E)) = \{a\}$. So $n(E) \not\subseteq n(\mathbf{e}_c(E))$.

Definition 36 (bn(P), fn(P)).

The *bound names* $\text{bn}(P)$ of a process P are defined inductively by:

$$\begin{aligned}
\text{bn}(\mathbf{0}) &:= \emptyset \\
\text{bn}(\tau.P) &:= \text{bn}(P) \\
\text{bn}(E(x).P) &:= \text{bn}(P) \cup \{x\} \\
\text{bn}(\overline{E}(F).P) &:= \text{bn}(P) \\
\text{bn}(\phi P) &:= \text{bn}(P) \\
\text{bn}(P \mid Q) &:= \text{bn}(P) \cup \text{bn}(Q) \\
\text{bn}(P + Q) &:= \text{bn}(P) \cup \text{bn}(Q) \\
\text{bn}((\nu x)P) &:= \text{bn}(P) \cup \{x\} \\
\text{bn}(!P) &:= \text{bn}(P)
\end{aligned}$$

The *free names* $\text{fn}(P)$ of a process P are defined inductively by:

$$\begin{aligned}
\text{fn}(\mathbf{0}) &:= \emptyset \\
\text{fn}(\tau.P) &:= \text{fn}(P) \\
\text{fn}(E(x).P) &:= \text{n}(E) \cup (\text{fn}(P) \setminus \{x\}) \\
\text{fn}(\overline{E}(F).P) &:= \text{n}(E) \cup \text{n}(F) \cup \text{fn}(P) \\
\text{fn}(\phi P) &:= \text{n}(\phi) \cup \text{fn}(P) \\
\text{fn}(P \mid Q) &:= \text{fn}(P) \cup \text{fn}(Q) \\
\text{fn}(P + Q) &:= \text{fn}(P) \cup \text{fn}(Q) \\
\text{fn}((\nu x)P) &:= \text{fn}(P) \setminus \{x\} \\
\text{fn}(!P) &:= \text{fn}(P)
\end{aligned}$$

As above, we extend these definitions to set of processes and extend the notation for finite sequence of processes.

We say that P and Q are α -equivalent and write $P =_\alpha Q$, if P and Q only differ by a change of bound names.

In the following, we identify α -equivalent processes. In particular, we assume that in a process bound names are different from each other and from the free names.

3.2.2 Substitutions

Definition 37 (substitution, $\text{supp}(\sigma)$, $\text{cosupp}(\sigma)$, $\text{n}(\sigma)$).

A substitution is a function $\sigma : N \rightarrow M$ such that its *support* $\text{supp}(\sigma) := \{x \in N \mid \sigma(x) \neq x\}$ is finite.

If σ is a substitution, the *co-support* $\text{cosupp}(\sigma)$ of σ is $\text{cosupp}(\sigma) := \{\sigma(x) \mid x \in \text{supp}(\sigma)\}$.

The *names* $\text{n}(\sigma)$ of σ are $\text{n}(\sigma) := \text{supp}(\sigma) \cup \text{n}(\text{cosupp}(\sigma))$.

We use the notation $[^{M_1/x_1, \dots, M_n/x_n}]$ when we enumerate a substitution σ . As previously, we use the postfix notation for applying substitutions.

Definition 38 (application of a substitution).

Substitutions are applied to messages according to the following inductive definition:

$$\begin{aligned}
(a)\sigma &:= \sigma(a) \\
((M_1 . M_2))\sigma &:= (M_1\sigma . M_2\sigma) \\
(\text{Enc}_{M_2}^s M_1)\sigma &:= \text{Enc}_{M_2\sigma}^s M_1\sigma \\
(\text{Enc}_{M_2}^a M_1)\sigma &:= \text{Enc}_{M_2\sigma}^a M_1\sigma \\
(\text{op}(M))\sigma &:= \text{op}(M\sigma) \quad \text{with op} \in \{\text{pub, priv, H}\}
\end{aligned}$$

Substitutions are applied to expressions according to the following inductive definition:

$$\begin{aligned}
(a)\sigma &:= \sigma(a) \\
((E_1 . E_2))\sigma &:= (E_1\sigma . E_2\sigma) \\
(\text{Enc}_{E_2}^s E_1)\sigma &:= \text{Enc}_{E_2\sigma}^s E_1\sigma \\
(\text{Enc}_{E_2}^a E_1)\sigma &:= \text{Enc}_{E_2\sigma}^a E_1\sigma \\
(\text{op}(E))\sigma &:= \text{op}(E\sigma) \quad \text{with op} \in \{\text{pub, priv, H}\} \\
(\pi_1(E))\sigma &:= \pi_1(E\sigma) \\
(\pi_2(E))\sigma &:= \pi_2(E\sigma) \\
(\text{Dec}_{E_2}^s E_1)\sigma &:= \text{Dec}_{E_2\sigma}^s E_1\sigma \\
(\text{Dec}_{E_2}^a E_1)\sigma &:= \text{Dec}_{E_2\sigma}^a E_1\sigma
\end{aligned}$$

Substitutions are applied to guards according to the following inductive definition:

$$\begin{aligned}
([E = F])\sigma &:= [E\sigma = F\sigma] \\
([E : \mathbf{N}])\sigma &:= [E\sigma : \mathbf{N}]
\end{aligned}$$

Substitutions are applied to processes according to the following inductive definition:

$$\begin{aligned}
(\mathbf{0})\sigma &:= \mathbf{0} \\
(\tau.P)\sigma &:= \tau.(P\sigma) \\
(E(x).P)\sigma &:= E\sigma(x).P\sigma \quad \text{if } x \notin \mathbf{n}(\sigma) \\
(\overline{E}(F).P)\sigma &:= \overline{E\sigma}(F\sigma).P\sigma \\
(\phi P)\sigma &:= \phi\sigma P\sigma \\
((\nu z)P)\sigma &:= (\nu z)P\sigma \quad \text{if } z \notin \mathbf{n}(\sigma) \\
(P \mid Q)\sigma &:= P\sigma \mid Q\sigma \\
(P + Q)\sigma &:= P\sigma + Q\sigma \\
(!P)\sigma &:= !P\sigma
\end{aligned}$$

To satisfy the side conditions for input prefix or restriction, we might first need to rename the bound names of P . This is possible because we have identified α -equivalent processes.

We extend the application of a substitution to sets: if X is a set (of expressions, messages, processes, ...), $X\sigma := \{x\sigma \mid x \in X\}$.

The inverse key relation is preserved by substitutions.

Lemma 21:

Let $M, N \in \mathbf{M}$ and $\sigma : N \rightarrow M$ a substitution. If $\text{inv}(M) = N$ then $\text{inv}(M\sigma) = N\sigma$. \square

If the evaluation of E succeeds and equals M , then the evaluation of $E\sigma$ succeeds and equals $M\sigma$.

Lemma 22:

Let $E \in \mathbf{E}$ and $\sigma : N \rightarrow M$ a substitution. If $\mathbf{e}_c(E) = M \in \mathbf{M}$ then $\mathbf{e}_c(E\sigma) = \mathbf{e}_c(E)\sigma = M\sigma$.

PROOF

By induction on E and by Lemma 19. \blacksquare

If we see E as an approximation of $E\sigma$ and M as an approximation of $M\sigma$, the previous lemma says that the the more precise the expression, the more precise the evaluation.

The converse result is not true. For example, take $E = \pi_1(x)$ and $\sigma = \{(a.b)/x\}$. Then $\mathbf{e}_c(E\sigma) = a$ but $\mathbf{e}_c(E) = \perp$.

3.2.3 Late Semantics

As we briefly mentioned in the chapter devoted to the pi calculus (see Page 33), we adopt a different presentation for the labelled transitions system. Thus, the late semantics of the spi calculus relates processes with *agents*.

Definition 39 (agent).

An *agent* A is either a process P or an *abstraction* $F = (x)P$ (where $P \in \mathbf{P}$) or a *concretion* $C = (\nu\bar{z})\langle M \rangle P$ (where $M \in \mathbf{M}$, \bar{z} is a finite set of names such that $\{\bar{z}\} \subseteq \mathbf{n}(M)$ and $P \in \mathbf{P}$). When \bar{z} is empty, we simply write $\langle M \rangle P$ instead of $(\nu\emptyset)\langle M \rangle P$.

The bound names $\text{bn}(A)$ of an agent are defined by:

$$\begin{aligned} \text{bn}(A) &:= \text{bn}(P) && \text{if } A = P \in \mathbf{P} \\ \text{bn}(A) &:= \{x\} \cup \text{bn}(P) && \text{if } A = (x)P \\ \text{bn}(A) &:= \{\bar{z}\} \cup \text{bn}(P) && \text{if } A = (\nu\bar{z})\langle M \rangle P \end{aligned}$$

The free names $\text{fn}(A)$ of an agent are defined by:

$$\begin{aligned}
\text{fn}(A) &:= \text{fn}(P) && \text{if } A = P \in \mathbf{P} \\
\text{fn}(A) &:= \text{fn}(P) \setminus \{x\} && \text{if } A = (x)P \\
\text{fn}(A) &:= \text{fn}(P) \setminus \{\bar{z}\} && \text{if } A = (\nu\bar{z}) \langle M \rangle P
\end{aligned}$$

As for processes, we define α -equivalence for agents. In the following, we identify α -equivalent agents.

Substitutions are applied to agents according to the following definition:

$$\begin{aligned}
(A)\sigma &:= P\sigma && \text{if } A = P \in \mathbf{P} \\
(A)\sigma &:= (x)P\sigma && \text{if } A = (x)P \text{ and } x \notin \text{n}(\sigma) \\
(A)\sigma &:= (\nu\bar{z}) \langle M\sigma \rangle P\sigma && \text{if } A = (\nu\bar{z}) \langle M \rangle P \text{ and } \{\bar{z}\} \cap \text{n}(\sigma) = \emptyset
\end{aligned}$$

To simplify the presentation of the semantics, we define the composition of agents and processes.

Definition 40 (composition of agents and processes).

We define composition between agents and processes as follows:

$$\begin{aligned}
(\nu x)P &:= (\nu x)P \\
(\nu y)((x)P) &:= (x)(\nu y)P && \text{if } y \neq x \\
(\nu y)((\nu\bar{z}) \langle M \rangle P) &:= (\nu y\bar{z}) \langle M \rangle P && \text{if } y \notin \{\bar{z}\} \text{ and } y \in \text{n}(M) \\
(\nu y)((\nu\bar{z}) \langle M \rangle P) &:= (\nu\bar{z}) \langle M \rangle (\nu y)P && \text{if } y \notin \{\bar{z}\} \text{ and } y \notin \text{n}(M) \\
((x)P) | Q &:= (x)(P | Q) && \text{if } x \notin \text{fn}(Q) \\
((\nu\bar{z}) \langle M \rangle P) | Q &:= (\nu\bar{z}) \langle M \rangle (P | Q) && \text{if } \{\bar{z}\} \cap \text{fn}(Q) = \emptyset \\
Q | ((x)P) &:= (x)(Q | P) && \text{if } x \notin \text{fn}(Q) \\
Q | ((\nu\bar{z}) \langle M \rangle P) &:= (\nu\bar{z}) \langle M \rangle (Q | P) && \text{if } \{\bar{z}\} \cap \text{fn}(Q) = \emptyset
\end{aligned}$$

The pseudo-application defines how abstractions and concretions may interact.

Definition 41 (pseudo-application).

Let $F = (x)P$ an abstraction and $C = (\nu\bar{z}) \langle M \rangle Q$ a concretion with $\{\bar{z}\} \cap \text{fn}(P) = \emptyset$. The *pseudo-application* of F and C —written $F \bullet C$ — is defined by:

$$F \bullet C := (\nu\bar{z}) (P\{M/\bar{x}\} | Q)$$

Likewise, the pseudo-application of C and F is defined by:

$$C \bullet F := (\nu\bar{z}) (Q | P\{M/\bar{x}\})$$

μ	$\mathfrak{n}(\mu)$	$\mu\sigma$
τ	\emptyset	τ
\bar{a}	$\{a\}$	if $a\sigma \in N$ then $\overline{a\sigma}$ otherwise undefined
a	$\{a\}$	if $a\sigma \in N$ then $a\sigma$ otherwise undefined

Table 3.7: Notation for actions.

Labelled transitions take the form of $P \xrightarrow{\mu} A$ where P is a process, μ is an *action* and A is an agent. An action is the *silent action* τ or a barb. A *barb* is either a name a (representing input) or a *co-name* \bar{a} (representing output). We define several operations on actions, as summarised in Table 3.7. Note that in contrast to pi calculus actions, there is no notion of bound names on spi calculus actions, thanks to the presentation in terms of abstractions and concretions.

Definition 42 (late semantics of the spi calculus).

The labelled late semantics of the spi calculus is given by the derivation rules of Table 3.8 enriched by the symmetric variants of CLOSE-L, PAR-L and SUM-L.

Example 3

Consider the processes $A(M)$, B and $P(M)$ defined previously.

The following late transitions can be derived.

$$\begin{aligned}
A(M) &\xrightarrow{\overline{c_B}} \langle \text{Enc}_{k_{AB}}^s M \rangle \mathbf{0} \\
B &\xrightarrow{c_B} (x) [\text{Dec}_{k_{AB}}^s x : \mathbf{M}] F(\text{Dec}_{k_{AB}}^s x) \\
P(M) &\xrightarrow{\overline{c_B}} (\nu k_{AB}) \langle \text{Enc}_{k_{AB}}^s M \rangle (\mathbf{0} \mid c_B(x). [\text{Dec}_{k_{AB}}^s x : \mathbf{M}] F(\text{Dec}_{k_{AB}}^s x)) \\
P(M) &\xrightarrow{c_B} (x) (\overline{c_B} \langle \text{Enc}_{k_{AB}}^s M \rangle. \mathbf{0} \mid [\text{Dec}_{k_{AB}}^s x : \mathbf{M}] F(\text{Dec}_{k_{AB}}^s x)) \\
P(M) &\xrightarrow{\tau} (\nu k_{AB}) (\mathbf{0} \mid [\text{Dec}_{k_{AB}}^s \text{Enc}_{k_{AB}}^s M : \mathbf{M}] F(\text{Dec}_{k_{AB}}^s \text{Enc}_{k_{AB}}^s M))
\end{aligned}$$

*

The reaction relation is defined by $P \longrightarrow Q$ if and only if $P \xrightarrow{\tau} Q$. It expresses that P can evolve to Q as an effect of an internal action. We note \Longrightarrow the reflexive and transitive closure of \longrightarrow .

3.2.4 Semantics with Constraints

In the pi calculus, application of a substitution to a process does not diminish its capabilities, i.e. if $P \xrightarrow{\mu} A$ then $P\sigma \xrightarrow{\mu\sigma} A\sigma$. This result does not

$$\begin{array}{c}
\text{SILENT} \frac{}{\tau.P \xrightarrow{\tau} P} \qquad \text{INPUT} \frac{\mathbf{e}_c(E) = a \in N}{E(x).P \xrightarrow{a} (x)P} \\
\text{OUTPUT} \frac{\mathbf{e}_c(E) = a \in N \quad \mathbf{e}_c(F) = M \in M}{\bar{E}\langle F \rangle.P \xrightarrow{a} \langle M \rangle P} \\
\text{CLOSE-L} \frac{P \xrightarrow{a} F \quad Q \xrightarrow{\bar{a}} C}{P | Q \xrightarrow{\tau} F \bullet C} \qquad \text{RES} \frac{P \xrightarrow{\mu} A}{(vz)P \xrightarrow{\mu} (vz)A} \quad z \notin \mathfrak{n}(\mu) \\
\text{IFTHEN} \frac{P \xrightarrow{\mu} P' \quad \mathbf{e}(\phi) = \mathbf{true}}{\phi P \xrightarrow{\mu} P'} \qquad \text{PAR-L} \frac{P \xrightarrow{\mu} A}{P | Q \xrightarrow{\mu} A | Q} \\
\text{SUM-L} \frac{P \xrightarrow{\mu} A}{P + Q \xrightarrow{\mu} A} \qquad \text{REP-ACT} \frac{P \xrightarrow{\mu} A}{!P \xrightarrow{\mu} A | !P} \\
\text{REP-CLOSE} \frac{P \xrightarrow{a} F \quad P \xrightarrow{\bar{a}} C}{!P \xrightarrow{\tau} (F \bullet C) | !P} \qquad \text{ALPHA} \frac{P =_{\alpha} Q \quad Q \xrightarrow{\mu} B \quad B =_{\alpha} A}{P \xrightarrow{\mu} A}
\end{array}$$

Table 3.8: The late semantics of the spi calculus

hold in the spi calculus. A simple reason is that communication channels are required to be names. However, when applying a substitution, a name that was used as a communication channel may suddenly become a more complex message. Think for example about $P := a(x). \mathbf{0} | \bar{a}\langle b \rangle. \mathbf{0}$. Then P can perform the transition $P \xrightarrow{\tau} \mathbf{0} | \mathbf{0}$. But if we consider the substitution $\sigma := \{(a \cdot a)/a\}$ then $P\sigma$ cannot perform any transition.

Nevertheless, it is possible to give a simple condition on substitutions for this result to be true. To explicate this condition, we first need to define another labelled transitions system which accumulates type constraints along the derivation. The type constraints are simply (finite) sets of names S ; the names that cannot be substituted by anything else than a name for the transition to take place. They come from the initial application of rules INPUT, OUTPUT or IFTHEN. Indeed, the simple guard $[E:N]$ constrain the evaluation of E to be a name.

So, we first define the type constraint $\mathbf{nc}(\phi)$ induced by a guard ϕ .

Definition 43 ($\mathbf{nc}(\phi)$).

If $E \in E$, we define $\mathbf{nc}([E:N]) := \{\mathbf{e}_c(E)\}$.

If $E, F \in E$, we define $\mathbf{nc}([E=F]) := \emptyset$.

It is clear that if $\mathbf{e}(\phi) = \mathbf{true}$ then $\mathbf{nc}(\phi) \subseteq N$.

Labelled transitions with type constraints take the form of $P \xrightarrow[S]{\mu} A$ where μ is an action (see above) and S is a (finite) set of names.

Definition 44.

The labelled late semantics (with name constraints) of the spi calculus is given by the derivation rules of Table 3.9 enriched by the symmetric variants of NC-CLOSE-L, NC-PAR-L and NC-SUM-L.

As announced, the rules NC-INPUT, NC-OUTPUT and NC-IFTHEN add new type constraints. On the contrary, rule NC-RES may remove the restricted name z from the derived constraint; the reason being that only free names are of interests.

Example 4

With this new transition system, the transitions of Example 3 become:

$$\begin{aligned}
 A(M) &\xrightarrow[\{c_B\}]{\bar{c}_B} \langle \text{Enc}_{k_{AB}}^s M \rangle \mathbf{0} \\
 B &\xrightarrow[\{c_B\}]{c_B} (x)[\text{Dec}_{k_{AB}}^s x : \mathbf{M}]F(\text{Dec}_{k_{AB}}^s x) \\
 P(M) &\xrightarrow[\{c_B\}]{\bar{c}_B} (\nu k_{AB}) \langle \text{Enc}_{k_{AB}}^s M \rangle (\mathbf{0} \mid c_B(x).[\text{Dec}_{k_{AB}}^s x : \mathbf{M}]F(\text{Dec}_{k_{AB}}^s x)) \\
 P(M) &\xrightarrow[\{c_B\}]{c_B} (x)(\bar{c}_B \langle \text{Enc}_{k_{AB}}^s M \rangle . \mathbf{0} \mid [\text{Dec}_{k_{AB}}^s x : \mathbf{M}]F(\text{Dec}_{k_{AB}}^s x)) \\
 P(M) &\xrightarrow[\{c_B\}]{\tau} (\nu k_{AB}) (\mathbf{0} \mid [\text{Dec}_{k_{AB}}^s \text{Enc}_{k_{AB}}^s M : \mathbf{M}]F(\text{Dec}_{k_{AB}}^s \text{Enc}_{k_{AB}}^s M))
 \end{aligned}$$

*

The two semantics are equivalent:

Theorem 3:

1. If $P \xrightarrow{\mu} A$ there exists $S \subseteq N$ such that $P \xrightarrow[S]{\mu} A$.
2. If $P \xrightarrow[S]{\mu} A$ then $P \xrightarrow{\mu} A$.

PROOF

By rule induction. ■

$\text{NC-SILENT} \frac{}{\tau.P \xrightarrow[\emptyset]{\tau} P}$	$\text{NC-INPUT} \frac{\mathbf{e}_c(E) = a \in \mathbf{N}}{E(x).P \xrightarrow[\{a\}]{a} (x)P}$
$\text{NC-OUTPUT} \frac{\mathbf{e}_c(E) = a \in \mathbf{N} \quad \mathbf{e}_c(F) = M \in \mathbf{M}}{\bar{E}\langle F \rangle.P \xrightarrow[\{a\}]{\bar{a}} \langle M \rangle P}$	
$\text{NC-CLOSE-L} \frac{P \xrightarrow[S]{a} F \quad Q \xrightarrow[S']{\bar{a}} C}{P Q \xrightarrow[SUS']{\tau} F \bullet C}$	$\text{NC-RES} \frac{P \xrightarrow[S]{\mu} A}{(\nu z) P \xrightarrow[S \setminus \{z\}]{\mu} (\nu z) A} \quad z \notin \mathfrak{n}(\mu)$
$\text{NC-IFTHEN} \frac{P \xrightarrow[S]{\mu} A}{\phi P \xrightarrow[S \cup \text{nc}(\phi)]{\mu} A} \quad \mathbf{e}(\phi) = \mathbf{true}$	$\text{NC-PAR-L} \frac{P \xrightarrow[S]{\mu} A}{P Q \xrightarrow[S]{\mu} A Q}$
$\text{NC-SUM-L} \frac{P \xrightarrow[S]{\mu} A}{P + Q \xrightarrow[S]{\mu} A}$	$\text{NC-REP-ACT} \frac{P \xrightarrow[S]{\mu} A}{!P \xrightarrow[S]{\mu} A !P}$
$\text{NC-REP-CLOSE} \frac{P \xrightarrow[S]{a} F \quad P \xrightarrow[S']{\bar{a}} C}{!P \xrightarrow[SUS']{\tau} (F \bullet C) !P}$	
$\text{NC-ALPHA} \frac{P =_{\alpha} Q \quad Q \xrightarrow[S]{\mu} B \quad B =_{\alpha} A}{P \xrightarrow[S]{\mu} A}$	

Table 3.9: The late semantics of the spi calculus (with constraints)

We can now show that if $P \xrightarrow[S]{\mu} A$ and σ is a substitution such that $S\sigma \subseteq N$, then $P\sigma \xrightarrow[S\sigma]{\mu\sigma} A\sigma$. We first need the following lemma.

Lemma 23:

Let $\phi \in F$ and $\sigma : N \rightarrow M$. If $\mathbf{e}(\phi) = \mathbf{true}$ then

$$\mathbf{e}(\phi\sigma) = \mathbf{true} \iff \forall x \in \mathbf{nc}(\phi) : x\sigma \in N$$

Moreover, if $\mathbf{e}(\phi\sigma) = \mathbf{true}$ then $\mathbf{nc}(\phi\sigma) = \mathbf{nc}(\phi)\sigma$.

PROOF

By induction on ϕ and thanks to Lemma 22. ■

Lemma 24:

Let $P \in P$. Assume that $P \xrightarrow[S]{\mu} A$ and let $\sigma : N \rightarrow M$ a substitution such that $S\sigma \subseteq N$. Then $P\sigma \xrightarrow[S\sigma]{\mu\sigma} A\sigma$.

PROOF

By induction on $P \xrightarrow[S]{\mu} A$.

3.3 Equivalences

As in the pi calculus, structural congruence is defined to identify terms that “obviously” represent the same entity. In contrast, the notions of bisimulation of the pi calculus are not very relevant for reasoning about cryptographic protocols. Before explaining why, we introduce two new notions of behavioural equivalences, namely may testing equivalence and barbed equivalence, that were shown to be relevant for formulating security properties. We then motivate the notion of environment-sensitive bisimulation.

3.3.1 Structural congruence

Structural congruence enables manipulation of term structure. Before defining this notion, we first define the notion of *contexts* and *congruence*.

Definition 45 (process contexts).

A process context $C[\cdot]$ is a process with a *hole* somewhere. The contexts are described by the grammar of Table 3.10.

$$\begin{array}{l}
C[\cdot] ::= [\cdot] \\
| E(x).C[\cdot] \mid \bar{E}(F).C[\cdot] \\
| \phi C[\cdot] \\
| C[\cdot] + P \mid P + C[\cdot] \\
| C[\cdot] \mid P \mid P \mid C[\cdot] \\
| (vz)C[\cdot] \\
| !C[\cdot]
\end{array}$$

Table 3.10: The process contexts of the spi calculus

$$\begin{array}{l}
\text{ALPHA } \frac{P =_{\alpha} Q}{P \equiv Q} \quad \text{SUM-ZERO } \frac{}{P + 0 \equiv P} \quad \text{SUM-COMM } \frac{}{P + Q \equiv Q + P} \\
\text{SUM-ASSOC } \frac{}{(P + Q) + R \equiv P + (Q + R)} \quad \text{PAR-ZERO } \frac{}{P \mid 0 \equiv P} \\
\text{PAR-COMM } \frac{}{P \mid Q \equiv Q \mid P} \quad \text{PAR-ASSOC } \frac{}{(P \mid Q) \mid R \equiv P \mid (Q \mid R)} \\
\text{NEW-SWAP } \frac{}{(vx)(vy)P \equiv (vy)(vx)P} \quad \text{NEW-ZERO } \frac{}{(vx) \mathbf{0} \equiv \mathbf{0}} \\
\text{NEW-SCOPE } \frac{}{((vx)P) \mid Q \equiv (vx)(P \mid Q)} \quad x \notin \text{fn}(Q) \quad \text{BANG } \frac{}{!P \equiv P \mid !P}
\end{array}$$

Table 3.11: Axioms of structural congruence

If $C[\cdot]$ is a context and P is a process, we write $C[P]$ for the process obtained by replacing the hole $[\cdot]$ in $C[\cdot]$ by P . Note that some free names of P may be bound in $C[P]$.

Definition 46 (congruence).

An equivalence relation \mathcal{R} on processes is a *congruence* if $(P, Q) \in \mathcal{R}$ implies $(C[P], C[Q]) \in \mathcal{R}$ for every context $C[\cdot]$.

We can now define the notion of *structural congruence*.

Definition 47 (structural congruence).

Structural congruence, \equiv , is the smallest congruence on processes that satisfies the axioms of Table 3.11.

We extend the notion of structural congruence to agents. It is the smallest equivalence relation on agents that contains \equiv and that satisfies:

1. If $A =_{\alpha} B$ then $A \equiv B$
2. If $P \equiv Q$ then $(x)P \equiv (x)Q$
3. If $P \equiv Q$ then $(v\bar{z}) \langle M \rangle P \equiv (v\bar{z}) \langle M \rangle Q$ provided that $\{\bar{z}\} \subseteq n(M)$

Structural congruence is preserved by the reduction relation, i.e.

Theorem 4:

1. If $P \equiv Q$ and $P \xrightarrow{\mu} A$ then there exists B such that $A \equiv B$ and $Q \xrightarrow{\mu} B$.
2. If $P \equiv Q$ and $P \xrightarrow[S]{\mu} A$ then there exists B such that $A \equiv B$ and $Q \xrightarrow[S]{\mu} B$. ◇

3.3.2 Testing equivalence, barbed equivalence

Testing equivalence relates processes that reveal the same information to observers. This notion of equivalence was first defined in [106] and adapted to mobile processes in [34]. As in [9], we do not consider a distinct “success” action ω for tests. As it was argued, this is only a superficial difference and it can be shown that the following definition is a version of De Nicola and Hennessy’s may-testing equivalence. A more faithful instantiation of may-testing equivalence to the spi calculus can be found in [36].

We say that a process P exhibits a barb β , and write $P \downarrow_{\beta}$, if there exists an agent A such that $P \xrightarrow{\beta} A$. The process P may eventually exhibit a barb β , written $P \Downarrow_{\beta}$, if there exists P' such that $P \Longrightarrow P'$ and $P' \downarrow_{\beta}$.

A test is a pair (R, β) where R is a process and β a barb. A process P passes the test (R, β) if $P \mid R \downarrow_{\beta}$. We may interpret this as “ P reveals a piece of information to an observer R ”.

Definition 48 (testing equivalence).

Two processes P and Q are testing equivalent, written $P \simeq Q$, if they pass exactly the same tests, i.e. for all tests (R, β) , $P \mid R \downarrow_{\beta}$ if and only if $Q \mid R \downarrow_{\beta}$.

Hence, two processes that are testing equivalent may reveal the same information to observers. From a security point of view, they should then be considered equivalent.

Indeed, testing equivalence has been used in several works (e.g. [7, 9, 8]) to define secrecy properties. The basic idea is to say that P keeps message x secret if $P\{M/x\} \simeq P\{N/x\}$ for any message M and N .

The use of testing equivalence however entails serious difficulties in verification. Indeed, the naive way to prove that two processes are testing equivalent is to consider an arbitrary attacker R and arbitrary sequences of interactions with R .

A notion of equivalence which is easier to work with is barbed equivalence [103]. The idea is the same as for testing equivalence but is based on a notion of step-by-step simulation between processes.

Definition 49 (weak barbed equivalence).

A symmetric relation $\mathcal{R} \subseteq \mathbf{P} \times \mathbf{P}$ is a *weak barbed bisimulation* if for all $(P, Q) \in \mathcal{R}$ we have

1. for each P' , if $P \longrightarrow P'$ then there exists Q' such that $Q \Longrightarrow Q'$ and $(P', Q') \in \mathcal{R}$, and
2. for each β , if $P \downarrow_\beta$ then $Q \Downarrow_\beta$.

Weak barbed bisimilarity, written \cong , is the largest weak barbed bisimulation relation, i.e. $P \cong Q$ if and only if there exists a weak barbed bisimulation \mathcal{R} such that $(P, Q) \in \mathcal{R}$.

Two processes P and Q are *weak barbed equivalent*, written $P \cong Q$, if for all R we have $P | R \cong Q | R$.

Observe that the definition of weak barbed bisimilarity does not require to investigate arbitrary reduction sequences. One has instead to consider single transitions.

Barbed equivalence is a sound proof technique for showing testing equivalence. Indeed,

Lemma 25:

If $P \cong Q$ then $P \simeq Q$.

PROOF

Let (R, β) a test that P passes.

We have $P | R = P_0 \longrightarrow P_1 \longrightarrow \dots \longrightarrow P_n \downarrow_\beta$.

Since $P_0 = P | R \cong Q | R = Q_0$, there exists Q_1 such that $Q_0 \Longrightarrow Q_1$ and $P_1 \cong Q_1$.

By induction on n , we show the existence of Q_1, \dots, Q_n such that for $0 \leq i < n$ we have $Q_i \Longrightarrow Q_{i+1}$ and for $0 \leq i \leq n$, $P_i \cong Q_i$.

Since $P_n \cong Q_n$ and $P_n \downarrow_\beta$, we have $Q_n \Downarrow_\beta$.

So we have $Q | R = Q_0 \Longrightarrow Q_1 \Longrightarrow \dots \Longrightarrow Q_n \Downarrow_\beta$.

Hence Q passes the test (R, β) . ■

We can reformulate the notion of total secrecy using barbed equivalence: P keeps message x as totally secret if $P\{^M/x\} \cong P\{^N/x\}$ for any messages M and N . This yields to a stronger notion of secrecy since barbed equivalence is stronger than testing equivalence. Eijiro Sumii even argued in [131] that testing equivalence was too weak to define total secrecy and that barbed equivalence should be preferred.

It is also possible to define a stronger version of barbed bisimulation and barbed equivalence that takes into account the number of internal steps of computation:

Definition 50 (strong barbed equivalence).

A symmetric relation $\mathcal{R} \subseteq \mathbf{P} \times \mathbf{P}$ is a *strong barbed bisimulation* if for all $(P, Q) \in \mathcal{R}$ we have

1. for each P' , if $P \longrightarrow P'$ then there exists Q' such that $Q \longrightarrow Q'$ and $(P', Q') \in \mathcal{R}$, and
2. for each β , if $P \downarrow_\beta$ then $Q \downarrow_\beta$.

Strong barbed bisimilarity, written \sim , is the largest strong barbed bisimulation relation, i.e. $P \sim Q$ if and only if there exists a strong barbed bisimulation \mathcal{R} such that $(P, Q) \in \mathcal{R}$.

Two processes P and Q are *strong barbed equivalent*, written $P \sim Q$, if for all R we have $P \mid R \sim Q \mid R$.

Clearly we have that $P \sim Q$ implies $P \cong Q$ and that $P \sim Q$ implies $P \cong Q$.

For the reasons already given in Chapter 2, we will mainly focus our attention on strong versions of equivalences. In the sequel, we omit the qualifier “strong” when referring to strong barbed bisimulation or strong barbed equivalence.

3.3.3 Bisimulations

Even if a quantification over arbitrary sequences of interactions has been removed in the definition of barbed bisimulation, it is difficult to prove directly barbed equivalence because of the infinite quantification over arbitrary observers R .

The idea of the bisimulation proof technique is to make this explicit quantification over arbitrary observers implicit and to simulate step-by-step not only internal transitions but also observable actions (e.g. inputs and outputs).

If we rephrase in the context of the spi calculus the definition of early bisimulation of the pi calculus, this gives:

Definition 51 (early bisimulation).

A symmetric relation $\mathcal{R} \subseteq P \times P$ is an *early bisimulation* if for all $(P, Q) \in \mathcal{R}$

1. whenever $P \longrightarrow P'$ then there exists Q' such that $Q \longrightarrow Q'$ and $(P', Q') \in \mathcal{R}$
2. whenever $P \xrightarrow{a} (x)P'$ then then for all $M \in \mathbf{M}$, there exists Q' such that $Q \xrightarrow{a} (x)Q'$ and $(P'\{M/x\}, Q'\{M/x\}) \in \mathcal{R}$
3. whenever $P \xrightarrow{\bar{a}} (\nu\bar{z}) \langle M \rangle P'$ (with $\bar{z} \cap \text{fn}(P, Q) = \emptyset$) then there exists Q' such that $Q \xrightarrow{\bar{a}} (\nu\bar{z}) \langle M \rangle Q'$ and $(P', Q') \in \mathcal{R}$.

Early bisimilarity, written \sim_e , is the largest early bisimulation.

If, in the pi calculus, we have that $P \sim_e Q$ if and only if $P \sim Q$ (see Theorem 2.2.9 of [129]), this result does not hold anymore in the spi calculus. Indeed, we only have that $P \sim_e Q$ implies $P \sim Q$ (as stated in [7] for example).

The other direction does not hold because \sim_e is a rather fine-grained equivalence for the spi calculus. For instance, it discriminates between the processes $P(M)$ and $P(N)$ (where $P(M) := (\nu k) \bar{c} \langle \text{Enc}_k^s M \rangle. \mathbf{0}$) whereas one would wish to equate these processes since they send a message encrypted under keys that are never disclosed. It was indeed shown in [7] that $P(M) \sim P(N)$ suggesting that barbed equivalence is coarse-grained enough for interesting cryptographic applications.

This example suggests that a relevant notion of bisimulation for reasoning about cryptographic protocols should consider some pair of messages (resulting from output actions) indistinguishable. Indeed, in the above example, the two messages $\text{Enc}_k^s M$ and $\text{Enc}_k^s N$ should be considered as indistinguishable since the key k is never disclosed to the environment. However, if we imagine a similar protocol that acts first like P but then discloses the secret key k to the environment (for example Q where $Q(M) := (\nu k) \bar{c} \langle \text{Enc}_k^s M \rangle. \bar{c} \langle k \rangle. \mathbf{0}$), then we note that two messages that were first indistinguishable might suddenly become distinguishable. Thus the two messages $\text{Enc}_k^s M$ and $\text{Enc}_k^s N$ become distinguishable at the moment the secret key k is disclosed. In short, the notion of indistinguishability is sensitive to future events.

If we now think about input actions, the values that are to be considered are not arbitrary; they are generated by the environment with the knowledge that has been acquired until a certain point. Thus, if we think about $R(M) := (\nu k) \bar{c} \langle \text{Enc}_k^s M \rangle. c(x). \bar{c} \langle k \rangle. c(y). \mathbf{0}$, then the possible values for the first input action exclude any message that contains k in another

form than $\text{Enc}_k^s M$ (thus $\text{Enc}_k^s M$ is allowed whereas k is forbidden) whereas the possible values for the second input action are not restricted. We note here that the set of possible input values grows with time as the environment intercepts messages and discovers values that were previously secret.

Based on these observations, Abadi and Gordon devised in [8] that “a definition of bisimulation for cryptographic protocols should explain what outputs are indistinguishable for the environment, and what inputs the environment can generate at any point in time” and introduced the notion of *environment-sensitive bisimulation*. This idea has been implemented in the spi calculus in different styles:

- Abadi and Gordon proposed *framed* bisimulation [8]. The environment knowledge is represented by a *frame-theory* pair that accompanies every bisimulation pair. The frame is the set of names (channels, keys) that the environment has learnt so far, while the theory is the set of non-name data items received from the pair of processes during the bisimulation game that the environment must consider indistinguishable, because it has no means (yet) to tell the difference. It was shown that framed bisimilarity is strictly finer than barbed equivalence when pairing is in the language.
- Boreale, De Nicola and Pugliese proposed another notion of bisimulation that they call environment-sensitive bisimulation [36]. In this variant, each process of a bisimulation pair comes with an environment which roughly lists the messages the process has received in the past. An explicit condition of equivalence on environments is imposed to implement the indistinguishability relation. They proved that this notion of bisimilarity is sound w.r.t. barbed equivalence and also complete for a large class of processes (the class of *structurally image finite processes*). This notion of bisimulation is sometimes known as *alley* bisimulation, following Borgström and Nestmann naming.
- Elkjaer, Höhle, Hüttel and Overgaard introduced *fenced* bisimulation as an approximation of framed bisimulation by removing one of its infinitary quantification [70].
- Borgström and Nestmann compared the above notions of bisimilarity in [40, 41]. They pointed out that fenced bisimulation was not complete w.r.t. framed bisimulation. To remedy incompleteness of

framed bisimilarity, they also proposed a new notion of bisimulation called *hedged* bisimulation that is defined in the style of framed bisimulation. They have shown that this new proposal coincides with alley bisimilarity and thus also with barbed equivalence.

Thus the two notions of bisimilarity to be preferred are alley bisimilarity and hedged bisimilarity. Since these two definitions are equivalent, it is mainly a matter of taste to choose between them. As we consider that hedges are more intuitive to represent the environment knowledge, we are going to work with hedged bisimilarity in the following.

Conclusion

We have reviewed the syntax and the operational semantics of the spi calculus, an extension of the pi calculus with cryptographic primitives. We have defined an alternative labelled transitions system that collects the name constraints for the transitions to take place. We have presented two notions of equivalence — may testing and barbed equivalence — and shown how one can express security properties with them. We have discussed on the notion of bisimulation and explained why standard notion of bisimulations inherited from the pi calculus are not very relevant when dealing with security protocols. This has led us to present informally the notion of environment-sensitive bisimulation which has been implemented in several ways in the spi calculus.

Apart from the equational style, cryptographic protocols in the spi calculus are analysed by control flow analysis, trace analysis, reachability analysis and type systems. [1, 3] define type systems for the spi calculus to guarantee secrecy properties. A generalisation of this approach is presented in [4, 5] and a simple protocol checker based on Prolog rules is given. In [13, 12], a mini spi calculus is presented. Within this setting, the task of verifying secrecy and authenticity properties of cryptographic protocols relying on symmetric shared keys is expressed as a reachability problem and a decision procedure is given for finite terms. It is possible to verify authentication and secrecy properties of a protocol by analysing the traces generated by the corresponding spi calculus system. Based on this approach, [33] develops symbolic techniques to avoid state explosion inherent to this method. A method, which is shown to be complete for the considered class of properties, is given to carry out trace analysis directly on the symbolic model. In [32, 30], static analysis technology and control flow analysis is applied to variants of the spi calculus (LYSA, ν SPI) to verify certain security properties of security protocols.

Chapter 4

Representing environment knowledge as hedges

As we have seen, the classical notion of bisimulation used in the pi calculus is too fine-grained for the spi calculus. The reason is that requiring an exact match between observable actions is too strong in a cryptographic context where, for example, $\text{Enc}_k^s M$ and $\text{Enc}_k^s N$ need to be identified as long as k is unknown to the observer. To be able to have a cryptographic aware equivalence between actions, bisimulations are extended with structures (e.g. frame-theory pairs, hedges, ...) that explicitly keep track of the identities between messages. In this chapter, we focus on the structure of *hedges* to represent the environment knowledge about processes.

Definition 52 (Hedge).

A *hedge* is a finite subset of $M \times M$. The set of hedges is denoted H .

The results presented afterwards are a generalisation of the hedge theory presented in [40, 41] to the message language defined previously (which thus includes non atomic keys, asymmetric cryptography and hashing). We have formalised all the following definitions and results in the proof assistant `coq` [135, 26]. The formalisation uses the finite set library [71] and the CoLoR library [29].

4.1 Synthesis

4.1.1 Definition

The *synthesis* of a hedge h is the set of message pairs that can be constructed from the knowledge of h . For example, if, on one hand M and N are identified (i.e. $(M, N) \in h$), and on the other hand K and L are identified (i.e.

$(K, L) \in h$) then $\text{Enc}_K^s M$ and $\text{Enc}_L^s N$ are also identified (by applying the shared-key encryption algorithm component-wise).

Definition 53 ($\mathcal{S}(h)$).

If h is a hedge, the *synthesis* $\mathcal{S}(h)$ of h is the smallest subset of $M \times M$ satisfying the following rules:

$$\begin{aligned} & \text{SYN-INC} \frac{(M, N) \in h}{(M, N) \in \mathcal{S}(h)} \\ & \text{SYN-ENC-S} \frac{(M_1, N_1) \in \mathcal{S}(h) \quad (M_2, N_2) \in \mathcal{S}(h)}{(\text{Enc}_{M_2}^s M_1, \text{Enc}_{N_2}^s N_1) \in \mathcal{S}(h)} \\ & \text{SYN-ENC-A} \frac{(M_1, N_1) \in \mathcal{S}(h) \quad (M_2, N_2) \in \mathcal{S}(h)}{(\text{Enc}_{M_2}^a M_1, \text{Enc}_{N_2}^a N_1) \in \mathcal{S}(h)} \\ & \text{SYN-PAIR} \frac{(M_1, N_1) \in \mathcal{S}(h) \quad (M_2, N_2) \in \mathcal{S}(h)}{((M_1 \cdot M_2), (N_1 \cdot N_2)) \in \mathcal{S}(h)} \\ & \text{SYN-OP} \frac{(M, N) \in \mathcal{S}(h)}{(\text{op}(M), \text{op}(N)) \in \mathcal{S}(h)} \text{ op} \in \{\text{pub}, \text{priv}, \text{H}\} \end{aligned}$$

Example 5

Let $h := \{(a, b), (k, k), (\text{Enc}_k^s l, \text{Enc}_k^s l), (\text{Enc}_i^s c, \text{Enc}_i^s d)\}$.

Then we have $(\text{Enc}_k^s a, \text{Enc}_k^s b) \in \mathcal{S}(h)$.

*

The following lemma is a simple but useful result.

Lemma 26:

Let $h \in \mathbf{H}$, $a \in N$ and $N \in M$. Then

$$(a, N) \in h \iff (a, N) \in \mathcal{S}(h) \quad \square$$

In the general case, the synthesis of a hedge is not a hedge since it is infinite.

Lemma 27:

If $h \neq \emptyset$, then $\mathcal{S}(h)$ is infinite.

□

4.1.2 Comparing hedges power

To compare the knowledge of two hedges, we define the following pre-order and the induced equivalence relation.

Definition 54 ($<_H, \geq_H$).

Let g, h be two hedges. We say that

- g is *less powerful than* h —written $g <_H h$ — if $\mathcal{S}(g) \subseteq \mathcal{S}(h)$.
- g is *as powerful as* h —written $g \geq_H h$ — if $\mathcal{S}(g) = \mathcal{S}(h)$.

Example 6

Let $g_1 := \{(a, a)\}$ and $h_1 := \{(a, a), (b, b)\}$. Then $g_1 <_H h_1$.

Let $g_2 := \{(a, a), (H(a), H(a))\}$ and $h_2 := \{(a, a)\}$. Then $g_2 \geq_H h_2$. *

Lemma 28:

1. $<_H$ is a partial preorder on H .
2. \geq_H is the equivalence relation on H induced by $<_H$. □

The next lemma gives a characterisation of $<_H$.

Lemma 29:

Let $g, h \in H$. Then

$$g <_H h \iff g \subseteq \mathcal{S}(h)$$

PROOF

\Rightarrow Since $g \subseteq \mathcal{S}(g) \subseteq \mathcal{S}(h)$.

\Leftarrow By rule induction on $(M, N) \in \mathcal{S}(h)$. ■

Corollary 1:

Let $g, g', h, h' \in H$.

1. If $g \subseteq h$ then $g <_H h$.
2. If $g <_H h$ and $g' <_H h$ then $g \cup g' <_H h$.
3. If $g <_H h$ and $g' <_H h'$ then $g \cup g' <_H h \cup h'$. ♠

4.2 Analysis

4.2.1 Weak analysis

The *analysis* of a hedge h is the set of message pairs that can be deconstructed from the knowledge of h . For example, if, on one hand $\text{Enc}_K^s M$ and $\text{Enc}_L^s N$ are identified, and on the other hand K and L are identified, then M and N are also identified (by applying the shared key decryption algorithm component-wise). Unfortunately, the analysis can not be defined directly (see Section 4.2.2) so we introduce an intermediate notion called *weak analysis* that will help us define the analysis.

Definition 55 (analz(h)).

If h is a hedge, the *weak analysis* $\text{analz}(h)$ of h is the smallest subset of $\mathbf{M} \times \mathbf{M}$ satisfying the following rules:

$$\begin{array}{c}
 \text{ANA-INC} \frac{(M, N) \in h}{(M, N) \in \text{analz}(h)} \\
 \\
 \text{ANA-DEC-S} \frac{(\text{Enc}_{M_2}^s M_1, \text{Enc}_{N_2}^s N_1) \in \text{analz}(h) \quad (M_2, N_2) \in \mathcal{S}(h)}{(M_1, N_1) \in \text{analz}(h)} \\
 \\
 \text{ANA-DEC-A} \frac{\begin{array}{c} (\text{Enc}_{M_2}^a M_1, \text{Enc}_{N_2}^a N_1) \in \text{analz}(h) \\ \text{inv}(M_2) = M'_2 \in \mathbf{M} \quad \text{inv}(N_2) = N'_2 \in \mathbf{M} \\ (M'_2, N'_2) \in \mathcal{S}(h) \end{array}}{(M_1, N_1) \in \text{analz}(h)} \\
 \\
 \text{ANA-FST} \frac{((M_1 \cdot M_2), (N_1 \cdot N_2)) \in \text{analz}(h)}{(M_1, N_1) \in \text{analz}(h)} \\
 \\
 \text{ANA-SND} \frac{((M_1 \cdot M_2), (N_1 \cdot N_2)) \in \text{analz}(h)}{(M_2, N_2) \in \text{analz}(h)}
 \end{array}$$

Example 7

Let $h := \{(a, b), (k, k), (\text{Enc}_k^s l, \text{Enc}_k^s l), (\text{Enc}_c^s c, \text{Enc}_d^s d)\}$.

Then we have $\text{analz}(h) = h \cup \{(l, l)\}$. *

The weak analysis of a hedge is a hedge.

Lemma 30:

If h is a hedge, then $\text{analz}(h)$ is finite (and thus is a hedge). □

The weak analysis of a hedge is more powerful than the hedge itself.

Lemma 31:

Let $h \in \mathbf{H}$. Then $h <_{\mathbf{H}} \text{analz}(h)$.

PROOF

By Corollary 1 since $h \subseteq \text{analz}(h)$. ■

The function $h \mapsto \text{analz}(h)$ is compatible with \subseteq and $<_{\mathbf{H}}$.

Lemma 32:

Let $g, h \in \mathbf{H}$.

1. If $g \subseteq h$ then $\text{analz}(g) \subseteq \text{analz}(h)$.
2. If $g <_{\mathbf{H}} h$ then $\text{analz}(g) <_{\mathbf{H}} \text{analz}(h)$.

PROOF

1. By rule induction on $(M, N) \in \text{analz}(g)$ and thanks to Corollary 1.
2. We prove that if $(M, N) \in \text{analz}(g)$ then $(M, N) \in \mathcal{S}(\text{analz}(h))$ by rule induction on $(M, N) \in \text{analz}(g)$. Then, by Lemma 29, we get $\text{analz}(g) <_{\mathbf{H}} \text{analz}(h)$. ■

4.2.2 Analysis

In rules ANA-DEC-S and ANA-DEC-A, it is checked whether the decryption keys can be constructed from the original hedge. Intuitively, one would want instead to check whether they can be constructed from the hedge analysed so far; i.e. one would want to replace in the premise of these two rules $\mathcal{S}(h)$ by $\mathcal{S}(\text{analz}(h))$. Unfortunately, the resulting definition would not be well-founded. So, we instead define the analysis of h as being the smallest set that contains h and which is closed under weak analysis.

Definition 56.

Let $h, h' \in \mathbf{H}$. We define the two following predicates:

- $\text{AnaCond}(h, h') := h \subseteq h' \wedge \text{analz}(h') \subseteq h'$
- $\text{IsAna}(h, h') := \begin{cases} \text{AnaCond}(h, h') \\ \forall h'' \in \mathbf{H} : \text{AnaCond}(h, h'') \implies h' \subseteq h'' \end{cases}$

Example 8

Let $h := \{(k, k), (\text{Enc}_k^s a, \text{Enc}_k^s b)\}$ and $h' := h \cup \{(a, b)\}$.

Then we have $h \subseteq h'$ and $\text{analz}(h') = h' \subseteq h'$.

So we have $\text{AnaCond}(h, h')$. *

The remaining of this section is devoted to show the following theorem.

Theorem 5 (existence and uniqueness of analysis):

Let $h \in \mathbf{H}$. Then there exists a unique $\mathcal{A}(h) \in \mathbf{H}$ —called the analysis of h —such that $\text{IsAna}(h, \mathcal{A}(h))$. ◇

If this theorem holds, it is obvious that h is less powerful than $\mathcal{A}(h)$.

Lemma 33:

Let $h \in \mathbf{H}$. Then $h <_{\mathbf{H}} \mathcal{A}(h)$.

PROOF

Since $h \subseteq \mathcal{A}(h)$ and by Corollary 1. ■

The uniqueness of the analysis, in case of existence, is obvious.

PROOF (THEOREM 5, UNIQUENESS IN CASE OF EXISTENCE)

Let h_1 and h_2 such that $\text{IsAna}(h, h_1)$ and $\text{IsAna}(h, h_2)$.

Since $\text{AnaCond}(h, h_2)$, we have $h_1 \subseteq h_2$.

Symmetrically, we have $h_2 \subseteq h_1$.

Thus $h_1 = h_2$. ■

The constructive proof of the existence of the analysis relies on the following lemma.

Lemma 34:

Let $h, h' \in \mathbf{H}$.

1. If $\text{analz}(h) = h$, then $\text{IsAna}(h, h)$.
2. If $\text{IsAna}(\text{analz}(h), h')$, then $\text{IsAna}(h, h')$.

PROOF

1. Assume that $\text{analz}(h) = h$.

We have $\text{AnaCond}(h, h)$ since $h \subseteq h$ and $\text{analz}(h) \subseteq h$.

Assume that there is h' such that $\text{AnaCond}(h, h')$.

By definition, we have $h \subseteq h'$.

Thus $\text{IsAna}(h, h)$.

2. Assume that $IsAna(\text{analz}(h), h')$.

We have $h \subseteq \text{analz}(h) \subseteq h'$.

Moreover, we have $\text{analz}(h') \subseteq h'$.

So $AnaCond(h, h')$.

Let h'' such that $AnaCond(h, h'')$.

We have to show that $h' \subseteq h''$.

It suffices to show that $AnaCond(\text{analz}(h), h'')$.

We have by hypothesis $\text{analz}(h'') \subseteq h''$.

Moreover, $h \subseteq h''$.

By Lemma 32, we have thus $\text{analz}(h) \subseteq \text{analz}(h'') \subseteq h''$.

So $AnaCond(\text{analz}(h), h'')$ and $h' \subseteq h''$. ■

Example 9

Let $h := \{(k, k), (\text{Enc}_k^s a, \text{Enc}_k^s b)\}$ and $h' := h \cup \{(a, b)\}$.

Since $\text{analz}(h') = h'$, we have $IsAna(h', h')$.

Since $\text{analz}(h) = h'$, we have $IsAna(h, h')$. *

The previous lemma suggests that for computing the analysis of h , it is sufficient to iterate the weak analysis until reaching a fixpoint. The main difficulty is to show that a *finite* number of iterations is sufficient. In other words, we are going to show that:

Lemma 35:

Let $h \in \mathbf{H}$. Then there exists $n_0 \in \mathbb{N}$ such that $\mathcal{A}(h) = \text{analz}^{n_0}(h)$, where $\text{analz}^n(h)$ is defined by induction on n as

$$\begin{aligned} \text{analz}^0(h) &:= h \\ \text{analz}^{n+1}(h) &:= \text{analz}(\text{analz}^n(h)) \end{aligned} \quad \square$$

To show this result, we first give an alternative definition of $\text{analz}(h)$ that will ease the study of the sequence $(\text{analz}^i(h))_{i \in \mathbb{N}}$. We are going to show that this sequence is decreasing w.r.t. to a well-founded order. So necessarily it reaches a fixpoint.

Definition 57.

Let $h \in \mathbf{H}$ and $M, N \in \mathbf{M}$. We say that the pair (M', N') results from the analysis of (M, N) against h —written $(M, N) \vdash_{\mathcal{A}}^h (M', N')$ —if a proof of $(M, N) \vdash_{\mathcal{A}}^h (M', N')$ can be derived from the following deduction system:

$$\begin{array}{c}
\text{A1} \frac{}{((M_1 \cdot M_2), (N_1 \cdot N_2)) \vdash_{\mathcal{A}}^h (M_1, N_1)} \\
\text{A2} \frac{}{((M_1 \cdot M_2), (N_1 \cdot N_2)) \vdash_{\mathcal{A}}^h (M_2, N_2)} \\
\text{A3} \frac{(M_1, N_1) \vdash_{\mathcal{A}}^h (M, N)}{((M_1 \cdot M_2), (N_1 \cdot N_2)) \vdash_{\mathcal{A}}^h (M, N)} \\
\text{A4} \frac{(M_2, N_2) \vdash_{\mathcal{A}}^h (M, N)}{((M_1 \cdot M_2), (N_1 \cdot N_2)) \vdash_{\mathcal{A}}^h (M, N)} \\
\text{A5} \frac{(M_2, N_2) \in \mathcal{S}(h)}{(\text{Enc}_{M_2}^s M_1, \text{Enc}_{N_2}^s N_1) \vdash_{\mathcal{A}}^h (M_1, N_1)} \\
\text{A6} \frac{(M_2, N_2) \in \mathcal{S}(h) \quad (M_1, N_1) \vdash_{\mathcal{A}}^h (M, N)}{(\text{Enc}_{M_2}^s M_1, \text{Enc}_{N_2}^s N_1) \vdash_{\mathcal{A}}^h (M, N)} \\
\text{A7} \frac{M'_2 = \text{inv}(M_2) \in \mathbf{M} \quad N'_2 = \text{inv}(N_2) \in \mathbf{M} \quad (M'_2, N'_2) \in \mathcal{S}(h)}{(\text{Enc}_{M_2}^a M_1, \text{Enc}_{N_2}^a N_1) \vdash_{\mathcal{A}}^h (M_1, N_1)} \\
\text{A8} \frac{M'_2 = \text{inv}(M_2) \in \mathbf{M} \quad N'_2 = \text{inv}(N_2) \in \mathbf{M} \quad (M'_2, N'_2) \in \mathcal{S}(h) \quad (M_1, N_1) \vdash_{\mathcal{A}}^h (M, N)}{(\text{Enc}_{M_2}^a M_1, \text{Enc}_{N_2}^a N_1) \vdash_{\mathcal{A}}^h (M, N)}
\end{array}$$

From the above definition, it is clear that the set of pairs (M', N') such that $(M, N) \vdash_{\mathcal{A}}^h (M', N')$ is decidable.

Example 10

Let $h := \{(a, b), (k, k), (\text{Enc}_k^s l, \text{Enc}_k^s l), (\text{Enc}_i^s c, \text{Enc}_i^s d)\}$.

Then we have $(\text{Enc}_k^s l, \text{Enc}_k^s l) \vdash_{\mathcal{A}}^h (l, l)$ since $(k, k) \in \mathcal{S}(h)$.

But there is no (M, N) such that $(\text{Enc}_i^s c, \text{Enc}_i^s d) \vdash_{\mathcal{A}}^h (M, N)$ since $(l, l) \notin \mathcal{S}(h)$. *

The next lemma gives an alternative definition of the weak analysis.

Lemma 36:

Let $h \in \mathbf{H}$. Then $\text{analz}(h) = h \cup \bigcup_{(M,N) \in h} \{(M', N') \mid (M, N) \vdash_{\mathcal{A}}^h (M', N')\}$.

PROOF

We first show by a simple induction on M that if $(M, N) \in \text{analz}(h)$ and $(M, N) \vdash_{\mathcal{A}}^h (M', N')$ then $(M', N') \in \text{analz}(h)$.

Thus, since $h \subseteq \text{analz}(h)$, we have

$$h \cup \bigcup_{(M,N) \in h} \{(M', N') \mid (M, N) \vdash_{\mathcal{A}}^h (M', N')\} \subseteq \text{analz}(h)$$

Then, let $(M, N) \in \text{analz}(h)$.

We show that $(M, N) \in h \cup \bigcup_{(M,N) \in h} \{(M', N') \mid (M, N) \vdash_{\mathcal{A}}^h (M', N')\}$

by rule induction on $(M, N) \in \text{analz}(h)$.

Hence the equality. \blacksquare

The strategy is now to simplify the formula of Lemma 36. First, we can notice from Definition 57 that some message pairs (M, N) are such that there is no (M', N') such that $(M, N) \vdash_{\mathcal{A}}^h (M', N')$.

We characterise precisely these pairs with the following definition.

Definition 58.

Let $h \in \mathbf{H}$ and $M, N \in \mathbf{M}$. We define the predicate *analysable* $(h, (M, N))$ —read (M, N) is analysable in h —with the following deduction rules:

$$\text{AA1} \frac{}{\text{analysable}(h, ((M_1 \cdot M_2), (N_1 \cdot N_2)))}$$

$$\text{AA2} \frac{(M_2, N_2) \in \mathcal{S}(h)}{\text{analysable}(h, (\text{Enc}_{M_2}^s M_1, \text{Enc}_{N_2}^s N_1))}$$

$$\text{AA3} \frac{M'_2 = \text{inv}(M_2) \in \mathbf{M} \quad N'_2 = \text{inv}(N_2) \in \mathbf{M} \quad (M'_2, N'_2) \in \mathcal{S}(h)}{\text{analysable}(h, (\text{Enc}_{M_2}^a M_1, \text{Enc}_{N_2}^a N_1))}$$

Example 11

Let $h := \{(a, b), (k, k), (\text{Enc}_k^s l, \text{Enc}_k^s l), (\text{Enc}_i^s c, \text{Enc}_i^s d)\}$.

Then, since $(k, k) \in \mathcal{S}(h)$, we have *analysable* $(h, (\text{Enc}_k^s l, \text{Enc}_k^s l))$ and since $(l, l) \notin \mathcal{S}(h)$, we do *not* have *analysable* $(h, (\text{Enc}_i^s c, \text{Enc}_i^s d))$. *

The next lemma justifies the name of the predicate.

Lemma 37:

Let $h \in \mathbf{H}$ and $M, N \in \mathbf{M}$. Then

$$\{(M', N') \mid (M, N) \vdash_{\mathcal{A}}^h (M', N')\} = \emptyset \iff \neg \text{analysable}(h, (M, N))$$

PROOF

By case analysis on M and N . ■

Corollary 2:

Let $h \in \mathbf{H}$ and $h' := \{(M, N) \in h \mid \text{analysable}(h, (M, N))\}$. Then

$$\begin{aligned} \bigcup_{(M, N) \in h} \{(M', N') \mid (M, N) \vdash_{\mathcal{A}}^h (M', N')\} \\ = \bigcup_{(M, N) \in h'} \{(M', N') \mid (M, N) \vdash_{\mathcal{A}}^h (M', N')\} \spadesuit \end{aligned}$$

The previous corollary gives us a first simplification. Another possible simplification is to remove from the big union of Lemma 36 the pairs that are *already analysed* in the sense that their contribution to this equality is not relevant.

Definition 59.

Let $h \in \mathbf{H}$ and $M, N \in \mathbf{M}$. We define the predicate *analysed*($h, (M, N)$)—read (M, N) is analysed in h —with the following deduction rules:

$$\begin{aligned} \text{AD1} \quad & \frac{(M_1, N_1) \in h \quad (M_2, N_2) \in h}{\text{analysed}(h, ((M_1 \cdot M_2), (N_1 \cdot N_2)))} \\ \text{AD2} \quad & \frac{(M_2, N_2) \in \mathcal{S}(h) \quad (M_1, N_1) \in h}{\text{analysed}(h, (\text{Enc}_{M_2}^s M_1, \text{Enc}_{N_2}^s N_1))} \\ \text{AD3} \quad & \frac{M'_2 = \text{inv}(M_2) \in \mathbf{M} \quad N'_2 = \text{inv}(N_2) \in \mathbf{M} \quad (M'_2, N'_2) \in \mathcal{S}(h) \quad (M_1, N_1) \in h}{\text{analysed}(h, (\text{Enc}_{M_2}^a M_1, \text{Enc}_{N_2}^a N_1))} \end{aligned}$$

Example 12

Let $h := \{(a, b), (k, k), (l, l), (\text{Enc}_k^s l, \text{Enc}_k^s l), (\text{Enc}_l^s c, \text{Enc}_l^s d)\}$.

Then *analysed*($h, (\text{Enc}_k^s l, \text{Enc}_k^s l)$) holds since we have $(k, k) \in \mathcal{S}(h)$ and $(l, l) \in \mathcal{S}(h)$.

Since $(l, l) \in \mathcal{S}(h)$, we have *analysable*($h, (\text{Enc}_l^s c, \text{Enc}_l^s d)$). However since $(c, d) \notin \mathcal{S}(h)$, we do not have *analysed*($h, (\text{Enc}_l^s c, \text{Enc}_l^s d)$). *

The next lemma justifies a new simplification:

Lemma 38:

Let $h \in \mathbf{H}$.

1. Let M, N such that $\text{analysed}(h, (M, N))$.

Let M', N' such that $(M, N) \vdash_{\mathcal{A}}^h (M', N')$.

Then either $(M', N') \in h$ or there exists $(M_0, N_0) \in h$ such that $(M_0, N_0) \vdash_{\mathcal{A}}^h (M', N')$ and $\neg \text{analysed}(h, (M_0, N_0))$.

2. Let $h' := \{(M, N) \in h \mid \neg \text{analysed}(h, (M, N))\}$. Then

$$\text{analz}(h) = h \cup \bigcup_{(M, N) \in h'} \left\{ (M', N') \mid (M, N) \vdash_{\mathcal{A}}^h (M', N') \right\}$$

PROOF

1. By induction on M .

2. Follows from previous point. ■

Corollary 3:

Let $h \in \mathbf{H}$. Let $h' := \left\{ (M, N) \in h \mid \left\{ \begin{array}{l} \text{analysable}(h, (M, N)) \\ \neg \text{analysed}(h, (M, N)) \end{array} \right\} \right\}$. Then

$$\text{analz}(h) = h \cup \bigcup_{(M, N) \in h'} \left\{ (M', N') \mid (M, N) \vdash_{\mathcal{A}}^h (M', N') \right\}$$

We now define a measure on hedges.

We first define (in the obvious way) the *size* $\text{size}((M, N))$ of a message pair (M, N) as being the maximal number of constructors in M (depth of the term M seen as a tree) or in N .

If $h \in \mathbf{H}$, we define $w(h)$ the *weight* of h as being the finite multiset of the sizes of those pairs of messages in h that are not analysed in h , i.e.

$$w(h) := \{ \{ \text{size}((M, N)) \mid \neg \text{analysed}(h, (M, N)) \}$$

We consider on the finite multiset of elements of \mathbb{N} the multiset ordering $<$ induced by the order $<$ on \mathbb{N} : it is a well-founded order on the finite multisets of elements of \mathbb{N} (see [17] for example).

We have the following results:

Lemma 39:

Let $h \in \mathbf{H}$. Let $h' := \left\{ (M, N) \in h \mid \left\{ \begin{array}{l} \text{analysable}(h, (M, N)) \\ \neg \text{analysed}(h, (M, N)) \end{array} \right\} \right\}$. Then

1. If $h' = \emptyset$, then $\text{analz}(h) = h$.
2. If $h' \neq \emptyset$, then $w(\text{analz}(h)) < w(h)$.

PROOF

1. Trivial according to Corollary 3.
2. It follows from Corollary 3 and the three following results:
 - (a) If $(M, N) \vdash_A^h (M', N')$ then $\text{size}((M', N')) < \text{size}((M, N))$.
 - (b) If $\text{analysable}(h, (M, N))$ and $(M, N) \in h$ then
 $\text{analysed}(\text{analz}(h), (M, N))$.
 - (c) If $h \subseteq h'$ and $\text{analysed}(h, (M, N))$ then $\text{analysed}(h', (M, N))$. ■

We are now ready to prove the existence of the analysis.

PROOF (THEOREM 5, EXISTENCE)

We proceed by induction on $w(h)$.

Let $h \in \mathbf{H}$.

Let $h' := \left\{ (M, N) \in h \mid \left\{ \begin{array}{l} \text{analysable}(h, (M, N)) \\ \neg \text{analysed}(h, (M, N)) \end{array} \right\} \right\}$. Then

- If $h' = \emptyset$, then by Lemma 39 and Lemma 34, we have $\text{IsAna}(h, h)$ so h is a candidate.
- If $h' \neq \emptyset$, then since $w(\text{analz}(h)) < w(h)$, by induction there exists h' such that $\text{IsAna}(\text{analz}(h), h')$. By Lemma 34, we have $\text{IsAna}(h, h')$ so h' is a candidate. ■

Example 13

Let $h := \{(a, b), (k, k), (\text{Enc}_k^s l, \text{Enc}_k^s l), (\text{Enc}_i^s c, \text{Enc}_i^s d)\}$.

Then

$$\begin{aligned} \text{analz}^0(h) &= h \\ \text{analz}^1(h) &= h \cup \{(l, l)\} \\ \text{analz}^2(h) &= h \cup \{(l, l), (c, d)\} \\ \text{analz}^3(h) &= \text{analz}^2(h) \end{aligned}$$

So $\mathcal{A}(h) = \text{analz}^2(h) = h \cup \{(l, l), (c, d)\}$.

*

4.3 Irreducible hedges

4.3.1 Reducing hedges

A hedge h may contain redundant information. The hedge $reduce(h)$ is obtained from h by removing all the pairs that can also be constructed from h in another way than by immediate membership.

Definition 60 (*reduce*(h)).

Let $h \in \mathbf{H}$.

If $M, N \in \mathbf{M}$, we define the predicate $h \vdash_{\mathcal{S}} (M, N)$ —read (M, N) is *strictly synthesisable* by h — by the rules:

$$\begin{aligned}
 \text{S1} \quad & \frac{(M', N') \in \mathcal{S}(h)}{h \vdash_{\mathcal{S}} (\text{op}(M'), \text{op}(N'))} \quad \text{op} \in \{\text{pub}, \text{priv}, \text{H}\} \\
 \text{S2} \quad & \frac{(M_1, N_1) \in \mathcal{S}(h) \quad (M_2, N_2) \in \mathcal{S}(h)}{h \vdash_{\mathcal{S}} ((M_1 \cdot M_2), (N_1 \cdot N_2))} \\
 \text{S3} \quad & \frac{(M_1, N_1) \in \mathcal{S}(h) \quad (M_2, N_2) \in \mathcal{S}(h)}{h \vdash_{\mathcal{S}} (\text{Enc}_{M_2}^s M_1, \text{Enc}_{N_2}^s N_1)} \\
 \text{S4} \quad & \frac{(M_1, N_1) \in \mathcal{S}(h) \quad (M_2, N_2) \in \mathcal{S}(h)}{h \vdash_{\mathcal{S}} (\text{Enc}_{M_2}^a M_1, \text{Enc}_{N_2}^s N_1)}
 \end{aligned}$$

We define $reduce(h) := \{(M, N) \in h \mid h \not\vdash_{\mathcal{S}} (M, N)\}$.

Example 14

Let $h := \{(a, b), (k, k), (\text{Enc}_k^s l, \text{Enc}_k^s l), (\text{Enc}_i^s c, \text{Enc}_i^s d), (l, l)\}$.

Then $reduce(h) = \{(a, b), (k, k), (l, l), (\text{Enc}_i^s c, \text{Enc}_i^s d)\}$. *

Reducing hedges does not diminish their power.

Lemma 40:

Let $h \in \mathbf{H}$. Then $h \geq_{\mathbf{H}} reduce(h)$.

PROOF

- $reduce(h) <_{\mathbf{H}} h$ because $reduce(h) \subseteq h$ and thanks to Corollary 1.
- Let $(M, N) \in \mathcal{S}(h)$.
We show by induction on M that $(M, N) \in \mathcal{S}(reduce(h))$.
Thus $h <_{\mathbf{H}} reduce(h)$. ■

The following lemma gives two methods for showing that $reduce(g)$ is included in $reduce(h)$.

Lemma 41:

Let $g, h \in \mathbf{H}$.

1. If $g \subseteq h$ and $h <_{\mathbf{H}} g$, then $reduce(g) \subseteq reduce(h)$.
2. If $reduce(g) \subseteq h$ and $h <_{\mathbf{H}} reduce(g)$ then $reduce(g) \subseteq reduce(h)$.

PROOF

1. We show that if $(M, N) \in reduce(g)$ then $(M, N) \in reduce(h)$ by case analysis on M .
2. We show that if $(M, N) \in reduce(g)$ then $(M, N) \in reduce(h)$ by case analysis on M . We also use the fact that $reduce(g) <_{\mathbf{H}} g$. ■

The function $h \mapsto reduce(h)$ is idempotent.

Lemma 42:

Let $h \in \mathbf{H}$. Then $reduce(reduce(h)) = reduce(h)$.

PROOF

We have $reduce(reduce(h)) \subseteq reduce(h)$ by definition.

Since $reduce(h) \subseteq reduce(h)$ and $reduce(h) <_{\mathbf{H}} reduce(h)$ then thanks to Lemma 41, we get $reduce(h) \subseteq reduce(reduce(h))$. ■

The following lemma gives a method for showing that $reduce(g)$ is included in h .

Lemma 43:

Let $g, h \in \mathbf{H}$. If $reduce(g) \geq_{\mathbf{H}} reduce(h)$ then $reduce(g) \subseteq h$.

PROOF

By Lemma 40, we have $g \geq_{\mathbf{H}} reduce(g) \geq_{\mathbf{H}} reduce(h) \geq_{\mathbf{H}} h$.

Since $reduce(g) <_{\mathbf{H}} h$, we have, by Lemma 29, $reduce(g) \subseteq \mathcal{S}(h)$.

Let $(M, N) \in reduce(g)$. Then $(M, N) \in \mathcal{S}(h)$. By case analysis on $(M, N) \in \mathcal{S}(h)$ and since $h <_{\mathbf{H}} g$, we show that $(M, N) \in h$. ■

The function $h \mapsto reduce(h)$ is compatible with $<_{\mathbf{H}}$.

Lemma 44:

Let $g, h \in \mathbf{H}$. If $g <_{\mathbf{H}} h$ then $reduce(g) <_{\mathbf{H}} reduce(h)$.

PROOF

Since $reduce(g) \subseteq g$, we have by Corollary 1 that $reduce(g) <_{\mathbf{H}} g$.

By Lemma 40, we have $h <_{\mathbf{H}} reduce(h)$.

So $reduce(g) <_{\mathbf{H}} reduce(h)$. ■

4.3.2 Irreducible hedges

The *irreducible part* of h is a compact representation of $\mathcal{A}(h)$. We are going to show that it is the smallest hedge which is as powerful as $\mathcal{A}(h)$.

Definition 61 ($\mathcal{I}(h)$).

Let $h \in \mathbf{H}$. We define the *irreducible part* $\mathcal{I}(h)$ of h as $\mathcal{I}(h) := \text{reduce}(\mathcal{A}(h))$.

Example 15

Let $h := \{(a, b), (k, k), (\text{Enc}_k^s l, \text{Enc}_k^s l), (\text{Enc}_i^c c, \text{Enc}_i^c d)\}$.

Then $\mathcal{A}(h) = h \cup \{(l, l), (c, d)\}$.

So $\mathcal{I}(h) = \{(a, b), (k, k), (l, l), (c, d)\}$. *

$\mathcal{I}(h)$ and $\mathcal{A}(h)$ have the same power which is greater than the one of h .

Lemma 45:

Let $h \in \mathbf{H}$. We have $h <_{\mathbf{H}} \mathcal{A}(h) \geq_{\mathbf{H}} \mathcal{I}(h)$.

PROOF

By Lemma 40 and Lemma 33. ■

The analysis of $\mathcal{I}(h)$ is included in the analysis of h .

Lemma 46:

Let $h \in \mathbf{H}$. Then $\mathcal{A}(\mathcal{I}(h)) \subseteq \mathcal{A}(h)$.

PROOF

By definition, we have $\mathcal{I}(h) = \text{reduce}(\mathcal{A}(h)) \subseteq \mathcal{A}(h)$.

Then, by definition of $\mathcal{A}(h)$, we have $\text{analz}(\mathcal{A}(h)) \subseteq \mathcal{A}(h)$.

So $\text{AnaCond}(\mathcal{I}(h), \mathcal{A}(h))$.

Thus, by definition of $\mathcal{A}(\mathcal{I}(h))$ we get $\mathcal{A}(\mathcal{I}(h)) \subseteq \mathcal{A}(h)$. ■

Definition 62.

A hedge h is *irreducible* if $\mathcal{I}(h) = h$.

The irreducible part of a hedge is irreducible.

Theorem 6:

Let $h \in \mathbf{H}$. Then $\mathcal{I}(\mathcal{I}(h)) = \mathcal{I}(h)$, i.e. $\mathcal{I}(h)$ is irreducible.

PROOF

By Lemma 46, we have $\mathcal{A}(\mathcal{I}(h)) \subseteq \mathcal{A}(h)$.

By Lemma 45, we have $\mathcal{A}(h) \geq_{\mathbf{H}} \mathcal{I}(h) <_{\mathbf{H}} \mathcal{A}(\mathcal{I}(h))$.

So by Lemma 41, $\text{reduce}(\mathcal{A}(\mathcal{I}(h))) \subseteq \text{reduce}(\mathcal{A}(h))$, i.e. $\mathcal{I}(\mathcal{I}(h)) \subseteq \mathcal{I}(h)$.

We have $\mathcal{I}(h) = \text{reduce}(\mathcal{A}(h)) \subseteq \mathcal{A}(\text{reduce}(\mathcal{A}(h))) = \mathcal{A}(\mathcal{I}(h))$ by definition of $\mathcal{A}(\mathcal{I}(h))$.

By Lemma 46, we have $\mathcal{A}(\mathcal{I}(h)) \subseteq \mathcal{A}(h)$ so by Corollary 1, we have $\mathcal{A}(\mathcal{I}(h)) <_H \mathcal{A}(h)$.

But, by Lemma 45, we have $\mathcal{A}(h) \geq_H \mathcal{I}(h)$.

So $\mathcal{A}(\mathcal{I}(h)) <_H \mathcal{I}(h)$.

Hence, by Lemma 41, we have $\mathcal{I}(h) \subseteq \text{reduce}(\mathcal{A}(\mathcal{I}(h))) = \mathcal{I}(\mathcal{I}(h))$.

Finally, $\mathcal{I}(\mathcal{I}(h)) = \mathcal{I}(h)$. ■

If g is irreducible and is as powerful as h then g is included in h . This proves that $\mathcal{I}(h)$ is the smallest hedge such that $\mathcal{S}(\mathcal{I}(h)) = \mathcal{S}(\mathcal{A}(h))$.

Lemma 47:

Let $g, h \in H$. If $g \geq_H h$ and g is irreducible then $g \subseteq h$.

PROOF

We have $g \subseteq \mathcal{I}(g) = \text{reduce}(\mathcal{A}(g))$.

By Lemma 40, we have $h \geq_H \text{reduce}(h)$.

Since $g = \mathcal{I}(g)$, we have $g \geq_H \mathcal{I}(g) = \text{reduce}(\mathcal{A}(g))$.

So $\text{reduce}(\mathcal{A}(g)) \geq_H \text{reduce}(h)$.

So by Lemma 43, we have $\text{reduce}(\mathcal{A}(g)) \subseteq h$, i.e. $g = \mathcal{I}(g) \subseteq h$. ■

$\mathcal{I}(h)$ is the minimal seed for $\mathcal{S}(\mathcal{A}(h))$.

Corollary 4:

Let $g, h \in H$. If $g \geq_H \mathcal{A}(h)$ then $\mathcal{I}(h) \subseteq g$.

PROOF

Because $\mathcal{I}(h) \geq_H \mathcal{A}(h) \geq_H g$ and $\mathcal{I}(h)$ is irreducible. ■

Corollary 5:

Let $g, h \in H$. If $g \geq_H h$ and g and h are both irreducibles then $g = h$. ♠

The function $h \mapsto \mathcal{A}(h)$ is compatible with $<_H$.

Lemma 48:

Let $g, h \in H$. If $g <_H h$ then $\mathcal{A}(g) <_H \mathcal{A}(h)$.

PROOF

By Lemma 45, we have $\mathcal{I}(h) \geq_H \mathcal{A}(h)$.

We show by induction on $i \in \mathbb{N}$ that $\forall i \in \mathbb{N} : \text{analz}^i(g) <_H \mathcal{I}(h)$. Then thanks to Lemma 35, we will get $\mathcal{A}(g) <_H \mathcal{I}(h)$.

If $i = 0$, then we have by Lemma 45 that $h <_H \mathcal{I}(h)$.

So $\text{analz}^0(g) = g <_H h <_H \mathcal{I}(h)$.

To show the inductive case, we need the following result:

Lemma:

If $g <_H h$, then $\text{analz}(g) <_H \mathcal{I}(h)$.

PROOF

We show that if $(M, N) \in \text{analz}(g)$ then $(M, N) \in \mathcal{S}(\mathcal{I}(h))$ by rule induction on $(M, N) \in \text{analz}(g)$. So by Lemma 29, we get $\text{analz}(g) <_H \mathcal{I}(h)$. ■

So, for the inductive case, assume that $\text{analz}^i(g) <_H \mathcal{I}(h)$.

With the preceding result, we get $\text{analz}^{i+1}(g) <_H \mathcal{I}(\mathcal{I}(h))$. But by Theorem 6, we have $\mathcal{I}(\mathcal{I}(h)) = \mathcal{I}(h)$.

Thus $\text{analz}^{i+1}(g) <_H \mathcal{I}(h)$. ■

The function $h \mapsto \mathcal{I}(h)$ is compatible with $<_H$.

Lemma 49:

Let $g, h \in H$. If $g <_H h$ then $\mathcal{I}(g) <_H \mathcal{I}(h)$.

PROOF

By Lemma 48 and Lemma 45. ■

Lemma 50:

Let $g, h \in H$. If $g \geq_H h$ then $\mathcal{I}(g) = \mathcal{I}(h)$.

PROOF

If $g \geq_H h$ then by Lemma 49, we have $\mathcal{I}(g) \geq_H \mathcal{I}(h)$.

Since $\mathcal{I}(g)$ and $\mathcal{I}(h)$ are irreducibles, we conclude by Corollary 5 that $\mathcal{I}(g) = \mathcal{I}(h)$. ■

The following theorem gives a way to compute $\mathcal{I}(h \cup g)$ from $\mathcal{I}(h)$ and g .

Theorem 7:

Let $g, h \in H$. Then $\mathcal{I}(\mathcal{I}(h) \cup g) = \mathcal{I}(h \cup g)$.

PROOF

We first show that $\mathcal{I}(\mathcal{I}(h) \cup g) \geq_H \mathcal{I}(h \cup g)$.

We have $h <_H h \cup g$ so by Lemma 49, $\mathcal{I}(h) <_H \mathcal{I}(h \cup g)$.

Similarly, $\mathcal{I}(g) <_H \mathcal{I}(h \cup g)$. But since $g <_H \mathcal{I}(g)$, we have $g <_H \mathcal{I}(h \cup g)$.

So by Corollary 1, we have $\mathcal{I}(h) \cup g <_H \mathcal{I}(h \cup g)$.

So, by Lemma 49, $\mathcal{I}(\mathcal{I}(h) \cup g) <_H \mathcal{I}(\mathcal{I}(h \cup g)) = \mathcal{I}(h \cup g)$.

We have $h <_H \mathcal{I}(h)$ and $g <_H g$, so by Corollary 1, we have $h \cup g <_H \mathcal{I}(h) \cup g$.

So by Lemma 49, $\mathcal{I}(h \cup g) <_H \mathcal{I}(\mathcal{I}(h) \cup g)$.

Finally, $\mathcal{I}(\mathcal{I}(h) \cup g) \geq_H \mathcal{I}(h \cup g)$. So by Corollary 5 and Theorem 6, we conclude that $\mathcal{I}(\mathcal{I}(h) \cup g) = \mathcal{I}(h \cup g)$. ■

4.3.3 Characterisation of irreducible hedges

We now give a characterisation of irreducible hedges. We introduce the notion of *reduced hedges* and show that this notion coincides with irreducible hedges. We will use this characterisation afterwards (in Section 4.4.2).

Definition 63.

A hedge h is *reduced* if it satisfies the following rules:

$$\begin{array}{l}
 \text{R1 } \frac{(\text{op}(M'), \text{op}(N')) \in h}{(M', N') \notin \mathcal{S}(h)} \quad \text{op} \in \{\text{pub}, \text{priv}, \text{H}\} \\
 \\
 \text{R2 } \frac{}{((M_1 \cdot M_2), (N_1 \cdot N_2)) \notin h} \qquad \text{R3 } \frac{(\text{Enc}_{M_2}^s M_1, \text{Enc}_{N_2}^s N_1) \in h}{(M_2, N_2) \notin \mathcal{S}(h)} \\
 \\
 \text{R4 } \frac{(\text{Enc}_{M_2}^a M_1, \text{Enc}_{N_2}^a N_1) \in h}{(M_1, N_1) \notin \mathcal{S}(h) \vee (M_2, N_2) \notin \mathcal{S}(h)} \\
 \\
 \text{R5 } \frac{(\text{Enc}_{M_2}^a M_1, \text{Enc}_{N_2}^a N_1) \in h \quad (M'_2, N'_2) \in \mathcal{S}(h) \quad M'_2 = \text{inv}(M_2) \quad N'_2 = \text{inv}(N_2)}{(M_1, N_1) \in \mathcal{S}(h)}
 \end{array}$$

Reducing a reduced hedge has no effect (hence the name).

Lemma 51:

Let $h \in \mathbf{H}$. If h is reduced then $h = \text{reduce}(h)$.

PROOF

We have that $\text{reduce}(h) \subset h$. We then show by a simple case analysis that if $(M, N) \in h$, then $(M, N) \in \text{reduce}(h)$. ■

The weak analysis of a reduced hedge is less powerful than the hedge itself.

Lemma 52:

Let $h \in \mathbf{H}$. If h is reduced then $\text{analz}(h) <_H h$.

PROOF

We show that if $(M, N) \in \text{analz}(h)$ then $(M, N) \in \mathcal{S}(h)$ by rule induction on $(M, N) \in \text{analz}(h)$. Then, by Lemma 29, we get $\text{analz}(h) <_H h$. ■

The analysis of a reduced hedge is as powerful as the hedge itself.

Lemma 53:

Let $h \in \mathbf{H}$. If h is reduced then $\mathcal{A}(h) \geq_H h$.

PROOF

We have $h <_H \mathcal{A}(h)$ by Lemma 33.

We show by induction on $i \in \mathbb{N}$ that for all i , $\text{analz}^i(h) <_H h$.

For $i = 0$, it is trivial.

For the inductive case, assume that $\text{analz}^i(h) <_H h$.

By Lemma 32, we have $\text{analz}^{i+1}(h) <_H \text{analz}^i(h)$.

By Lemma 52, we have $\text{analz}(h) <_H h$.

So $\text{analz}^{i+1}(h) <_H h$.

So by Lemma 35, we get $\mathcal{A}(h) <_H h$. ■

A reduced hedge is irreducible.

Lemma 54:

Let $h \in H$. If h is reduced then h is irreducible.

PROOF

Since h is reduced, we have $h \geq_H \mathcal{A}(h)$ by Lemma 53.

So by Lemma 45, we have $h \geq_H \mathcal{I}(h)$.

So by Lemma 47, we get $\mathcal{I}(h) \subseteq h$ since $\mathcal{I}(h)$ is irreducible.

By Lemma 51, we have $h = \text{reduce}(h)$.

Since $\text{reduce}(h) = h \geq_H \mathcal{I}(h)$, we have by Lemma 43 that $h \subseteq \mathcal{I}(h)$.

So $\mathcal{I}(h) = h$. ■

A hedge is irreducible if and only if it is reduced.

Theorem 8:

Let $h \in H$. Then h is reduced if and only if h is irreducible.

PROOF

Lemma 54 gives one direction. The other direction is trivial. ■

4.4 Consistent hedges

4.4.1 Inversing hedges

The *inverse* of a hedge is obtained by swapping the two sides of a hedge.

Definition 64 (h^{-1}).

Let $h \in H$. The *inverse* of h is $h^{-1} := \{(N, M) \mid (M, N) \in h\}$.

The following properties can be easily proven.

Lemma 55:

Let $g, h \in H$. We have

- $(N, M) \in \mathcal{S}(h^{-1}) \iff (M, N) \in \mathcal{S}(h)$
- $\text{analz}(h^{-1}) = \text{analz}(h)^{-1}$
- $\mathcal{A}(h^{-1}) = \mathcal{A}(h)^{-1}$
- $\text{reduce}(h^{-1}) = \text{reduce}(h)^{-1}$
- $\mathcal{I}(h^{-1}) = \mathcal{I}(h)^{-1}$
- $g <_H h \implies g^{-1} <_H h^{-1}$
- $(h^{-1})^{-1} = h$ □

4.4.2 Consistency

Hedges are used to relate indistinguishable messages. However, it can happen that the attacker finds a contradiction in its knowledge. For example, if both (M, N_1) and (M, N_2) are in h and $N_1 \neq N_2$. The notion of *consistency* guarantees the absence of such contradictions and ensures also irreducibility.

Definition 65 (consistency).

A hedge h is *left consistent* if it satisfies the following rules:

$$\text{LC1 } \frac{(M, N) \in h \quad M \in N}{N \in N}$$

$$\text{LC2 } \frac{(M, N) \in h \quad (M', N') \in h \quad M' = M}{N' = N}$$

$$\text{LC3 } \frac{(M, N) \in h \quad (M', N') \in h \quad M' = \text{inv}(M)}{N' = \text{inv}(N)}$$

$$\text{LC4 } \frac{(\text{op}(M'), N) \in h}{(M', N') \notin \mathcal{S}(h)} \quad \text{op} \in \{\text{pub}, \text{priv}, \text{H}\} \quad \text{LC5 } \frac{}{((M_1 \cdot N_1), N) \notin h}$$

$$\text{LC6 } \frac{(\text{Enc}_{M_2}^s M_1, N) \in h}{(M_2, N_2) \notin \mathcal{S}(h)} \quad \text{LC7 } \frac{(\text{Enc}_{M_2}^a M_1, N) \in h}{(M_1, N_1) \notin \mathcal{S}(h) \vee (M_2, N_2) \notin \mathcal{S}(h)}$$

$$\text{LC8 } \frac{(\text{Enc}_{M_2}^a M_1, N) \in h \quad (M'_2, N'_2) \in \mathcal{S}(h) \quad M'_2 = \text{inv}(M_2)}{N'_2 = \text{inv}(N_2) \wedge N = \text{Enc}_{N_2}^a N_1 \wedge (M_1, N_1) \in \mathcal{S}(h)}$$

A hedge h is *consistent* if both h and h^{-1} are left consistent.

The above definition basically ensures that:

- names are identified with names (LC1).
- a message M is not identified with two different messages N and N' (LC2).
- two inverse keys M and M' are identified with inverse keys N and N' (LC3).
- if a message M is identified with N and M can be constructed from the knowledge in an other manner, then N can also be constructed in this same other manner (LC4, LC5, LC6, LC7; an irreducibility condition is also encoded in these rules).
- the decryption power coincides on both sides (LC6, LC8; an irreducibility condition is also encoded in these rules).

Example 16

1. $h_1 := \{(a, \text{Enc}_k^s b)\}$ is not left consistent (by LC1).
2. $h_2 := \{(a, b), (a, c)\}$ is not left consistent (by LC2).
3. $h_3 := \{(\text{pub}(a), b), (\text{priv}(a), c)\}$ is not left consistent (by LC3).
4. $h_4 := \{(a, b), (H(a), c)\}$ is not left consistent (by LC4).
5. $h_5 := \{((a \cdot b), c)\}$ is not left consistent (by LC5).
6. $h_6 := \{(\text{Enc}_k^s a, b), (k, k)\}$ is not left consistent (by LC6).
7. $h_7 := \{(\text{Enc}_k^a a, b), (a, a), (k, k)\}$ is not left consistent (by LC7).
8. $h_8 := \{(\text{Enc}_{\text{priv}(k)}^a a, b), (\text{pub}(k), \text{pub}(l))\}$ is not left consistent (by LC8).
9. $h_9 := \{(a, b), (\text{pub}(k), \text{priv}(l)), (\text{Enc}_{\text{priv}(k)}^a a, \text{Enc}_{\text{pub}(l)}^a b)\}$ is consistent.
10. $h_{10} := \{(\text{priv}(k), \text{pub}(l)), (\text{pub}(k), \text{priv}(l)), (a, b)\}$ is consistent. *

As we have previously suggested, the definition of consistency contains irreducibility conditions. Indeed, a consistent hedge is reduced.

Lemma 56:

Let $h \in H$.

1. If h is left consistent then h is reduced.

2. If h is consistent then h is reduced. □

Hence, a consistent hedge is irreducible.

Theorem 9:

If h is consistent then h is irreducible.

PROOF

By Theorem 8 and Lemma 56. ■

So two consistent hedges that have the same power are equal.

Corollary 6:

Let $g, h \in \mathbf{H}$. If $g \succeq_{\mathbf{H}} h$ and both g and h are consistent then $g = h$.

PROOF

By Theorem 9 and Corollary 5. ■

The following lemma gives a method for showing that an irreducible hedge is left consistent.

Lemma 57:

Let $g, h \in \mathbf{H}$. If g is irreducible, h is left consistent and $g <_{\mathbf{H}} h$ then g is left consistent. □

Hence the following method for showing that an irreducible hedge is consistent.

Theorem 10:

Let $g, h \in \mathbf{H}$. If g is irreducible, h is consistent and $g <_{\mathbf{H}} h$ then g is consistent.

PROOF

By Lemma 57 and Lemma 55. ■

Finally, an interesting subclass of hedges is those hedges h such that $h \subseteq N \times N$. The consistency of such hedges can be simplified to

Lemma 58:

Let $h \subseteq N \times N$ be a hedge. Then h is left consistent if and only if it satisfies LC2.

Hence h is consistent if and only if

$$\forall (M, N), (M', N') \in h : M = M' \iff N = N'$$

4.4.3 Properties

We state some properties of consistent hedges. To have a symmetric treatment of shared-key cryptography and public key cryptography, we split the rule LC6 in two rules LC6A and LC6B built in the spirit of LC7 and LC8.

Definition 66.

Let $h \in \mathbf{H}$. We define

$$\text{LC6A} \frac{(\text{Enc}_{M_2}^s M_1, N) \in h}{(M_1, N_1) \notin \mathcal{S}(h) \vee (M_2, N_2) \notin \mathcal{S}(h)}$$

$$\text{LC6B} \frac{(\text{Enc}_{M_2}^s M_1, N) \in h \quad (M_2, N_2) \in \mathcal{S}(h)}{N = \text{Enc}_{N_2}^s N_1 \wedge (M_1, N_1) \in \mathcal{S}(h)}$$

Lemma 59:

Let $h \in \mathbf{H}$. Then h satisfies LC6 if and only if h satisfies LC6A and LC6B. \square

Hence, we can replace in Definition 65 the rule LC6 by the rules LC6A and LC6B.

If a hedge is left consistent, then $\mathcal{S}(h)$ satisfies LC2 and LC3. More precisely

Lemma 60:

Let $h \in \mathbf{H}$.

1. If h satisfies LC2, LC4, LC5, LC6A, LC7 then $\mathcal{S}(h)$ satisfies LC2, i.e.

$$\forall (M, N), (M', N') \in \mathcal{S}(h) : M' = M \implies N' = N$$

2. If h satisfies LC2, LC3, LC4, LC5, LC6A, LC7 then $\mathcal{S}(h)$ satisfies LC3, i.e.

$$\forall (M, N), (M', N') \in \mathcal{S}(h) : M' = \text{inv}(M) \implies N' = \text{inv}(N)$$

If a hedge is left consistent, then the following properties hold for its synthesis:

Lemma 61:

If $h \in \mathbf{H}$ is left consistent then for all $(M, N) \in \mathcal{S}(h)$,

1. if M is a pair $(M_1 . M_2)$ then N is a pair $(N_1 . N_2)$ such that $(M_1, N_1) \in \mathcal{S}(h)$ and $(M_2, N_2) \in \mathcal{S}(h)$.

2. if M is a shared-key encrypted message $\text{Enc}_{M_2}^s M_1$ and if there exists N_2 such that $(M_2, N_2) \in \mathcal{S}(h)$ (i.e. the shared-key M_2 is known) then N is a shared-key encrypted message $\text{Enc}_{N_2}^s N_1$ such that $(M_1, N_1) \in \mathcal{S}(h)$.
3. if M is a public-key encrypted message $\text{Enc}_{M_2}^a M_1$ with $\text{inv}(M_2) = M'_2 \in \mathcal{M}$ and if there exists N'_2 such that $(M'_2, N'_2) \in \mathcal{S}(h)$ then N is a public-key encrypted message $\text{Enc}_{N_2}^a N_1$ with $(M_1, N_1) \in \mathcal{S}(h)$ and $\text{inv}(N_2) = N'_2$. \square

Conclusion

We have presented the notion of hedges which is used in hedged bisimulations to represent the environment knowledge; it simply consists of a finite set of pairs of indistinguishable messages. We have defined two basic operations on hedges: synthesis is the (infinite) set of indistinguishable messages that can be constructed from a hedge and analysis is the finite set of indistinguishable messages that can be deconstructed from a hedge. For this last operation, we have described a constructive proof of existence. We then have defined the concept of irreducible hedges and shown several properties about them, one of which is that the irreducible part of a hedge h is the minimal seed for generating the synthesis of $\mathcal{A}(h)$. Finally, we have defined the notion of consistent hedges which are the hedges whose knowledge is not considered contradictory.

Chapter 5

Open Bisimulation for the Spi Calculus

In the previous chapters, we have presented the spi calculus and explained why standard bisimulation definitions (i.e. inherited from the pi calculus) are not well-suited when reasoning about security protocols. To be able to apply the bisimulation proof technique to security protocols, we have explained that bisimulation should be enriched with a data structure that explicitly keeps track of indistinguishable messages. In this chapter, we first present quickly the notion of late hedged bisimulation, which is one environment-sensitive notion of bisimulation for the spi calculus that uses the data structure of hedges (see Chapter 4). Secondly, we define a new notion of bisimulation for the spi calculus called *open hedged bisimulation*; the idea being to generalise Sangiorgi's open bisimulation to the spi calculus framework. We show that this definition is indeed an extension of K-open bisimulation as defined in Chapter 2. We finally give a symbolic characterisation of open hedged bisimulation and explain why it is an important step towards mechanisation of this new notion of bisimulation.

5.1 Late hedged bisimulation

A hedged bisimulation [41] is a symmetric consistent hedged relation that satisfies the bisimulation game.

Definition 67 (hedged relation).

A *hedged relation* \mathcal{R} is a subset of $\mathbf{H} \times \mathbf{P} \times \mathbf{P}$ such that whenever $(h, P, Q) \in \mathcal{R}$, we have $\text{fn}(P) \subseteq \text{n}(\pi_1(h))$ and $\text{fn}(Q) \subseteq \text{n}(\pi_2(h))$.

A hedged relation \mathcal{R} is *symmetric* if whenever $(h, P, Q) \in \mathcal{R}$ we have $(h^{-1}, Q, P) \in \mathcal{R}$.

A hedged relation \mathcal{R} is *consistent* if whenever $(h, P, Q) \in \mathcal{R}$, we have that h is a consistent hedge.

Before giving the definition of hedged bisimulation, we define the sets of input messages that an intruder is able to construct from a hedge h . Basically, these messages come from the synthesis of h but the observer is also allowed to use some fresh names.

Definition 68.

Let $h \in \mathbf{H}$, $(M, N) \in \mathbf{M} \times \mathbf{M}$

Let $B \subseteq N \times N$ a consistent hedge such that

1. $\pi_1(B) \cap \mathfrak{n}(\pi_1(h)) = \emptyset$
2. $\pi_2(B) \cap \mathfrak{n}(\pi_2(h)) = \emptyset$

i.e. the names of B are fresh component-wise w.r.t. to those of h .

We write $h \vdash_B (M, N)$ if

1. $\forall (b_1, b_2) \in B : b_1 \in \mathfrak{n}(M) \vee b_2 \in \mathfrak{n}(N)$
2. $(M, N) \in \mathcal{S}(h \cup B)$

The first condition requires that the names of B are needed for constructing the pair (M, N) and the second condition requires that M and N are two indistinguishable messages that can be constructed from $h \cup B$.

Note that in the first condition, it is equivalent to require that $b_1 \in \mathfrak{n}(M) \wedge b_2 \in \mathfrak{n}(N)$ or more simply $\pi_1(B) \subseteq \mathfrak{n}(M)$.

Definition 69 (late hedged bisimulation).

A symmetric and consistent hedged relation \mathcal{R} is a (*strong*) *late hedged bisimulation* if whenever $(h, P, Q) \in \mathcal{R}$, we have that

1. if $P \xrightarrow{\tau} P'$ then
there exists Q' such that $Q \xrightarrow{\tau} Q'$ and $(h, P', Q') \in \mathcal{R}$
2. if $P \xrightarrow{a} (x)P'$ (with $x \notin \mathfrak{n}(\pi_1(h))$)
and $(a, b) \in \mathcal{S}(h)$ then
there exist y and Q' such that $Q \xrightarrow{b} (y)Q'$
(with $y \notin \mathfrak{n}(\pi_2(h))$)
and for all B and (M, N) such that $h \vdash_B (M, N)$
we have $(h \cup B, P' \{^M/x\}, Q' \{^N/y\}) \in \mathcal{R}$.

3. if $P \xrightarrow{\bar{a}} (v\bar{c}) \langle M \rangle P'$ (with $\{\bar{c}\} \cap \mathbf{n}(\pi_1(h)) = \emptyset$)
and $(a, b) \in \mathcal{S}(h)$ then
there exist $\{\bar{d}\}, Q'$ and N such that $Q \xrightarrow{\bar{b}} (v\bar{d}) \langle N \rangle Q'$
(with $\{\bar{d}\} \cap \mathbf{n}(\pi_2(h)) = \emptyset$)
and $(\mathcal{I}(h \cup \{(M, N)\}), P', Q') \in \mathcal{R}$.

In the above definition, emitted messages are added to the indistinguishability relation encoded as a hedge yielding $h \cup \{(M, N)\}$ (clause 3). The addition of this new pair to the indistinguishability relation may allow to deconstruct some more pairs of the environment knowledge; this is done by computing the irreducible elements of $h \cup \{(M, N)\}$. The consistency condition ensures that the resulting hedge is not contradictory. Finally, the condition $(a, b) \in \mathcal{S}(h)$ of clauses 2 and 3 may be interpreted as "the environment is able to detect an action on channel a ". This can be equivalently written $(a, b) \in h$ thanks to Lemma 26.

Note also that since it is required that the free names of P and Q are mentioned in h , this imposes that the names of B are distinct from every free names of P and Q .

Given a hedge h and two processes P and Q , we say that P and Q are *late hedged bisimilar* under h if there exists a late hedged bisimulation \mathcal{R} such that $(h, P, Q) \in \mathcal{R}$ and we write $P \sim_{\text{LH}}^h Q$.

As previously mentioned, late hedged bisimilarity is a sound proof technique for showing barbed equivalence.

5.2 Open hedged bisimulation

Open bisimulation is an attractive notion of bisimulation for the pi calculus for a number of different reasons explained in Chapter 2. Based on this observation, we have generalised this notion of bisimulation to the spi calculus. Our proposal is presented afterwards, following [48, 50, 46]. Firstly, we present the notion of S-environment that is used to represent the environment-knowledge and to characterise the set of *respectful substitutions*. Secondly, we give the definition of open hedged bisimulation and the corresponding notion of bisimilarity. Thirdly, we show that open hedged bisimilarity is sound w.r.t. late hedged bisimilarity.

5.2.1 S-environments

For defining open bisimulation in the spi calculus, we have to record on each input, during the bisimulation game, every message the attacker can

substitute to the input variable given its current knowledge. This information is represented by S-environments which consist of a hedge h representing the attacker's current knowledge, v which are names used as input names so far and \prec which allows to recover the hedge the attacker had when a given input name was input. Moreover, since we require that communications can only occur on channel names, S-environments also need to remember which input names can be substituted by names only, this is stored in γ_l, γ_r .

The form of S-environments follows the form of K-environments we have introduced in Chapter 2. Note however that it is not sufficient to consider as S-environment a simple extension of K-environment by saying that a S-environment is a triple (O, V, \prec) where O would be a set of messages, V a (finite) set of names, and \prec a subset of $O \times V$. One reason is that it would not be possible to build up an indistinguishability relation on top of this data. Thus, as with hedges, we split the sets O and V and obtain respectively a set of message pairs h (i.e. a hedge) and a set of name pairs v (i.e. a hedge containing only names). Another reason is that in the spi calculus, unlike the pi calculus, we need to record that some names that were at some moment considered as channels must not later on be replaced by complex messages (see Lemma 24).

Definition 70 (S-environment).

A S-environment is a quadruple $\mathbf{se} = (h, v, \prec, (\gamma_l, \gamma_r))$ where $h \in \mathbf{H}$, $v \subseteq N \times N$ is a consistent hedge, $\prec \subseteq h \times v$, $\gamma_l \subseteq \pi_1(v)$ and $\gamma_r \subseteq \pi_2(v)$. The set of all S-environments is written \mathcal{S}_H .

The *hedge available* to $(x, y) \in v$ according to \prec is defined by $\mathbf{se}|_{(x,y)} := \{(M, N) \in h \mid (M, N) \prec (x, y)\}$.

The *concrete hedge* of \mathbf{se} is $\mathfrak{H}(\mathbf{se}) := h \cup v$.

The inverse of \mathbf{se} is $\mathbf{se}^{-1} := (h^{-1}, v^{-1}, \prec^{-1}, (\gamma_r, \gamma_l))$ where $(N, M) \prec^{-1} (y, x)$ iff $(M, N) \prec (x, y)$.

The intuition behind \prec is that if $(M, N) \prec (x, y)$, the attacker knew about (M, N) whenever (x, y) was used for input in the bisimulation game. In that case, we need to require that $x \notin n(M)$ and $y \notin n(N)$ to avoid circularities, which is included in the following definition.

Definition 71 (well-formed S-environment).

A S-environment $(h, v, \prec, (\gamma_l, \gamma_r))$ is *well-formed* if

1. $\pi_1(h) \cap \pi_1(v) = \emptyset$
2. $\pi_2(h) \cap \pi_2(v) = \emptyset$

$$3. \forall (M, N) \in h : \forall (x, y) \in v : (M, N) \prec (x, y) \implies \begin{cases} x \notin n(M) \\ y \notin n(N) \end{cases}$$

There are three relevant ways to add information to a S-environment \mathfrak{se} . We can add a pair of indistinguishable messages (M, N) to the hedge h (on process outputs) —note that whenever (M, N) was produced by the attacker, we don't put it in h since it adds no information to the attacker's knowledge. We can add a fresh pair (x, y) of input variables to v and update \prec so that the hedge $\mathfrak{se}|_{(x,y)}$ corresponds to the current hedge h (on process inputs). And finally, we can add new constraints in γ_l and γ_r to reflect that some input names were used as channels (on process transitions).

Definition 72.

Let $\mathfrak{se} = (h, v, \prec, (\gamma_l, \gamma_r))$ be a S-environment.

If $(M, N) \in \mathbf{M} \times \mathbf{M}$, we define $\mathfrak{se} +_o(M, N) := (h', v, \prec, (\gamma_l, \gamma_r))$ where $h' = h$ if $(M, N) \in v$ and $h' = h \cup \{(M, N)\}$ otherwise.

If $(x, y) \in \mathbf{N} \times \mathbf{N}$, we define $\mathfrak{se} +_i(x, y) := (h, v \cup \{(x, y)\}, \prec', (\gamma_l, \gamma_r))$ where $\prec' := \prec \cup (h \times \{(x, y)\})$.

If $S_1, S_2 \subseteq \mathbf{N}$, we define $\mathfrak{se} +_c(S_1, S_2) := (h, v, \prec, (\gamma'_l, \gamma'_r))$ where $\gamma'_l := \gamma_l \cup (S_1 \cap \pi_1(v))$ and $\gamma'_r := \gamma_r \cup (S_2 \cap \pi_2(v))$.

By adding information to particular S-environments as shown above, hedges available to variables in v can be ordered in an increasing sequence of hedges. This property is captured by the following definition.

Definition 73 (growing S-environment).

A S-environment $\mathfrak{se} = (h, v, \prec, (\gamma_l, \gamma_r))$ is *growing* if there exists an injective mapping $z : \llbracket 1, n \rrbracket \rightarrow v$ (where $n = \text{card}(v)$) such that for all $1 \leq i < n$, we have $h_i \subseteq h_{i+1}$ where $h_i := \mathfrak{se}|_{z(i)}$.

The part (h, v, \prec) of a growing S-environment can be seen as a sequence of hedges $h_1 \cdot h_2 \cdot \dots \cdot h_n$ and a sequence of pairs of input names $(x_1, y_1) \cdot (x_2, y_2) \cdot \dots \cdot (x_{n-1}, y_{n-1})$ with $h_i \subseteq h_{i+1}$ for $1 \leq i < n$, $h = h_n$, $v = \{(x_1, y_1), \dots, (x_{n-1}, y_{n-1})\}$ and $(M, N) \prec (x_i, y_i)$ iff $(M, N) \in h_i$ for $1 \leq i < n$.

Conceptually, a S-environment \mathfrak{se} is a concise representation of every pair of substitutions resulting from plays performed by the attacker in the bisimulation game. These pairs are said to *respect* \mathfrak{se} and are given by the following definition.

Definition 74 (respectful substitutions).

Let (σ, ρ) be a pair of substitutions, $B \subseteq \mathbf{N} \times \mathbf{N}$ a consistent hedge and $\mathfrak{se} = (h, v, \prec, (\gamma_l, \gamma_r))$ a S-environment. We say that (σ, ρ) *respects* \mathfrak{se} with B — written $(\sigma, \rho) \triangleright_B \mathfrak{se}$ — if

1. $\text{supp}(\sigma) \subseteq \pi_1(v)$
2. $\text{supp}(\rho) \subseteq \pi_2(v)$
3. $\forall (b_1, b_2) \in B : b_1 \in \mathbf{n}(\sigma(\pi_1(v))) \vee b_2 \in \mathbf{n}(\rho(\pi_2(v)))$
4. $\pi_1(B) \cap (\mathbf{n}(\pi_1(h)) \setminus \pi_1(v)) = \emptyset$
5. $\pi_2(B) \cap (\mathbf{n}(\pi_2(h)) \setminus \pi_2(v)) = \emptyset$
6. $\forall (x, y) \in v : (x\sigma, y\rho) \in \mathcal{S}(\mathcal{I}(\mathbf{se}|_{(x,y)}(\sigma, \rho) \cup B))$
7. $\forall x \in \gamma_l : x\sigma \in N$
8. $\forall y \in \gamma_r : y\rho \in N$

In this definition, substitutions affect only names in v (input names). Given $(x, y) \in v$, these names can be replaced by any pair of messages the attacker could have synthesised from $\mathbf{se}|_{(x,y)}$ possibly adding fresh names (B) and taking into account previous choices made by the attacker for other input names. Moreover, input names used as communication channels (mentioned in γ_l or γ_r) are prevented from being substituted by something else than a name.

In a given S-environment \mathbf{se} , choices made by the attacker during the bisimulation game correspond to pairs (σ, ρ) of respectful substitutions. These choices lead to an updated S-environment $\mathbf{se}_B^{(\sigma, \rho)}$.

Definition 75 (S-environment updating).

Let (σ, ρ) be a pair of substitutions, $B \subseteq N \times N$ a consistent hedge and $\mathbf{se} = (h, v, \prec, (\gamma_l, \gamma_r))$ a S-environment such that $(\sigma, \rho) \triangleright_B \mathbf{se}$. The *update* $\mathbf{se}_B^{(\sigma, \rho)} = (h', v', \prec', (\gamma'_l, \gamma'_r))$ of \mathbf{se} by (σ, ρ) is defined as follows:

- $h' = h(\sigma, \rho)$
- $v' = B$
- \prec' is defined by

$$(M\sigma, N\rho) \prec' (x', y') \iff \bigwedge_{\substack{(x, y) \in v \\ x' \in \mathbf{n}(x\sigma) \vee y' \in \mathbf{n}(y\rho)}} (M, N) \prec (x, y)$$

for $(M, N) \in h$ and $(x', y') \in v'$.

- $\gamma'_l = \sigma(\gamma_l) \cap \pi_1(v')$

- $\gamma'_r = \rho(\gamma_r) \cap \pi_2(v')$

Well-formedness of S-environments is preserved by updates.

Lemma 62 (well-formedness preservation):

Let (σ, ρ) be a pair of substitutions, $B \subseteq N \times N$ a consistent hedge and $\mathbf{se} = (h, v, \prec, (\gamma_l, \gamma_r))$ a S-environment such that $(\sigma, \rho) \triangleright_B \mathbf{se}$. If \mathbf{se} is well-formed then $\mathbf{se}_B^{(\sigma, \rho)}$ is well-formed.

PROOF

We note $\mathbf{se}_B^{(\sigma, \rho)} = (h', v', \prec', (\gamma'_l, \gamma'_r))$.

1. By contradiction, assume that $x \in \pi_1(h') \cap \pi_1(v')$.

By definition of h' , there is $M \in \pi_1(h)$ such that $x = M\sigma$. Since $x \in N$, necessarily $M = a \in N$.

Since $\pi_1(h) \cap \pi_1(v) = \emptyset$ and $\text{supp}(\sigma) \subseteq \pi_1(v)$, we have $x = M\sigma = a\sigma = a$ because $a \notin \text{supp}(\sigma)$.

We have thus $a \in n(\pi_1(h))$, $a \notin \pi_1(v)$ and $a \in \pi_1(B)$. This is a contradiction. So $\pi_1(h') \cap \pi_1(v') = \emptyset$.

2. Similarly $\pi_2(h') \cap \pi_2(v') = \emptyset$.

3. Assume that $(M\sigma, N\rho) \prec' (x', y')$ with $(M, N) \in h$ and $(x', y') \in v'$.

By contradiction, assume that $x' \in n(M\sigma)$. Necessarily, there exists $(x, y) \in v$ such that $x \in n(M)$ and $x' \in n(x\sigma)$. So we have $(M, N) \prec (x, y)$. This is a contradiction with $x \notin n(M)$. So $x' \notin n(M\sigma)$.

Similarly, $y' \notin n(N\rho)$. ■

Growth of S-environments is preserved by updates.

Lemma 63 (growth preservation):

Let (σ, ρ) be a pair of substitutions, $B \subseteq N \times N$ a consistent hedge and $\mathbf{se} = (h, v, \prec, (\gamma_l, \gamma_r))$ a S-environment such that $(\sigma, \rho) \triangleright_B \mathbf{se}$. If \mathbf{se} is growing then $\mathbf{se}_B^{(\sigma, \rho)}$ is growing.

PROOF

We write $v = \{(x_1, y_1), \dots, (x_n, y_n)\}$ and $h_i = \mathbf{se}|_{(x_i, y_i)}$ and assume that for all $1 \leq i < n$, we have $h_i \subseteq h_{i+1}$.

We note $\mathbf{se}_B^{(\sigma, \rho)} = (h', v', \prec', (\gamma'_l, \gamma'_r))$.

By definition, we have $(M, N) \prec (x_i, y_i) \iff (M, N) \in h_i$ for $(M, N) \in h$ and $1 \leq i \leq n$.

Let $(x', y') \in v'$ and $(M', N') \in h'$. We have $M' = M\sigma$ and $N' = N\rho$ for $(M, N) \in h$.

By definition, we have

$$\begin{aligned}
(M', N') \prec' (x', y') &\iff \bigwedge_{\substack{(x, y) \in v \\ x' \in \mathbf{n}(x\sigma) \vee y' \in \mathbf{n}(y\rho)}} (M, N) \prec (x, y) \\
&\iff \bigwedge_{1 \leq i \leq n} (M, N) \prec (x_i, y_i) \\
&\iff \bigwedge_{1 \leq i \leq n} (M, N) \in h_i \\
&\iff x' \in \mathbf{n}(x_i\sigma) \vee y' \in \mathbf{n}(y_i\rho)
\end{aligned}$$

Moreover, we know that if $(x', y') \in v' = B$, we have $x' \in \mathbf{n}(\sigma(\pi_1(v)))$ or $y' \in \mathbf{n}(\sigma(\pi_2(v)))$, so

$$A_{(x', y')} := \{1 \leq i \leq n \mid x' \in \mathbf{n}(x_i\sigma) \vee y' \in \mathbf{n}(y_i\rho)\} \neq \emptyset$$

Since $A_{(x', y')}$ is a non empty subset of \mathbb{N} , its minimum element exists. We thus define $idx((x', y')) := \min A_{(x', y')}$.

Since we have $h_1 \subseteq h_2 \subseteq \dots \subseteq h_n$, we have

$$(M', N') \prec' (x', y') \iff (M, N) \in h_{idx((x', y'))}$$

for every $(M, N) \in h$, $(x', y') \in v'$, $M' = M\sigma$ and $N' = N\rho$.

We sort the elements $(x', y') \in v'$ according to the value of $idx((x', y'))$, i.e. let $z : \llbracket 1, k \rrbracket \rightarrow v'$ injective where $k := \text{card}(v')$ such that if $i \leq j$ then $idx(z(i)) \leq idx(z(j))$.

For $1 \leq i \leq k$, we define

$$\begin{aligned}
h'_i &:= \mathbf{se}_B^{(\sigma, \rho)}|_{z(i)} = \{(M', N') \in h' \mid (M', N') \prec' z(i)\} \\
&= \{(M\sigma, N\rho) \mid (M, N) \in h \wedge (M, N) \in h_{idx(z(i))}\} \\
&= h_{idx(z(i))}(\sigma, \rho)
\end{aligned}$$

Thus since $h_1 \subseteq h_2 \subseteq \dots \subseteq h_n$, we have for $1 \leq i < k$ that $h'_i \subseteq h'_{i+1}$.

Hence $\mathbf{se}_B^{(\sigma, \rho)}$ is growing. \blacksquare

We are now going to show that respectful substitutions compose, as stated by Theorem 11 (page 130). We show some auxiliary results before proving this theorem.

In case of well-formed and growing S-environments, the third condition of Definition 74 can be strengthened according to the following lemma.

Lemma 64:

Let (σ, ρ) be a pair of substitutions, $B \subseteq N \times N$ a consistent hedge and $\mathbf{se} = (h, v, \prec, (\gamma_l, \gamma_r))$ a S-environment such that $(\sigma, \rho) \triangleright_B \mathbf{se}$. Then if \mathbf{se} is well-formed and growing we have

$$\forall (b_1, b_2) \in B : b_1 \in \mathfrak{n}(\sigma(\pi_1(v))) \wedge b_2 \in \mathfrak{n}(\rho(\pi_2(v)))$$

PROOF

We write $v = \{(x_1, y_1), \dots, (x_n, y_n)\}$ such that if $h_i := \mathbf{se}|_{(x_i, y_i)}$ then $h_i \subseteq h_{i+1}$ for $1 \leq i < n$.

By contradiction, assume that there is $(b_1, b_2) \in B$ such that $b_1 \notin \mathfrak{n}(\sigma(\pi_1(v)))$ or $b_2 \notin \mathfrak{n}(\rho(\pi_2(v)))$.

By symmetry, assume for example that $b_2 \notin \mathfrak{n}(\rho(\pi_2(v)))$. By hypothesis, we have then that $b_1 \in \mathfrak{n}(\sigma(\pi_1(v)))$.

Let i_0 minimal such that $b_1 \in \mathfrak{n}(x_{i_0}\sigma)$. We have $b_2 \notin \mathfrak{n}(y_{i_0}\rho)$.

By hypothesis, we have $(x_{i_0}\sigma, y_{i_0}\rho) \in \mathcal{S}(\mathcal{I}(h_{i_0}(\sigma, \rho) \cup B))$.

Since $b_2 \notin \mathfrak{n}(y_{i_0}\rho)$, SYN-INC have not been applied with (b_1, b_2) as premise. So necessarily, there is $(M, N) \in h_{i_0}$ such that $b_1 \in \mathfrak{n}(M\sigma)$. Since $\pi_1(B) \cap (\mathfrak{n}(\pi_1(h)) \setminus \pi_1(v)) = \emptyset$, there exists j such that $x_j \in \mathfrak{n}(M)$ and $b_1 \in \mathfrak{n}(x_j\sigma)$. By choice of i_0 , we have $i_0 \leq j$ so $h_{i_0} \subseteq h_j$.

Since $(M, N) \in h_{i_0} \subseteq h_j$, we have $(M, N) \prec (x_j, y_j)$.

Since \mathbf{se} is well-formed, we have $x_j \notin \mathfrak{n}(M)$. This is a contradiction.

This even proves that if $(b_1, b_2) \in B$ and i is minimal such that $b_1 \in \mathfrak{n}(x_i\sigma)$ then necessarily $b_2 \in \mathfrak{n}(y_i\rho)$ (this result will be used afterwards). ■

Lemma 65:

Let $h \in \mathbf{H}$ and $\{(x_1, y_1), \dots, (x_n, y_n)\} \subseteq N \times N$.

Let also $(M_1, N_1), \dots, (M_n, N_n) \in M \times M$ and $B \subseteq N \times N$ such that

$$\forall 1 \leq i \leq n : (M_i, N_i) \in \mathcal{S}(\mathcal{A}(h(\sigma, \rho) \cup B))$$

where σ and ρ are defined such that

$$x\sigma := \begin{cases} M_i & \text{if } x = x_i \\ x & \text{otherwise} \end{cases} \quad y\rho := \begin{cases} N_i & \text{if } y = y_i \\ y & \text{otherwise} \end{cases}$$

Then

$$\forall (M, N) \in \mathcal{S}(\mathcal{A}(h \cup \{(x_1, y_1), \dots, (x_n, y_n)\})) : \\ (M\sigma, N\rho) \in \mathcal{S}(\mathcal{A}(h(\sigma, \rho) \cup B))$$

PROOF

We use Lemma 35 and actually show that

$$\forall i \in \mathbb{N} : \forall (M, N) \in \mathcal{S}(\text{analz}^i(h \cup \{(x_1, y_1), \dots, (x_n, y_n)\})) : \\ (M\sigma, N\rho) \in \mathcal{S}(\mathcal{A}(h(\sigma, \rho) \cup B))$$

Before showing this result, we show some auxiliary results.

Lemma:

1. Let $h' \in \mathbf{H}$ such that

$$\forall (M, N) \in h' : (M\sigma, N\rho) \in \mathcal{S}(\mathcal{A}(h(\sigma, \rho) \cup B))$$

Then

$$\forall (M, N) \in \mathcal{S}(h') : (M\sigma, N\rho) \in \mathcal{S}(\mathcal{A}(h(\sigma, \rho) \cup B))$$

PROOF

We show this result by rule induction on $(M, N) \in \mathcal{S}(h')$. The hypothesis gives the base case and the inductive cases are then obvious. ■

Lemma:

2. Let $h' \in \mathbf{H}$ such that

$$\forall (M, N) \in h' : (M\sigma, N\rho) \in \mathcal{S}(\mathcal{A}(h(\sigma, \rho) \cup B))$$

then

$$\forall (M, N) \in \text{analz}(h') : (M\sigma, N\rho) \in \mathcal{S}(\mathcal{A}(h(\sigma, \rho) \cup B))$$

PROOF

Again, we show this result by rule induction on $(M, N) \in \text{analz}(h')$.

- If $(M, N) \in \text{analz}(h')$ by ANA-INC. Then $(M, N) \in h'$ and the hypothesis gives the result.

- Assume that $(M, N) \in \text{analz}(h')$ by ANA-DEC-A. That means that $(\text{Enc}_K^a M, \text{Enc}_L^a N) \in \text{analz}(h')$ with $K' = \text{inv}(K) \in \mathbf{M}$, $L' = \text{inv}(L) \in \mathbf{M}$ and $(K', L') \in \mathcal{S}(h')$.

By induction, $(\text{Enc}_{K\sigma}^a M\sigma, \text{Enc}_{L'\rho}^a N\rho) \in \mathcal{S}(\mathcal{A}(h(\sigma, \rho) \cup B))$.

Either it was deduced by SYN-INC or by SYN-ENC-A.

- (a) If it was by SYN-INC:

Then $(\text{Enc}_{K\sigma}^a M\sigma, \text{Enc}_{L'\rho}^a N\rho) \in \mathcal{A}(h(\sigma, \rho) \cup B)$.

Trivially, $\text{inv}(K\sigma) = K'\sigma \in \mathbf{M}$, $\text{inv}(L'\rho) = L'\rho \in \mathbf{M}$.

According to the previous auxiliary result and since h' satisfies the premise, we have $(K'\sigma, L'\rho) \in \mathcal{S}(\mathcal{A}(h(\sigma, \rho) \cup B))$.

By definition of analysis, we have $\text{analz}(\mathcal{A}(h(\sigma, \rho) \cup B)) = \mathcal{A}(h(\sigma, \rho) \cup B)$.

So by ANA-DEC-A $(M\sigma, N\rho) \in \mathcal{A}(h(\sigma, \rho) \cup B)$.

Thus by SYN-INC, we have $(M\sigma, N\rho) \in \mathcal{S}(\mathcal{A}(h(\sigma, \rho) \cup B))$.

- (b) Otherwise, it was by SYN-ENC-A and then immediately, we have $(M\sigma, N\rho) \in \mathcal{S}(\mathcal{A}(h(\sigma, \rho) \cup B))$. ■

We show now that

$$\forall i \in \mathbb{N} : \forall (M, N) \in \text{analz}^i(h \cup \{(x_1, y_1), \dots, (x_n, y_n)\}) : \\ (M\sigma, N\rho) \in \mathcal{S}(\mathcal{A}(h(\sigma, \rho) \cup B))$$

By induction on i .

- $i = 0$

We have by definition

$$\text{analz}^0(h \cup \{(x_1, y_1), \dots, (x_n, y_n)\}) = h \cup \{(x_1, y_1), \dots, (x_n, y_n)\}$$

If $(M, N) \in h$, then by definition, $(M\sigma, N\rho) \in h(\sigma, \rho)$. So by definition of the synthesis, $(M\sigma, N\rho) \in \mathcal{S}(\mathcal{A}(h(\sigma, \rho) \cup B))$.

If $(M, N) = (x_i, y_i)$ for some $1 \leq i \leq n$. Then $(M\sigma, N\rho) = (M_i, N_i) \in \mathcal{S}(\mathcal{A}(h(\sigma, \rho) \cup B))$ by hypothesis.

- Assume the result holds for some $i \in \mathbb{N}$.

Then the second auxiliary lemma gives the result for $i + 1$ because $\text{analz}(\text{analz}^i(h)) = \text{analz}^{i+1}(h)$.

Then by the first auxiliary result, we obtain

$$\forall i \in \mathbb{N} : \forall (M, N) \in \mathcal{S}(\text{analz}^i(h \cup \{(x_1, y_1), \dots, (x_n, y_n)\})) : \\ (M\sigma, N\rho) \in \mathcal{S}(\mathcal{A}(h(\sigma, \rho) \cup B))$$

This completes the proof. \blacksquare

We can now prove that for well-formed and growing S-environments, respectful substitutions compose.

Theorem 11 (respectful substitutions composition):

Let $\mathbf{se} = (h, v, \prec, (\gamma_l, \gamma_r))$ a S-environment. We assume that \mathbf{se} is well-formed and growing.

Let (σ_1, ρ_1) be a pair of substitutions and $B_1 \subseteq N \times N$ a consistent hedge such that $(\sigma_1, \rho_1) \triangleright_{B_1} \mathbf{se}$. We note $\mathbf{se}_1 := \mathbf{se}_{B_1}^{(\sigma_1, \rho_1)}$.

Let (σ_2, ρ_2) be a pair of substitutions and $B_2 \subseteq N \times N$ a consistent hedge such that $(\sigma_2, \rho_2) \triangleright_{B_2} \mathbf{se}_1$. We note $\mathbf{se}_2 := \mathbf{se}_{B_2}^{(\sigma_2, \rho_2)}$.

Then $(\sigma, \rho) \triangleright_{B_2} \mathbf{se}$ and $\mathbf{se}_{B_2}^{(\sigma, \rho)} = \mathbf{se}_2$ where σ and ρ are defined such that

$$x\sigma := \begin{cases} x\sigma_1\sigma_2 & \text{if } x \in \pi_1(v) \\ x & \text{otherwise} \end{cases} \quad y\rho := \begin{cases} y\rho_1\rho_2 & \text{if } y \in \pi_2(v) \\ y & \text{otherwise} \end{cases}$$

PROOF

First, by Lemma 62 and by Lemma 63, we know that both \mathbf{se}_1 and \mathbf{se}_2 are well-formed and growing.

1. By definition, we have $\text{supp}(\sigma) \subseteq \pi_1(v)$.
2. Similarly, $\text{supp}(\rho) \subseteq \pi_2(v)$.
3. Let $(b_1, b_2) \in B_2$. By Lemma 64, we have $b_1 \in \mathfrak{n}(\sigma_2(\pi_1(B_1)))$. So, there exists $(a_1, a_2) \in B_1$ such that $b_1 \in \mathfrak{n}(a_1\sigma_2)$.
By Lemma 64, we have $a_1 \in \mathfrak{n}(\sigma_1(\pi_1(v)))$. So, there exists $(x, y) \in v$ such that $a_1 \in \mathfrak{n}(x\sigma_1)$.
Then $b_1 \in \mathfrak{n}(x\sigma_1\sigma_2) = \mathfrak{n}(x\sigma)$.
4. By contradiction, assume that there exists $b_1 \in \pi_1(B_2) \cap (\mathfrak{n}(\pi_1(h)) \setminus \pi_1(v))$.
By hypothesis, we have $b_1 \notin \mathfrak{n}(\pi_1(h(\sigma_1, \rho_1)))$ or $b_1 \in \pi_1(B_1)$.
If $b_1 \in \pi_1(B_1)$ then $b_1 \notin (\mathfrak{n}(\pi_1(h)) \setminus \pi_1(v))$.

So necessarily, $b_1 \notin n(\pi_1(h(\sigma_1, \rho_1)))$. But since $b_1 \in n(\pi_1(h))$ and $b_1 \notin \pi_1(v)$ and $\text{supp}(\sigma_1) \subseteq \pi_1(v)$, we have $b_1 \in n(\pi_1(h(\sigma_1, \rho_1)))$. This is a contradiction.

So $\pi_1(B_2) \cap (n(\pi_1(h)) \setminus \pi_1(v)) = \emptyset$.

5. Similarly $\pi_2(B_2) \cap (n(\pi_2(h)) \setminus \pi_2(v)) = \emptyset$.

6. We first prove that $h(\sigma_1, \rho_1)(\sigma_2, \rho_2) = h(\sigma, \rho)$, i.e. we show that for every $(M, N) \in h$, $(M\sigma_1\sigma_2, N\rho_1\rho_2) = (M\sigma, N\rho)$.

Let $(M, N) \in h$. We show that $M\sigma_1\sigma_2 = M\sigma$.

Let $x \in n(M)$. We have $x \in \pi_1(h)$. If $x \in \pi_1(v)$, then $x\sigma_1\sigma_2 = x\sigma$. Otherwise, if $x \notin \pi_1(v)$, then we have by hypothesis that $x \notin \pi_1(B_1)$. Moreover, since $\text{supp}(\sigma_1) \subseteq \pi_1(v)$, we have $x\sigma_1 = x$. And since $\text{supp}(\sigma_2) \subseteq \pi_1(B_1)$, we have $x\sigma_2 = x$. Thus $x\sigma_1\sigma_2 = x = x\sigma$. So for every name x of M , we have $x\sigma_1\sigma_2 = x\sigma$. So a simple induction on M shows that $M\sigma_1\sigma_2 = M\sigma$.

Thus $h(\sigma_1, \rho_1)(\sigma_2, \rho_2) = h(\sigma, \rho)$.

We write $v = \{(x_1, y_1), \dots, (x_n, y_n)\}$ such that if $h_i = \text{se}|_{(x_i, y_i)}$, then for $1 \leq i < n$, we have $h_i \subseteq h_{i+1}$.

Let $1 \leq i \leq n$.

We have by hypothesis $(x_i\sigma_1, y_i\rho_1) \in \mathcal{S}(\mathcal{I}(h_i(\sigma_1, \rho_1) \cup B_1))$.

Let $B_1^i := \{(b_1, b_2) \in B_1 \mid \exists j \leq i : b_1 \in n(x_j\sigma_1) \vee b_2 \in n(y_j\rho_1)\}$. We have $B_1 = B_1^1 \cup (B_1 \setminus B_1^1)$.

Let $(b_1, b_2) \in B_1$ such that $b_1 \in n(\pi_1(h_i(\sigma_1, \rho_1)))$. By definition, there exists $(M, N) \in h_i$ such that $b_1 \in n(M\sigma_1)$. This implies that there exists j such that $x_j \in n(M)$ and $b_1 \in n(x_j\sigma_1)$. If $j \geq i$, then since se is growing, we have $(M, N) \prec (x_j, y_j)$. But since se is well-formed, we have $x_j \notin n(M)$. This is a contradiction. Thus $j < i$ and $(b_1, b_2) \in B_1^i$.

Similarly if $(b_1, b_2) \in B_1$ is such that $b_2 \in n(\pi_2(h_i(\sigma_1, \rho_1)))$ then $(b_1, b_2) \in B_1^i$.

This proves that the useful names of B_1 to compute the analysis $\mathcal{A}(h_i(\sigma_1, \rho_1) \cup B_1)$ are included in B_1^i .

In other words, we have just proven that $\mathcal{S}(\mathcal{I}(h_i(\sigma_1, \rho_1) \cup B_1)) = \mathcal{S}(\mathcal{I}(h_i(\sigma_1, \rho_1) \cup B_1^i))$.

And by definition of B_1^i , we have $(x_i\sigma_1, y_i\rho_1) \in \mathcal{S}(\mathcal{I}(h_i(\sigma_1, \rho_1) \cup B_1^i))$ (i.e. the names of $B_1 \setminus B_1^i$ are irrelevant to synthesise $(x\sigma_1, y\rho_1)$).

Let $(b_1, b_2) \in B_1^i$. We have that $(b_1\sigma_2, b_2\sigma_2) \in \mathcal{S}(\mathcal{I}(\mathbf{se}_1|_{(b_1, b_2)}(\sigma_2, \rho_2) \cup B_2))$.

Let $(M\sigma_1, N\rho_1) \in \mathbf{se}_1|_{(b_1, b_2)}$ where $(M, N) \in h$. We have $b_1 \in n(x_j\sigma)$ or $b_2 \in n(y_j\sigma)$ for some $j \leq i$. So by definition, we have $(M, N) \prec (x_j, y_j)$. Since \mathbf{se} is growing, we have also $(M, N) \prec (x_i, y_i)$. So $(M, N) \in \mathbf{se}|_{(x_i, y_i)} = h_i$.

Thus $(b_1\sigma_2, b_2\sigma_2) \in \mathcal{S}(\mathcal{A}(h_i(\sigma_1, \rho_1)(\sigma_2, \rho_2) \cup B_2))$ for every $(b_1, b_2) \in B_1^i$.

So by Lemma 65, we get $(M\sigma_1\sigma_2, N\sigma_1\sigma_2) \in \mathcal{S}(\mathcal{A}(h_i(\sigma_1, \rho_1)(\sigma_2, \rho_2) \cup B_2))$, i.e. $(M\sigma, N\rho) \in \mathcal{S}(\mathcal{A}(h_i(\sigma, \rho) \cup B_2))$.

7. Let $x \in \gamma_l$. We have $x\sigma_1 \in N$. If $x\sigma_1 \in \pi_1(B_1)$, then we have $x\sigma_1\sigma_2 = x\sigma \in N$. If $x\sigma_1 \notin \pi_1(B_1)$, then $x\sigma_1\sigma_2 = x\sigma_1 \in N$.
8. Similarly, if $y \in \gamma_r$, then $y\rho \in N$.

9. We note $(h_1, B_1, \prec_1, (\gamma_l^1, \gamma_r^1)) = \mathbf{se}_1$, $(h_2, B_2, \prec_2, (\gamma_l^2, \gamma_r^2)) = \mathbf{se}_2$ and $(h', B_2, \prec', (\gamma_l', \gamma_r')) = \mathbf{se}_{B_2}^{\sigma, \rho} = \mathbf{se}'$.

We have $h_1 = h(\sigma_1, \rho_1)$, $h_2 = h_1(\sigma_2, \rho_2) = h(\sigma_1, \rho_1)(\sigma_2, \rho_2)$ and $h' = h(\sigma, \rho)$. According to the previous results, we have $h_2 = h'$.

If $x_2 \in \gamma_l^2$ then $x_2 = x_1\sigma_2$ for some $x_1 \in \gamma_l^1$. Since $x_1 \in \gamma_l^1$, there exists $x \in \gamma_l$ such that $x_1 = x\sigma_1$. We have $x_2 = x\sigma_1\sigma_2 = x\sigma \in \sigma(\gamma_l)$. Moreover $x_2 \in \pi_1(B_2)$ so $x_2 \in \gamma_l'$.

Conversely, if $x_2 \in \gamma_l'$, there is $x \in \gamma_l$ such that $x_2 = x\sigma = x\sigma_1\sigma_2$. Since $x\sigma_1\sigma_2 = x \in N$, we have $x\sigma_1 \in N$. Necessarily, $x\sigma_1 \in \pi_1(B_1)$ otherwise $x\sigma_1\sigma_2 \notin \pi_1(B_2)$ which would be a contradiction. So $x_2 \in \gamma_l^2$.

Hence $\gamma_l' = \gamma_l^2$ and $\gamma_r' = \gamma_r^2$.

It remains to show that $\prec' = \prec_2$.

Since \mathbf{se} is growing, we write $v = \{(x_1, y_1), \dots, (x_n, y_n)\}$ such that if $h_i := \mathbf{se}|_{(x_i, y_i)}$ we have $h_i \subseteq h_{i+1}$ for $1 \leq i < n$.

Since \mathbf{se}_1 is growing, we write $B_1 = \{(x'_1, y'_1), \dots, (x'_p, y'_p)\}$ such that if $h'_i := \mathbf{se}_1|_{(x'_i, y'_i)}$ we have $h'_i \subseteq h'_{i+1}$ for $1 \leq i < p$.

According to proof of Lemma 63, $(M\sigma_1\sigma_2, N\rho_1\rho_2) \prec_2 (x'', y'')$ if and only if $(M\sigma_1, N\rho_1) \prec_1 (x'_i, y'_i)$ where i is the minimal index such that $x'' \in n(x'_i\sigma_2)$ or $y'' \in n(y'_i\rho_2)$ (where $(M, N) \in h$ and $(x'', y'') \in B_2$).

Similarly, $(M\sigma, N\rho) \prec' (x'', y'')$ if and only if $(M, N) \prec (x_i, y_i)$ where i is the minimal index such that $x'' \in n(x_i\sigma)$ or $y'' \in n(y_i\rho)$ (where $(M, N) \in h$ and $(x'', y'') \in B_2$).

Assume that $(M\sigma, N\rho) \prec' (x'', y'')$. Let i minimal such that $x'' \in n(x'_i\sigma_2)$ or $y'' \in n(y'_i\rho_2)$. According to proof of Lemma 64, we have that $x'' \in n(x'_i\sigma_2)$ and $y'' \in n(y'_i\rho_2)$ (because the S-environments are well-formed and growing). Now let j minimal such that $x'_i \in n(x_j\sigma_1)$ or $y'_i \in n(y_j\rho_1)$. Similarly, we have that $x'_i \in n(x_j\sigma_1)$ and $y'_i \in n(y_j\rho_1)$. So $x'' \in n(x_j\sigma)$ and $y'' \in n(y_j\rho)$. So we have $(M, N) \prec (x_j, y_j)$. Thus $(M\sigma_1, N\rho_1) \prec_1 (x'_j, y'_j)$ and finally $(M\sigma_1\sigma_2, N\rho_1\rho_2) \prec_2 (x'', y'')$, i.e. $(M\sigma, N\rho) \prec_2 (x'', y'')$. We conclude that $\prec' \subseteq \prec_2$.

Assume now that $(M\sigma_1\sigma_2, N\rho_1\rho_2) \prec_2 (x'', y'')$. Let i minimal such that $x'' \in n(x_i\sigma)$ or $y'' \in n(y_i\rho)$. We have $x'' \in n(x_i\sigma)$ and $y'' \in n(y_i\rho)$. Necessarily, there is j such that $x'_j \in n(x_i\sigma_1)$ and $x'' \in n(x'_j\sigma_2)$. We then have $(M\sigma_1, N\rho_1) \prec_1 (x'_j, y'_j)$. Hence $(M, N) \prec (x_i, y_i)$. So $(M\sigma, N\rho) \prec' (x'', y'')$. We conclude that $\prec_2 \subseteq \prec'$.

Finally, we have shown that $\mathbf{se}' = \mathbf{se}_2$. ■

Finally, as for hedges, a notion of consistency for S-environments is needed: under every possible substitution, the attacker is unable to get a contradiction from its updated knowledge.

Definition 76 (consistency).

A S-environment \mathbf{se} is *consistent* if it is well-formed, growing and for all σ, ρ and B such that $(\sigma, \rho) \triangleright_B \mathbf{se}$, we have

1. $\mathcal{I}(h' \cup v') = \mathcal{I}(\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)}))$ is a consistent hedge
2. $\forall (x, y) \in v' : x \in \gamma'_l \iff y \in \gamma'_r$

where $(h', v', \prec', (\gamma'_l, \gamma'_r)) = \mathbf{se}_B^{(\sigma, \rho)}$.

For well-formed and growing S-environments, this can be slightly simplified according to the following lemma.

Lemma 66 (consistency of well-formed and growing S-environments):

Let $\mathbf{se} = (h, v, \prec, (\gamma_l, \gamma_r))$ be a S-environment. Then \mathbf{se} is consistent if and only if it is well-formed, growing and for all σ, ρ and B such that $(\sigma, \rho) \triangleright_B \mathbf{se}$, we have

1. $\mathcal{I}(h(\sigma, \rho) \cup B)$ is a consistent hedge

$$2. \forall (x, y) \in v : x \in \gamma_l \iff y \in \gamma_r$$

PROOF

\Rightarrow Trivial.

\Leftarrow Let σ, ρ and B such that $(\sigma, \rho) \triangleright_B \mathbf{se}$. Let $(h', v', \prec', (\gamma'_l, \gamma'_r)) = \mathbf{se}_B^{(\sigma, \rho)}$.

By definition, $h' = h((\sigma, \rho))$ and $v' = B$. So $\mathcal{I}(h' \cup v')$ is consistent.

Let $(x', y') \in v' = B$. Assume that $x' \in \gamma'_l$. Since $\gamma'_l = \sigma(\gamma_l) \cap \pi_1(B)$, there exists $x \in \gamma_l$ such that $x' = x\sigma$. Since $x \in \gamma_l \subseteq \pi_1(v)$, there is y such that $(x, y) \in v$.

By hypothesis, we have $(x\sigma, y\rho) \in \mathcal{S}(\mathcal{I}(\mathbf{se}|_{(x,y)}((\sigma, \rho)) \cup B))$. Since $\mathbf{se}|_{(x,y)} \subseteq h$, we have by Lemma 49 that $(x\sigma, y\rho) \in \mathcal{S}(\mathcal{I}(h((\sigma, \rho)) \cup B))$.

Since $x\sigma = x' \in N$, we have $(x\sigma, y\rho) \in \mathcal{I}(h((\sigma, \rho)) \cup B)$. Since $\mathcal{I}(h((\sigma, \rho)) \cup B)$ is consistent and $(x', y') \in B$ and $x\sigma = x'$, we have $y\rho = y'$. Moreover, since $(x, y) \in v$ and $x \in \gamma_l$, we have $y \in \gamma_r$. Thus $y' \in \gamma'_r$.

By symmetry, if $y' \in \gamma'_r$ then $x' \in \gamma'_l$.

So \mathbf{se} is consistent. ■

5.2.2 Open hedged bisimulation

An *open hedged relation* \mathcal{R} is a subset of $S_H \times P \times P$ such that for $(\mathbf{se}, P, Q) \in \mathcal{R}$, we have $\text{fn}(P) \subseteq n(\pi_1(\mathfrak{H}(\mathbf{se})))$ and $\text{fn}(Q) \subseteq n(\pi_2(\mathfrak{H}(\mathbf{se})))$. It is *consistent* if, for every $(\mathbf{se}, P, Q) \in \mathcal{R}$, \mathbf{se} is consistent. It is *symmetric* if for every $(\mathbf{se}, P, Q) \in \mathcal{R}$ we have $(\mathbf{se}^{-1}, Q, P) \in \mathcal{R}$.

Definition 77 (open hedged bisimulation).

A symmetric consistent open hedged relation \mathcal{R} is an *open hedged bisimulation* if for all $(\mathbf{se}, P, Q) \in \mathcal{R}$, for all σ, ρ and B such that $(\sigma, \rho) \triangleright_B \mathbf{se}$,

1. if $P\sigma \xrightarrow{S_1} P'$ then

there exist Q' and S_2 such that $Q\rho \xrightarrow{S_2} Q'$

and $(\mathbf{se}_B^{(\sigma, \rho)} +_c(S_1, S_2), P', Q') \in \mathcal{R}$

2. if $P\sigma \xrightarrow{a}_{S_1} (x)P'$ (with $x \notin \mathfrak{n}(\pi_1(\mathfrak{H}(\mathbf{se}_B^{(\sigma,\rho)}))))$
 and $(a, b) \in \mathcal{S}(\mathcal{I}(\mathfrak{H}(\mathbf{se}_B^{(\sigma,\rho)})))$ then
 there exist y, Q' and S_2 such that $Q\rho \xrightarrow{b}_{S_2} (y)Q'$
 (with $y \notin \mathfrak{n}(\pi_2(\mathfrak{H}(\mathbf{se}_B^{(\sigma,\rho)}))))$
 and $(\mathbf{se}_B^{(\sigma,\rho)} +_1(x, y) +_c(S_1, S_2), P', Q') \in \mathcal{R}$
3. if $P\sigma \xrightarrow{\bar{a}}_{S_1} (v\bar{c}) \langle M \rangle P'$ (with $\{\bar{c}\} \cap \mathfrak{n}(\pi_1(\mathfrak{H}(\mathbf{se}_B^{(\sigma,\rho)}))) = \emptyset$)
 and $(a, b) \in \mathcal{S}(\mathcal{I}(\mathfrak{H}(\mathbf{se}_B^{(\sigma,\rho)})))$ then
 there exist \bar{d}, N, Q' and S_2 such that $Q\rho \xrightarrow{\bar{b}}_{S_2} (v\bar{d}) \langle N \rangle Q'$
 (with $\{\bar{d}\} \cap \mathfrak{n}(\pi_2(\mathfrak{H}(\mathbf{se}_B^{(\sigma,\rho)}))) = \emptyset$)
 and $(\mathbf{se}_B^{(\sigma,\rho)} +_o(M, N) +_c(S_1, S_2), P', Q') \in \mathcal{R}$

In any case, names used as channels (collected in S_1, S_2) are added to the environment's γ_l and γ_r . On inputs (clause 2), input names are added to the environment's v . On outputs (clause 3), messages are added to the environment's h .

Let $\mathbf{se} \in \mathcal{S}_H$ and $P, Q \in \mathcal{P}$. We say that P and Q are *open hedged bisimilar under \mathbf{se}* —written $P \sim_{\text{OH}}^{\mathbf{se}} Q$ —if there exists an open hedged bisimulation \mathcal{R} such that $(\mathbf{se}, P, Q) \in \mathcal{R}$.

5.2.3 Soundness of open hedged bisimulation

We are going to show that open hedged bisimulation is sound w.r.t. late hedged bisimulation. Before proving this theorem, we show some auxiliary results.

Recall that ϵ denotes the substitution with an empty support.

Lemma 67:

Let $\mathbf{se} = (h, v, \prec, (\gamma_l, \gamma_r))$ be a S -environment.

Then $(\epsilon, \epsilon) \triangleright_v \mathbf{se}$ and $\mathbf{se}_v^{(\epsilon, \epsilon)} = \mathbf{se}$. □

Lemma 68:

Let $\mathbf{se} = (h, v \cup \{(x, y)\}, \prec, (\gamma_l, \gamma_r))$ be a well-formed S -environment such that $(x, y) \notin v$, $x \notin \gamma_l$, $y \notin \gamma_r$ and such that for all $(M, N) \in h$, we have $(M, N) \prec (x, y)$.

Let M, N and B such that $\mathcal{I}(h \cup v) \vdash_B (M, N)$.

Then $(\{M/x\}, \{N/y\}) \triangleright_{B \cup v} \mathbf{se}$ and $\mathcal{I}(\mathfrak{H}(\mathbf{se}_{B \cup v}^{\{\{M/x\}, \{N/y\}\}})) = \mathcal{I}(h \cup v) \cup B$.

PROOF

1. Obviously $\text{supp}(\{M/x\}) \subseteq \pi_1(v \cup \{(x, y)\})$.
2. Similarly, $\text{supp}(\{N/y\}) \subseteq \pi_2(v \cup \{(x, y)\})$.
3. Let $(b_1, b_2) \in B \cup v$.
 If $(b_1, b_2) \in v$ then $b_1 \in \mathfrak{n}(\{M/x\}(\pi_1(v))) = \mathfrak{n}(\pi_1(v)) \subseteq \mathfrak{n}(\pi_1(v \cup \{(x, y)\}))$.
 Otherwise, if $(b_1, b_2) \in B$, then we have by hypothesis $b_1 \in \mathfrak{n}(M)$ and $b_2 \in \mathfrak{n}(N)$. So $b_1 \in \mathfrak{n}(\{M/x\}(\pi_1(v \cup \{(x, y)\}))) = \mathfrak{n}(M) \cup \mathfrak{n}(\pi_1(v))$.
4. By hypothesis, $\pi_1(B) \cap \mathfrak{n}(\pi_1(h \cup v)) = \emptyset$. So $\pi_1(v \cup B) \cap (\mathfrak{n}(\pi_1(h)) \setminus \pi_1(v \cup \{(x, y)\})) = \emptyset$.
5. Similarly $\pi_2(v \cup B) \cap (\mathfrak{n}(\pi_2(h)) \setminus \pi_2(v \cup \{(x, y)\})) = \emptyset$.
6. Let $(x', y') \in v \cup \{(x, y)\}$.
 If $(x', y') \in v$ then $x' \{M/x\} = x'$ and $y' \{N/y\} = y'$. Hence the condition on $(x' \{M/x\}, y' \{N/y\})$ is clearly satisfied.
 Or $(x', y') = (x, y)$. Then $\mathfrak{se}|_{(x, y)} = h$. Since \mathfrak{se} is well-formed, we have $(*) h(\{M/x\}, \{N/y\}) = h$. And by hypothesis, $(M, N) \in \mathcal{S}(h \cup v \cup B)$.
7. Since $x \notin \gamma_l$, $\{M/x\}$ satisfies the condition for γ_l .
8. Similarly for $\{N/y\}$ and γ_r .
 Finally, $(\{M/x\}, \{N/y\}) \triangleright_{B \cup v} \mathfrak{se}$.
 Moreover, we have $\mathfrak{H}(\mathfrak{se}_{B \cup v}^{\{\{M/x\}, \{N/y\}\}}) = h \cup B \cup v$ because $(*)$ holds.
 So $\mathcal{I}(\mathfrak{H}(\mathfrak{se}_{B \cup v}^{\{\{M/x\}, \{N/y\}\}})) = \mathcal{I}(h \cup v \cup B)$.
 By Theorem 7, $\mathcal{I}(h \cup v \cup B) = \mathcal{I}(\mathcal{I}(h \cup v) \cup B)$. And since B is a consistent hedge of fresh names, we have $\mathcal{I}(\mathcal{I}(h \cup v) \cup B) = \mathcal{I}(h \cup v) \cup B$. ■

We can now show the soundness theorem. The following theorem thus states that open hedged bisimulation is a sound proof technique for showing that two processes are late hedged bisimilar.

Theorem 12:

Assume that $P \sim_{OH}^{\mathfrak{se}} Q$ and let σ, ρ and B such that $(\sigma, \rho) \triangleright_B \mathfrak{se}$.

Then $P\sigma \sim_{LH}^{\mathcal{I}(\mathfrak{H}(\mathfrak{se}_B^{(\sigma, \rho)}))} Q\rho$.

PROOF

Let \mathcal{R} an open hedged bisimulation such that $(se, P, Q) \in \mathcal{R}$.

Define $\mathcal{R}' = \left\{ (\mathcal{I}(\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)})), P\sigma, Q\rho) \mid (se, P, Q) \in \mathcal{R} \wedge (\sigma, \rho) \triangleright_B se \right\}$

Then \mathcal{R}' is a late hedged bisimulation.

First \mathcal{R}' is clearly a symmetric consistent hedged relation because \mathcal{R} is a symmetric open hedged relation.

Let $(\mathcal{I}(\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)})), P\sigma, Q\rho) \in \mathcal{R}'$ with $(se, P, Q) \in \mathcal{R}$ and $(\sigma, \rho) \triangleright_B se$.

Assume that $P\sigma \xrightarrow{\tau} P'$. Then there exists S_1 such that $P\sigma \xrightarrow[S_1]{\tau} P'$. So

there exist S_2 and Q' such that $Q\rho \xrightarrow[S_2]{\tau} Q'$ and $(\mathbf{se}_B^{(\sigma, \rho)} +_c(S_1, S_2), P', Q') \in$

\mathcal{R} . Thus, by Lemma 67, we get that $(\mathcal{I}(\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)})), P', Q') \in \mathcal{R}'$ and we have $Q\rho \xrightarrow{\tau} Q'$.

Assume that $P\sigma \xrightarrow{a} (x)P'$ with $x \notin n(\pi_1(\mathcal{I}(\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)}))))$ and that $(a, b) \in \mathcal{S}(\mathcal{I}(\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)})))$. Then there exists S_1 such that $P\sigma \xrightarrow[S_1]{a} (x)P'$.

Since $x \notin n(\pi_1(\mathcal{I}(\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)}))))$, we have $x \notin n(\pi_1(\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)})))$. So there exist y , Q' and S_2 such that $Q\rho \xrightarrow[S_2]{b} (y)Q'$ with $y \notin n(\pi_2(\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)})))$ and $(\mathbf{se}_B^{(\sigma, \rho)} +_i(x, y) +_c(S_1, S_2), P', Q') \in \mathcal{R}$. So $Q\rho \xrightarrow{b} (y)Q'$ with $y \notin n(\pi_2(\mathcal{I}(\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)}))))$.

Let M, N and B' such that $\mathcal{I}(\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)})) \vdash_{B'} (M, N)$.

We apply Lemma 68 with $\mathbf{se}_B^{(\sigma, \rho)} +_i(x, y) +_c(S_1, S_2)$, (x, y) , M , N and B' . Note that by definition the pair (x, y) satisfies the required condition and that $\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)})$ plays the role of $h \cup v$.

Thus $(\mathcal{I}(\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)})) \cup B', P'\{M/x\}, Q'\{N/x\}) \in \mathcal{R}'$.

Assume that $P\sigma \xrightarrow{\vec{a}} (v\vec{c}) \langle M \rangle P'$ with $\{\vec{c}\} \cap n(\pi_1(\mathcal{I}(\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)})))) = \emptyset$ and that $(a, b) \in \mathcal{S}(\mathcal{I}(\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)})))$. Then there exists S_1 such that $P\sigma \xrightarrow[S_1]{\vec{a}}$

$(v\vec{c}) \langle M \rangle P'$. Since $\{\vec{c}\} \cap n(\pi_1(\mathcal{I}(\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)})))) = \emptyset$, we also have that $\{\vec{c}\} \cap n(\pi_1(\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)})))) = \emptyset$. So there exist \vec{d} , N , Q' and S_2 such that

$Q\rho \xrightarrow[S_2]{\vec{b}} (v\vec{d}) \langle N \rangle Q'$ with $\{\vec{d}\} \cap n(\pi_2(\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)})))) = \emptyset$ and we have that

$(\mathbf{se}_B^{(\sigma, \rho)} +_o(M, N) +_c(S_1, S_2), P', Q') \in \mathcal{R}$.

So $Q\rho \xrightarrow{\vec{b}} (v\vec{d}) \langle N \rangle Q'$ with $\{\vec{d}\} \cap n(\pi_2(\mathcal{I}(\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)})))) = \emptyset$. And by Lemma 67, we have $(\mathcal{I}(\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)})) \cup \{(M, N)\}), P', Q') \in \mathcal{R}'$.

So $(\mathcal{I}(\mathcal{I}(\mathfrak{H}(\mathbf{se}_B^{(\sigma,\rho)}))) \cup \{(M, N)\}), P', Q' \in \mathcal{R}'$ thanks to Theorem 7.
Hence \mathcal{R}' is a late hedged bisimulation. \blacksquare

5.3 Up to techniques

Up to techniques [128] are powerful proof techniques for showing bisimilarity results. These techniques have first been adapted to the spi calculus in [36]. They rely on the notion of progress.

Definition 78 (progress).

Let \mathcal{R} and \mathcal{R}' be two symmetric consistent open hedged relations. We say that \mathcal{R} *progresses to* \mathcal{R}' , and write $\mathcal{R} \rightsquigarrow \mathcal{R}'$, if for all $(\mathbf{se}, P, Q) \in \mathcal{R}$, for all σ, ρ and B such that $(\sigma, \rho) \triangleright_B \mathbf{se}$,

1. if $P\sigma \xrightarrow[S_1]{\tau} P'$ then
there exist Q' and S_2 such that $Q\rho \xrightarrow[S_2]{\tau} Q'$
and $(\mathbf{se}_B^{(\sigma,\rho)} +_c(S_1, S_2), P', Q') \in \mathcal{R}'$
2. if $P\sigma \xrightarrow[S_1]{a} (x)P'$ (with $x \notin \mathfrak{n}(\pi_1(\mathfrak{H}(\mathbf{se}_B^{(\sigma,\rho)})))$)
and $(a, b) \in \mathcal{S}(\mathcal{I}(\mathfrak{H}(\mathbf{se}_B^{(\sigma,\rho)})))$ then
there exist y, Q' and S_2 such that $Q\rho \xrightarrow[S_2]{b} (y)Q'$
(with $y \notin \mathfrak{n}(\pi_2(\mathfrak{H}(\mathbf{se}_B^{(\sigma,\rho)})))$)
and $(\mathbf{se}_B^{(\sigma,\rho)} +_i(x, y) +_c(S_1, S_2), P', Q') \in \mathcal{R}'$
3. if $P\sigma \xrightarrow[S_1]{\bar{a}} (v\bar{c}) \langle M \rangle P'$ (with $\{\bar{c}\} \cap \mathfrak{n}(\pi_1(\mathfrak{H}(\mathbf{se}_B^{(\sigma,\rho)}))) = \emptyset$)
and $(a, b) \in \mathcal{S}(\mathcal{I}(\mathfrak{H}(\mathbf{se}_B^{(\sigma,\rho)})))$ then
there exist \bar{d}, N, Q' and S_2 such that $Q\rho \xrightarrow[S_2]{\bar{b}} (v\bar{d}) \langle N \rangle Q'$
(with $\{\bar{d}\} \cap \mathfrak{n}(\pi_2(\mathfrak{H}(\mathbf{se}_B^{(\sigma,\rho)}))) = \emptyset$)
and $(\mathbf{se}_B^{(\sigma,\rho)} +_o(M, N) +_c(S_1, S_2), P', Q') \in \mathcal{R}'$

Clearly, \mathcal{R} is an open hedged bisimulation if and only if $\mathcal{R} \rightsquigarrow \mathcal{R}$.
Moreover, it is obvious that if $\mathcal{R} \rightsquigarrow \mathcal{R}'$ and $\mathcal{R}' \subseteq \mathcal{R}''$ then $\mathcal{R} \rightsquigarrow \mathcal{R}''$.

Given a function \mathcal{F} on open hedged relations, we say that \mathcal{R} is an open hedged bisimulation *up to* \mathcal{F} if $\mathcal{R} \rightsquigarrow \mathcal{F}(\mathcal{R})$.

We say that an open hedged relation \mathcal{R} is *sound* if there exists an open hedged bisimulation that contains \mathcal{R} . Observe then that if \mathcal{R} is sound, we have for any $(se, P, Q) \in \mathcal{R}$ that $P \dot{\sim}_{OH}^{se} Q$.

Definition 79 (safeness).

A function \mathcal{F} is *safe* if, for any \mathcal{R} , $\mathcal{R} \rightsquigarrow \mathcal{F}(\mathcal{R})$ implies that \mathcal{R} is sound.

Definition 80 (respectfulness).

A function \mathcal{F} is *respectful* if whenever $\mathcal{R} \subseteq \mathcal{R}'$ and $\mathcal{R} \rightsquigarrow \mathcal{R}'$ holds then $\mathcal{F}(\mathcal{R}) \subseteq \mathcal{F}(\mathcal{R}')$ and $\mathcal{F}(\mathcal{R}) \rightsquigarrow \mathcal{F}(\mathcal{R}')$ also holds.

It is possible to adapt to our setting the following lemma taken from [128, 129].

Lemma 69:

If \mathcal{F} is respectful and $\mathcal{R} \rightsquigarrow \mathcal{F}(\mathcal{R})$ then \mathcal{R} and $\mathcal{F}(\mathcal{R})$ are sound. \square

Hence respectful functions are safe.

Thus, if \mathcal{F} is respectful, $\mathcal{R} \rightsquigarrow \mathcal{R}$ (i.e. \mathcal{R} is a bisimulation) and $\mathcal{R} \subseteq \mathcal{F}(\mathcal{R})$ then $\mathcal{F}(\mathcal{R})$ is sound.

5.3.1 Up to structural congruence

\mathcal{R} is an open hedged bisimulation up to structural congruence if it is an open hedged bisimulation up to \mathcal{F}_{\equiv} where

$$\mathcal{F}_{\equiv}(\mathcal{R}) := \left\{ (se, P', Q') \mid \left\{ \begin{array}{l} (se, P, Q) \in \mathcal{R} \\ P \equiv P' \text{ and } Q \equiv Q' \end{array} \right\} \right\}$$

Lemma 70:

1. for all \mathcal{R} , $\mathcal{R} \subseteq \mathcal{F}_{\equiv}(\mathcal{R})$
2. \mathcal{F}_{\equiv} is respectful.

PROOF

1. Trivial.
2. Straightforward since structural congruence preserves one-step transitions. Note also that if $P \equiv P'$, we have $\text{fn}(P) = \text{fn}(P')$ and thus $\mathcal{F}_{\equiv}(\mathcal{R})$ is an open hedged relation if \mathcal{R} is one. \blacksquare

5.3.2 Up to bijective renamings

Definition 81 (renaming of a S-environment).

Given a S-environment $\mathbf{se} = (h, v, \prec, (\gamma_l, \gamma_r))$ and two bijective renamings θ_1 and θ_2 , we define

$$\mathbf{se}(\theta_1, \theta_2) := (h(\theta_1, \theta_2), v(\theta_1, \theta_2), \prec', (\gamma_l\theta_1, \gamma_r\theta_2))$$

where \prec' is defined by

$$(M\theta_1, N\theta_2) \prec' (x\theta_1, y\theta_2) \iff (M, N) \prec (x, y)$$

Clearly well-formedness, growth and consistency of S-environments are preserved by renamings.

\mathcal{R} is an open hedged bisimulation up to bijective renamings if it is an open hedged bisimulation up to \mathcal{F}_i where

$$\mathcal{F}_i(\mathcal{R}) := \left\{ (\mathbf{se}(\theta_1, \theta_2), P\theta_1, Q\theta_2) \mid \left\{ \begin{array}{l} (\mathbf{se}, P, Q) \in \mathcal{R} \\ \theta_1 \text{ is a bijective renaming} \\ \theta_2 \text{ is a bijective renaming} \end{array} \right\} \right\}$$

Lemma 71:

1. for all \mathcal{R} , $\mathcal{R} \subseteq \mathcal{F}_i(\mathcal{R})$.
2. \mathcal{F}_i is respectful.

PROOF

1. Trivial.
2. Straightforward since bijective renamings preserves one-step transitions and since if $\mathbf{se} = (h, v, \prec, (\gamma_l, \gamma_r))$ and θ_1, θ_2 are bijective renamings we have that if $(\sigma, \rho) \triangleright_B \mathbf{se}(\theta_1, \theta_2)$ then $(\sigma', \rho') \triangleright_{B(\theta_1^{-1}, \theta_2^{-1})} \mathbf{se}$ where

$$x\sigma' = \begin{cases} x\theta_1\sigma\theta_1^{-1} & \text{if } x \in \pi_1(v) \\ x & \text{otherwise} \end{cases} \text{ and } y\rho' = \begin{cases} y\theta_2\rho\theta_2^{-1} & \text{if } y \in \pi_2(v) \\ y & \text{otherwise} \end{cases}$$

and then $\theta_1\sigma = \sigma'\theta_1$ and $\theta_2\rho = \rho'\theta_2$. ■

5.3.3 Up to respectful substitutions

\mathcal{R} is an open hedged bisimulation up to respectful substitutions if it is an open hedged bisimulation up to \mathcal{F}_s where

$$\mathcal{F}_s(\mathcal{R}) := \left\{ (\mathbf{se}_B^{(\sigma, \rho)}, P\sigma, Q\rho) \mid \left\{ \begin{array}{l} (\mathbf{se}, P, Q) \in \mathcal{R} \\ (\sigma, \rho) \triangleright_B \mathbf{se} \end{array} \right\} \right\}$$

Lemma 72:

1. for all \mathcal{R} , $\mathcal{R} \subseteq \mathcal{F}_s(\mathcal{R})$.
2. \mathcal{F}_s is respectful.

PROOF

1. Trivial.
2. This follows from the fact that composition of respectful substitutions yield respectful substitutions (see Theorem 11). ■

5.3.4 Open hedged bisimulation is an extension of K-open bisimulation

As a direct application of up to techniques presented above, we shall show that open hedged bisimulation is an extension of K-open bisimulation. This answers the question: assume that $P, Q \in \mathcal{P}$ are pi calculus processes that are open hedged bisimilar, are they also K-open bisimilar? The answer is yes with some conditions that we detail now.

Definition 82.

Let $\rho e = (O, V, \prec)$ be a K-environment. It induces the following set of S-environments.

$$\langle \rho e \rangle = \{ (h, v, \prec', (\gamma, \gamma)) \mid \gamma \subseteq V \}$$

where

$$\begin{aligned} h &:= \{ (n, n) \mid n \in O \} \\ v &:= \{ (x, x) \mid x \in V \} \\ \prec' &:= \{ ((n, n), (x, x)) \mid (n, x) \in \prec \} \end{aligned}$$

Clearly, every member of $\langle \rho e \rangle$ is a S-environment. The next lemma studies the well-formedness and growth of induced S-environments.

Lemma 73:

Let pe be a K-environment. Let $se \in \langle pe \rangle$.

1. Then se is well-formed.
2. Moreover, pe is growing iff se is growing.

PROOF

Let $(O, V, \prec) = pe$ and $(h, v, \prec', (\gamma, \gamma)) = se$.

1. Since by hypothesis $O \cap V = \emptyset$.
2. Trivial. ■

A substitution that respects a K-environment induces a pair of substitutions that respect any induced S-environments, as stated by the following lemma.

Lemma 74:

Let pe be a K-environment. Let $se \in \langle pe \rangle$.

Let σ a substitution such that $\sigma \blacktriangleright pe$.

Then $(\sigma, \sigma) \triangleright_B se$ where

$$B := \{(b, b) \mid b \in V'\}$$

$$V' := (V \setminus \text{supp}(\sigma)) \cup \{x\sigma \mid x \in \text{supp}(\sigma) \wedge x\sigma \notin O\}$$

Moreover $se_B^{(\sigma, \sigma)} \in \langle pe^\sigma \rangle$.

PROOF

By hypothesis σ maps names to names, i.e. $\sigma : N \rightarrow N$.

Let $(O, V, \prec) = pe$ and $(h, v, \prec', (\gamma, \gamma)) = se$.

1. By hypothesis, we have $\text{supp}(\sigma) \subseteq V$, so since $\pi_1(v) = V$, we have $\text{supp}(\sigma) \subseteq \pi_1(v)$.
2. Similarly, $\text{supp}(\sigma) \subseteq \pi_2(v)$.
3. Let $(b, b) \in B$, i.e. let $b \in V'$.

If $b \in V \setminus \text{supp}(\sigma)$ then $b \in V$ and $b\sigma = b$. Thus $b \in n(\sigma(V))$.

Otherwise, if $b \in \{x\sigma \mid x \in \text{supp}(\sigma) \wedge x\sigma \notin O\}$. Then clearly $b \in n(\sigma(V))$.

Thus, in both cases, we have $b \in n(\sigma(\pi_1(v)))$
(and also $b \in n(\sigma(\pi_2(v)))$).

4. By contradiction, assume that $\pi_1(B) \cap (\mathfrak{n}(\pi_1(h)) \setminus \pi_1(v)) \neq \emptyset$.

There exists $b \in V'$ such that $b \in O$ and $b \notin V$.

Since $b \in V'$, we have two cases:

- $b \in V \setminus \text{supp}(\sigma)$. This case is impossible since $b \notin V$.
- $b \in \{x\sigma \mid x \in \text{supp}(\sigma) \wedge x\sigma \notin O\}$. Then $b = x\sigma \notin O$. Contradiction.

Thus $\pi_1(B) \cap (\mathfrak{n}(\pi_1(h)) \setminus \pi_1(v)) = \emptyset$.

5. Similarly $\pi_2(B) \cap (\mathfrak{n}(\pi_2(h)) \setminus \pi_2(v)) = \emptyset$.

6. Let $(x, x) \in v$, i.e. let $x \in V$.

Clearly $h(\sigma, \sigma) = h$.

By hypothesis, we have that if $x\sigma \in O$ then $x\sigma \prec x$.

Otherwise, if $x\sigma \notin O$ then $x\sigma \in V'$.

Thus $(x\sigma, x\sigma) \in \mathcal{S}(\mathcal{I}(\mathbf{se}|_{(x,x)}(\sigma, \sigma) \cup B))$.

7. Let $x \in \gamma$. Then since $\sigma : \mathbf{N} \rightarrow \mathbf{N}$, we have $x\sigma \in \mathbf{N}$.

8. Idem.

Hence $(\sigma, \sigma) \triangleright_B \mathbf{se}$.

By definition, $\mathbf{se}_B^{(\sigma, \sigma)} = (h', B, \prec^2, (\gamma'_l, \gamma'_r))$ where

$$h' := h(\sigma, \sigma) = h$$

$$\gamma'_l := \sigma(\gamma) \cap \pi_1(B) = \sigma(\gamma) \cap V' = \gamma' \subseteq V'$$

$$\gamma'_r := \sigma(\gamma) \cap \pi_2(B) = \sigma(\gamma) \cap V' = \gamma'$$

And \prec^2 is defined for $(n, n) \in h$ and $(x', x') \in B$ by

$$\begin{aligned} (n\sigma, n\sigma) \prec^2 (x', x') &\iff \bigwedge_{\substack{x \in V \\ x' \in \mathfrak{n}(x\sigma)}} (M, M) \prec' (x, x) \\ &\iff \bigwedge_{\substack{x \in V \\ x' \in \mathfrak{n}(x\sigma)}} n \prec x \end{aligned}$$

i.e. since $n\sigma = n$, we have

$$(n, n) \prec^2 (x', x') \iff \forall x \in V : x' \in \mathfrak{n}(x\sigma) \implies n \prec x$$

Hence $\mathbf{se}_B^{(\sigma, \sigma)} \in \langle \mathbf{pe}^\sigma \rangle$. ■

The following proposition states that open hedged bisimulation is an extension of K-open bisimulation.

Proposition 4:

Let $P, Q \in \mathcal{P}$ two pi calculus processes such that $P \sim_{\text{OH}}^{\text{se}} Q$.

If $\text{se} \in \langle \text{pe} \rangle$ for some K-environment pe then $P \sim_K^{\text{pe}} Q$.

PROOF (SKETCH)

We define

$$\mathcal{R} := \left\{ (\text{pe}, P, Q) \mid \left\{ \begin{array}{l} P \sim_{\text{OH}}^{\text{se}} Q \\ P, Q \text{ are pi calculus processes} \\ \text{se} \in \langle \text{pe} \rangle \end{array} \right\} \right\}$$

We show that \mathcal{R} is a K-open bisimulation.

By Lemma 73, \mathcal{R} is clearly a symmetric K-relation.

Let $(\text{pe}, P, Q) \in \mathcal{R}$ and σ such that $\sigma \blacktriangleright \text{pe}$.

By hypothesis, we have $P \sim_{\text{OH}}^{\text{se}} Q$ for some $\text{se} \in \langle \text{pe} \rangle$.

By Lemma 74, we have $(\sigma, \sigma) \triangleright_B \text{se}$ for some B such that $\text{se}_B^{(\sigma, \sigma)} \in \langle \text{pe}^\sigma \rangle$.

Assume that $P\sigma \xrightarrow{\mu} P'$ with $\text{bn}(\mu) \cap (\text{n}(\text{pe}) \cup \text{n}(\sigma)) = \emptyset$.

- if $\mu = \tau$. Then $P\sigma \xrightarrow[S_1]{\tau} P'$.

Since $P \sim_{\text{OH}}^{\text{se}} Q$, we have $Q\sigma \xrightarrow[S_2]{\tau} Q'$ and $P' \sim_{\text{OH}}^{\text{se}'} Q'$ where $\text{se}' := \text{se}_B^{(\sigma, \sigma)} +_c(S_1, S_2)$.

Thus $Q\sigma \xrightarrow{\tau} Q'$.

Moreover, since se' is consistent, we clearly have that $\text{se}' \in \langle \text{pe}^\sigma \rangle$.

So $(\text{pe}^\sigma, P', Q') \in \mathcal{R}$.

- if $\mu = a(x)$. Then $P\sigma \xrightarrow[S_1]{a} (x)P'$.

Clearly, we have $x \notin \text{n}(\pi_1(\mathfrak{H}(\text{se}_B^{(\sigma, \sigma)})))$

and $(a, a) \in \mathcal{S}(\mathcal{I}(\mathfrak{H}(\text{se}_B^{(\sigma, \sigma)})))$.

Since $P \sim_{\text{OH}}^{\text{se}} Q$, we have $Q\sigma \xrightarrow[S_2]{a} (y)Q'$ with $P' \sim_{\text{OH}}^{\text{se}'} Q'$ where $\text{se}' := \text{se}_B^{(\sigma, \sigma)} +_i(x, y) +_c(S_1, S_2)$.

Since open hedged bisimilarity is preserved by bijective renamings, we also have that $P' \sim_{\text{OH}}^{\text{se}''} Q'\{x/y\}$ where

$$\text{se}'' := \text{se}_B^{(\sigma, \sigma')} +_i(x, x) +_c(S_1, S_2\{x/y\}).$$

So $\text{se}'' \in \langle \text{pe}^\sigma +_i x \rangle$.

Moreover $Q\sigma \xrightarrow{a(x)} Q'\{x/y\}$.

So $(\text{pe}^\sigma +_i x, P', Q') \in \mathcal{R}$.

- if $\mu = \bar{a}vz$ or $\mu = \bar{a}z$.

Similarly, we simulate the transition thanks to the hypothesis $P \sim_{\text{OH}}^{\text{se}}$ Q and we use the fact that open hedged bisimilarity is preserved by bijective renamings to have exactly the same transitions. Note that since P and Q are pi calculus processes, there is exactly one extruded name on bound output transitions. ■

We conjectured in [50] that a converse result was true, i.e. if $P \sim_K^{\text{pe}} Q$ (in the pi calculus) then there is $\text{se} \in \langle \text{pe} \rangle$ such that $P \sim_{\text{OH}}^{\text{se}} Q$.

Unfortunately, this converse result does not hold because we based our conjecture on a result of [86], which afterwards appeared to be not exactly what was stated in the article.

Roughly speaking, [86] explains that for deciding framed bisimilarity in the spi calculus, it is sufficient to inspect on input clauses all the messages that are not deeper than a certain *critical depth*. For pi calculus processes and induced environments, the critical depth would be 0.

Unfortunately, this result is wrong. For instance consider the expansion law of the pi calculus. It is true that $P \sim_1 Q$ (in the pi calculus) where

$$\begin{aligned} P &:= a(x).(x(z). \mathbf{0} \mid \bar{x}\langle z \rangle. \mathbf{0}) \\ Q &:= a(x).(x(z).\bar{x}\langle z \rangle. \mathbf{0} + \bar{x}\langle z \rangle.x(z). \mathbf{0} + \tau. \mathbf{0}) \end{aligned}$$

However, in spi calculus, these two processes are *not* bisimilar because after the first input transition, depending on the message substituted for the variable x , the process P may be stuck and thus unable to perform a silent transition whereas Q will in any case be able to perform a silent transition.

5.4 Symbolic Characterisation

The purpose of the symbolic characterisation is to bring the definition of open hedged bisimulation closer to an implementation. It relies on a symbolic transition system similar to the one of [39] that permits to characterise more precisely the pairs of respectful substitutions that are to be examined.

For the full spi calculus, where complex messages including keys and channel names pose new challenges, the only other symbolic semantics that we are aware of was proposed by Durante et al. [69]. However, it is rather complicated, mainly since it is tailored to capture trace semantics and thus not easily reusable for working with bisimulation techniques.

5.4.1 Symbolic transition system

The idea behind the symbolic semantics is to record —without checking— conditions needed to derive a transition. Restrictions are still handled by side conditions in derivation rules. Every other constraint is simply collected in *transition constraints* (or in short *constraints*).

Definition 83 (constraint).

A constraint c has the form $(\nu \bar{z}) \Phi$ where Φ is a finite set of guards and \bar{z} is a finite set of (restricted) names that occur in Φ , i.e. $\{\bar{z}\} \subseteq \mathfrak{n}(\Phi)$. When \bar{z} is empty, we simply write Φ instead of $(\nu \emptyset) \Phi$.

The bound names and the free names of a constraint are defined by:

$$\begin{aligned} \text{bn}((\nu \bar{z}) \Phi) &:= \{\bar{z}\} \\ \text{fn}((\nu \bar{z}) \Phi) &:= \mathfrak{n}(\Phi) \setminus \{\bar{z}\} \end{aligned}$$

We extend the notion of α -equivalence to constraints and will identify α -equivalent constraints.

Substitutions σ are applied to constraints as follows:

$$((\nu \bar{z}) \Phi) \sigma := (\nu \bar{z}) (\Phi \sigma) \quad \text{if } \mathfrak{n}(\sigma) \cap \{\bar{z}\} = \emptyset$$

To simplify the presentation of the symbolic semantics, we define several operations on constraints.

Definition 84 (operations on constraints).

If $c_1 = (\nu \bar{z}_1) \Phi_1$ and $c_2 = (\nu \bar{z}_2) \Phi_2$ and $\{\bar{z}_1\} \cap \{\bar{z}_2\} = \emptyset$, we define the conjunction of c_1 and c_2 as follows:

$$c_1 \& c_2 := (\nu \bar{z}_1 \bar{z}_2) (\Phi_1 \cup \Phi_2) \quad \text{if } \{\bar{z}_1\} \cap \text{fn}(c_2) = \{\bar{z}_2\} \cap \text{fn}(c_1) = \emptyset$$

If $c = (v\bar{z})\Phi$ and $x \notin \{\bar{z}\}$, we define the restriction as follows:

$$\begin{aligned} v_+(x, c) &:= (vx\bar{z})\Phi && \text{if } x \in \text{fn}(c) \\ &:= c && \text{otherwise} \end{aligned}$$

Note that due to our conventions, the operation $\&$ is associative and commutative.

Note that we also have $(c_1 \& c_2)\sigma = c_1\sigma \& c_2\sigma$ and if $x \notin \text{n}(\sigma)$, $(v_+(x, c))\sigma = v_+(x, c\sigma)$.

The symbolic semantics relates processes with *symbolic agents*. Symbolic agents are simply a generalisation of agents where concretions are now of the form $C = (v\bar{z})\langle E \rangle P$ where E is an arbitrary expression and \bar{z} is a finite set of names such that $\{\bar{z}\} \subseteq \text{n}(E)$.

In order to deal with symbolic agents, we extend the notion of substitution to finite maps from names to *expressions*. We extend the notion of support, co-support and names of substitutions to this more general notion of substitution. We also extend the manner a substitution is applied to terms (messages, expressions, guards, processes, symbolic agents, ...). Finally, we extend straightforwardly the definition of pseudo-application to deal with symbolic agents and the composition of symbolic agents with processes.

The *abstract evaluation* $\mathbf{e}_a(E)$ of an expression is the symbolic counterpart of concrete evaluation. Intuitively, it can be seen as the reduction of E without checking that encryption and decryption keys correspond and thus it never fails. It is defined in Table 5.1.

Example 17

Consider $E := \text{Dec}_c^s \text{Enc}_c^s \pi_1(a)$ then $\mathbf{e}_a(E) = \pi_1(a)$. *

Concrete evaluation and abstract evaluation coincide on expressions whose concrete evaluation succeeds.

Lemma 75:

Let $E \in E$. If $\mathbf{e}_c(E) = M \in \mathbf{M}$ then $\mathbf{e}_a(E) = \mathbf{e}_c(E)$.

PROOF

By induction on E . ■

However, the converse result is false: it is not true that if $\mathbf{e}_a(E) = M \in \mathbf{M}$ then $\mathbf{e}_c(E) \in \mathbf{M}$. Indeed, think about $E = \text{Dec}_c^s \text{Enc}_b^s a$. Then $\mathbf{e}_a(E) = a \in \mathbf{M}$ whereas $\mathbf{e}_c(E) = \perp$.

Symbolic late transitions take the form of $P \xrightarrow[c]{\mu} A$ where P is a process, μ is a *symbolic action*, c is a constraint and A is a symbolic agent. A

$\mathbf{e}_a(a)$	$:= a$	if $a \in N$
$\mathbf{e}_a(\text{Enc}_F^s E)$	$:= \text{Enc}_{\mathbf{e}_a(F)}^s \mathbf{e}_a(E)$	
$\mathbf{e}_a(\text{Enc}_F^a E)$	$:= \text{Enc}_{\mathbf{e}_a(F)}^a \mathbf{e}_a(E)$	
$\mathbf{e}_a((E \cdot F))$	$:= (\mathbf{e}_a(E) \cdot \mathbf{e}_a(F))$	
$\mathbf{e}_a(\text{op}(E))$	$:= \text{op}(\mathbf{e}_a(E))$	$\text{op} \in \{\text{pub}, \text{priv}, \text{H}\}$
$\mathbf{e}_a(\text{Dec}_F^s E)$	$:= E_1$	if $\mathbf{e}_a(E) = \text{Enc}_{E_2}^s E_1$
	$\text{Dec}_{\mathbf{e}_a(F)}^s \mathbf{e}_a(E)$	otherwise
$\mathbf{e}_a(\text{Dec}_F^a E)$	$:= E_1$	if $\mathbf{e}_a(E) = \text{Enc}_{E_2}^a E_1$
	$\text{Dec}_{\mathbf{e}_a(F)}^a \mathbf{e}_a(E)$	otherwise
$\mathbf{e}_a(\pi_1(E))$	$:= E_1$	if $\mathbf{e}_a(E) = (E_1 \cdot E_2)$
	$\pi_1(\mathbf{e}_a(E))$	otherwise
$\mathbf{e}_a(\pi_2(E))$	$:= E_2$	if $\mathbf{e}_a(E) = (E_1 \cdot E_2)$
	$\pi_2(\mathbf{e}_a(E))$	otherwise

Table 5.1: Abstract evaluation of expressions

symbolic action is the *silent action* τ or a symbolic barb. A *symbolic barb* is either an expression E (representing input) or a *co-expression* \bar{E} (representing output). The names $n(\mu)$ of an action are defined by $n(\tau) = \emptyset$ and $n(E) = n(\bar{E}) = n(E)$.

Definition 85.

The symbolic semantics of the spi calculus is given by the derivation rules of Table 5.2 enriched by the symmetric variants of S-CLOSE-L and S-PAR-L.

Example 18

Consider the process $P := (\nu k) (\pi_1(a)(x).\bar{c}\langle x \rangle. \mathbf{0} \mid \bar{b}\langle \text{Dec}_k^s \text{Enc}_l^s \pi_1(m) \rangle. \mathbf{0})$

The following symbolic transitions can be derived.

$$\begin{aligned}
 P &\xrightarrow[\{\pi_1(a):N\}]{\pi_1(a)} (x)((\nu k) (\bar{c}\langle x \rangle. \mathbf{0} \mid \bar{b}\langle \text{Dec}_k^s \text{Enc}_l^s \pi_1(m) \rangle. \mathbf{0})) \\
 P &\xrightarrow[(\nu k) \{[b:N], [\text{Dec}_k^s \text{Enc}_l^s \pi_1(m):M]\}]{b} \langle \pi_1(m) \rangle ((\nu k) (\pi_1(a)(x).\bar{c}\langle x \rangle. \mathbf{0} \mid \mathbf{0})) \\
 P &\xrightarrow[(\nu k) \{[\pi_1(a):N], [b:N], [\text{Dec}_k^s \text{Enc}_l^s \pi_1(m):M], [\pi_1(a)=b]\}]{\tau} (\nu k) \bar{c}\langle \pi_1(m) \rangle. \mathbf{0} \quad *
 \end{aligned}$$

$$\begin{array}{c}
\text{S-SILENT} \frac{}{\tau.P \xrightarrow[\emptyset]{\tau} P} \qquad \text{S-INPUT} \frac{}{E(x).P \xrightarrow[\{\{E:N\}\}]{\mathbf{e}_a(E)} (x)P} \\
\\
\text{S-OUTPUT} \frac{}{\overline{E}\langle F \rangle.P \xrightarrow[\{\{E:N\},\{F:M\}\}]{\overline{\mathbf{e}_a(E)}} \langle \mathbf{e}_a(F) \rangle P} \\
\\
\text{S-CLOSE-L} \frac{P \xrightarrow[c_1]{E} F \quad Q \xrightarrow[c_2]{\overline{E'}} C}{P|Q \xrightarrow[\{\{E=E'\}\}&c_1&c_2]{\tau} F \bullet C} \\
\\
\text{S-RES} \frac{P \xrightarrow[c]{\mu} A}{(\nu z)P \xrightarrow[\nu_+(z,c)]{\mu} (\nu z)A} \quad z \notin n(\mu) \qquad \text{S-GUARD} \frac{P \xrightarrow[c]{\mu} A}{\phi P \xrightarrow[c \& \{\phi\}]{\mu} A} \\
\\
\text{S-PAR-L} \frac{P \xrightarrow[c]{\mu} A}{P|Q \xrightarrow[c]{\mu} A|Q} \qquad \text{S-SUM-L} \frac{P \xrightarrow[c]{\mu} A}{P+Q \xrightarrow[c]{\mu} A} \\
\\
\text{S-REP-ACT} \frac{P \xrightarrow[c]{\mu} A}{!P \xrightarrow[c]{\mu} A \mid !P} \qquad \text{S-REP-CLOSE} \frac{P \xrightarrow[c_1]{E} F \quad P \xrightarrow[c_2]{\overline{E'}} C}{!P \xrightarrow[\{\{E=E'\}\}&c_1&c_2]{\tau} (F \bullet C) \mid !P} \\
\\
\text{S-ALPHA} \frac{P =_{\alpha} Q \quad c =_{\alpha} c' \quad Q \xrightarrow[c']{\mu} B \quad B =_{\alpha} A}{P \xrightarrow[c]{\mu} A}
\end{array}$$

Table 5.2: The symbolic semantics of the spi calculus

5.4.2 Properties of the symbolic transition system

We now study the link between the symbolic transition system and the transition system with constraints. The basic idea is that the constraints derived by the symbolic transition system characterise precisely the substitutions $\sigma : N \rightarrow M$ such that $P\sigma$ is able to perform a concrete action.

We extend the definition of concrete evaluation to symbolic actions according to Table 5.3.

$$\begin{aligned}
 \mathbf{e}_c(\tau) &:= \tau \\
 \mathbf{e}_c(E) &:= a \quad \text{if } \mathbf{e}_c(E) = a \in N \\
 &:= \perp \quad \text{otherwise} \\
 \mathbf{e}_c(\bar{E}) &:= \bar{a} \quad \text{if } \mathbf{e}_c(E) = a \in N \\
 &:= \perp \quad \text{otherwise}
 \end{aligned}$$

Table 5.3: Evaluation of symbolic actions

We extend the definition of evaluation to constraints according to Table 5.4.

$$\begin{aligned}
 \mathbf{e}((\nu\bar{z}\Phi)) &:= \mathbf{true} \quad \text{if } \forall \phi \in \Phi : \mathbf{e}(\phi) = \mathbf{true} \\
 &:= \mathbf{false} \quad \text{otherwise}
 \end{aligned}$$

Table 5.4: Evaluation of constraints

We extend the definition of $\mathbf{nc}(\cdot)$ to constraints according to Table 5.5.

$$\mathbf{nc}((\nu\bar{z}\Phi)) := \left(\bigcup_{\phi \in \Phi} \mathbf{nc}(\phi) \right) \setminus \{\bar{z}\}$$

Table 5.5: Type constraints of constraints

If $\sigma : N \rightarrow M$ is a substitution and $c = (\nu\bar{z})\Phi$ is a constraint, we say that σ *satisfies* c if $\mathbf{e}(c\sigma) = \mathbf{true}$.

To relate the symbolic transition system with the concrete one, we need to define an auxiliary relation between agents.

Definition 86 ($>_o$ for expressions).

We let $>_o$ be the smallest reflexive and transitive relation on expressions that satisfies:

1. $\pi_1((E_1 . E_2)) >_o E_1$ if $\mathbf{e}_c(E_2) \neq \perp$
2. $\pi_2((E_1 . E_2)) >_o E_2$ if $\mathbf{e}_c(E_1) \neq \perp$
3. $\text{Dec}_{E_2}^s \text{Enc}_{E_2}^s E_1 >_o E_1$ if $\mathbf{e}_c(E_2) \neq \perp$
4. $\text{Dec}_{\text{priv}(E_2)}^a \text{Enc}_{\text{pub}(E_2)}^a E_1 >_o E_1$ if $\mathbf{e}_c(E_2) \neq \perp$
5. $\text{Dec}_{\text{pub}(E_2)}^a \text{Enc}_{\text{priv}(E_2)}^a E_1 >_o E_1$ if $\mathbf{e}_c(E_2) \neq \perp$
6. $\text{Enc}_{E_2}^s E_1 >_o \text{Enc}_{F_2}^s F_1$ if $E_1 >_o F_1$ and $E_2 >_o F_2$
7. $\text{Dec}_{E_2}^s E_1 >_o \text{Dec}_{F_2}^s F_1$ if $E_1 >_o F_1$ and $E_2 >_o F_2$
8. $\text{Enc}_{E_2}^a E_1 >_o \text{Enc}_{F_2}^a F_1$ if $E_1 >_o F_1$ and $E_2 >_o F_2$
9. $\text{Dec}_{E_2}^a E_1 >_o \text{Dec}_{F_2}^a F_1$ if $E_1 >_o F_1$ and $E_2 >_o F_2$
10. $(E_1 . E_2) >_o (F_1 . F_2)$ if $E_1 >_o F_1$ and $E_2 >_o F_2$
11. $\pi_1(E) >_o \pi_1(F)$ if $E >_o F$
12. $\pi_2(E) >_o \pi_2(F)$ if $E >_o F$
13. $\text{op}(E) >_o \text{op}(F)$ if $E >_o F$ and $\text{op} \in \{\text{pub}, \text{priv}, \text{H}\}$

In other words, $>_o$ is the smallest precongruence on expressions that satisfies the axioms 1-5.

Lemma 76:

Let $E \in E$. If $\mathbf{e}_c(E) = M \in M$, then $E >_o M$. □

Lemma 77:

Let $E, F \in E$ and assume that $E >_o F$. Let $M \in M$. Then

$$\mathbf{e}_c(E) = M \iff \mathbf{e}_c(F) = M \quad \square$$

Lemma 78:

Let $E, F \in E$ and $\sigma : N \rightarrow M$ a substitution. Assume that $E >_o F$. Then $E\sigma >_o F\sigma$. □

We extend the relation $>_o$ to guards and processes.

Definition 87 ($>_o$ for guards and processes).

We let $>_o$ be the smallest relation on guards such that

1. $[E:N] >_o [F:N]$ if $E >_o F$
2. $[E_1 = E_2] >_o [F_1 = F_2]$ if $E_1 >_o F_1$ and $E_2 >_o F_2$

We let $>_o$ be the smallest relation on guards such that

1. $\mathbf{0} >_o \mathbf{0}$
2. $E(x).P >_o F(x).Q$ if $E >_o F$ and $P >_o Q$
3. $\overline{E_1}\langle E_2 \rangle.P >_o \overline{F_1}\langle F_2 \rangle.Q$ if $E_1 >_o F_1$, $E_2 >_o F_2$ and $P >_o Q$
4. $\phi P >_o \psi Q$ if $\phi >_o \psi$ and $P >_o Q$
5. $P_1 | P_2 >_o Q_1 | Q_2$ if $P_1 >_o Q_1$ and $P_2 >_o Q_2$
6. $P_1 + P_2 >_o Q_1 + Q_2$ if $P_1 >_o Q_1$ and $P_2 >_o Q_2$
7. $(\nu x)P >_o (\nu x)Q$ if $P >_o Q$
8. $!P >_o !Q$ if $P >_o Q$
9. $P >_o Q$ if $P =_\alpha P' >_o Q' =_\alpha Q$

Corollary 7 (of Lemma 77):

Let $\phi, \psi \in F$ and assume that $\phi >_o \psi$. Then

$$\mathbf{e}(\phi) = \mathbf{true} \iff \mathbf{e}(\psi) = \mathbf{true}$$

Moreover, if $\mathbf{e}(\phi) = \mathbf{e}(\psi) = \mathbf{true}$, then $\mathbf{nc}(\phi) = \mathbf{nc}(\psi)$. ♠

Finally, we extend the relation $>_o$ to agents and symbolic agents. This yields two different relations which differs in the way concretions are related.

Definition 88 ($>_o^e$ and $>_o^=$).

We let $>_o^e$ be the smallest relation on symbolic agents such that

1. $P >_o^e Q$ if $P >_o Q$
2. $(x)P >_o^e (x)Q$ if $P >_o Q$
3. $(\nu \bar{z})\langle E \rangle P >_o^e (\nu \bar{z})\langle M \rangle Q$ if $\mathbf{e}_c(E) = M \in \mathbf{M}$ and $P >_o Q$
4. $A >_o^e B$ if $A =_\alpha A' >_o^e B' =_\alpha B$

We let $>_o^=$ be the smallest relation on agents such that

1. $P >_o^= Q$ if $P >_o Q$

2. $(x)P >_o^= (x)Q$ if $P >_o Q$
3. $(\nu\bar{z})\langle M \rangle P >_o^= (\nu\bar{z})\langle M \rangle Q$ if $P >_o Q$
4. $A >_o^= B$ if $A =_\alpha A' >_o^= B' =_\alpha B$

Note that in the above definition, it is implicit that $\{\bar{z}\} \subseteq n(E)$ and $\{\bar{z}\} \subseteq n(M)$ since well-formed concretions should satisfy this condition.

Processes related by $>_o$ have the same concrete semantics.

Theorem 13:

Let $P, Q \in \mathbf{P}$ and assume that $P >_o Q$.

1. If $P \xrightarrow[S]{\mu} A$ then $Q \xrightarrow[S]{\mu} B$ and $A >_o^= B$
2. If $Q \xrightarrow[S]{\mu} B$ then $P \xrightarrow[S]{\mu} A$ and $A >_o^= B$ ◇

We now study abstract evaluation and how it interacts with substitutions and concrete evaluation.

Lemma 79:

Let $E \in E$ and $\sigma : N \rightarrow M$ a substitution. Then $\mathbf{e}_a(\mathbf{e}_a(E)\sigma) = \mathbf{e}_a(E\sigma)$. □

Hence, abstract evaluation is idempotent.

Corollary 8:

Let $E \in E$. Then $\mathbf{e}_a(\mathbf{e}_a(E)) = \mathbf{e}_a(E)$. ♠

Lemma 80:

Let $E \in E$ and $\sigma : N \rightarrow M$ a substitution. Assume that $\mathbf{e}_c(E\sigma) = M \in M$. Then $\mathbf{e}_c(\mathbf{e}_a(E)\sigma) = M$. □

The symbolic semantics and the concrete semantics are related according to the following theorem.

Theorem 14:

Let $P, Q \in \mathbf{P}$ and $\sigma : N \rightarrow M$ a substitution.

1. If $P \xrightarrow[c]{\mu_s} A$ and $\mathbf{e}(c\sigma) = \mathbf{true}$ then $P\sigma \xrightarrow[\mathbf{nc}(c\sigma)]{\mathbf{e}_c(\mu_s\sigma)} B$ with $A\sigma >_o^e B$
2. If $P\sigma \xrightarrow[S]{\mu} B$ then $P \xrightarrow[c]{\mu_s} A$ with $\mathbf{e}(c\sigma) = \mathbf{true}$, $\mathbf{nc}(c\sigma) = S$, $\mathbf{e}_c(\mu_s\sigma) = \mu$ and $A\sigma >_o^e B$ ◇

Note that in the above theorem, it is implicit that $\mathbf{e}_c(\mu_s\sigma) \neq \perp$.

5.4.3 Symbolic open hedged bisimulation

The previous results suggest to replace in Definition 77 the concrete transitions system by the symbolic transitions system. This yields the notion of *symbolic open hedged bisimulation*.

Definition 89 (symbolic open hedged bisimulation).

A symmetric consistent open hedged relation \mathcal{R} is a *symbolic open hedged bisimulation* if for all $(\mathbf{se}, P, Q) \in \mathcal{R}$, for all σ, ρ and B such that $(\sigma, \rho) \triangleright_B \mathbf{se}$,

1. if $P \xrightarrow{c_1} P'$ and $\mathbf{e}(c_1\sigma) = \mathbf{true}$ then
 there exist Q' and c_2 such that $Q \xrightarrow{c_2} Q'$, $\mathbf{e}(c_2\rho) = \mathbf{true}$
 and $(\mathbf{se}_B^{(\sigma, \rho)} +_c(\mathbf{nc}(c_1\sigma), \mathbf{nc}(c_2\rho)), P'\sigma, Q'\rho) \in \mathcal{R}$
2. if $P \xrightarrow{E} (x)P'$ (with $x \notin \mathfrak{n}(\pi_1(\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)}))))$,
 $\mathbf{e}(c_1\sigma) = \mathbf{true}$ and $(\mathbf{e}_c(E\sigma), b) \in \mathcal{S}(\mathcal{I}(\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)})))$ then
 there exist y, E', Q' and c_2 such that $Q \xrightarrow{E'} (y)Q'$
 (with $y \notin \mathfrak{n}(\pi_2(\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)}))))$, $\mathbf{e}(c_2\rho) = \mathbf{true}$, $\mathbf{e}_c(E'\rho) = b$
 and $(\mathbf{se}_B^{(\sigma, \rho)} +_1(x, y) +_c(\mathbf{nc}(c_1\sigma), \mathbf{nc}(c_2\rho)), P'\sigma, Q'\rho) \in \mathcal{R}$
3. if $P \xrightarrow{E} (v\tilde{c}) \langle F \rangle P'$ (with $\{\tilde{c}\} \cap \mathfrak{n}(\pi_1(\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)}))) = \emptyset$)
 $\mathbf{e}(c_1\sigma) = \mathbf{true}$ and $(\mathbf{e}_c(E\sigma), b) \in \mathcal{S}(\mathcal{I}(\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)})))$ then
 there exist \tilde{d}, E', F', Q' and c_2 such that $Q \xrightarrow{E'} (v\tilde{d}) \langle F' \rangle Q'$
 (with $\{\tilde{d}\} \cap \mathfrak{n}(\pi_2(\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)}))) = \emptyset$), $\mathbf{e}(c_2\rho) = \mathbf{true}$, $\mathbf{e}_c(E'\rho) = b$
 and $(\mathbf{se}_B^{(\sigma, \rho)} +_o(\mathbf{e}_c(F\sigma), \mathbf{e}_c(F'\rho)) +_c(\mathbf{nc}(c_1\sigma), \mathbf{nc}(c_2\rho)), P'\sigma, Q'\rho) \in \mathcal{R}$

Let $\mathbf{se} \in S_H$ and $P, Q \in P$. We say that P and Q are *symbolic open hedged bisimilar under \mathbf{se}* —written $P \sim_{SOH}^{\mathbf{se}} Q$ —if there exists a symbolic open hedged bisimulation \mathcal{R} such that $(\mathbf{se}, P, Q) \in \mathcal{R}$.

As expected, symbolic open hedged bisimilarity and open hedged bisimilarity coincide as stated by the following theorem.

Theorem 15 (Symbolic characterisation):

Let $\mathbf{se} \in S_H$ and $P, Q \in P$. Then

$$P \sim_{OH}^{\mathbf{se}} Q \iff P \sim_{SOH}^{\mathbf{se}} Q$$

PROOF

We prove both implications.

$$\Rightarrow \text{Let } \mathcal{R} = \left\{ (\mathbf{se}, P, Q) \mid \left\{ \begin{array}{l} P' \sim_{\text{OH}}^{\mathbf{se}} Q' \\ P >_o P' \wedge Q >_o Q' \\ \text{fn}(P) \subseteq \mathbf{n}(\pi_1(\mathfrak{H}(\mathbf{se}))) \\ \text{fn}(Q) \subseteq \mathbf{n}(\pi_2(\mathfrak{H}(\mathbf{se}))) \end{array} \right\} \right\}.$$

We show that \mathcal{R} is a symbolic open hedged bisimulation.

Clearly \mathcal{R} is a symmetric consistent open hedged relation.

Let $(\mathbf{se}, P, Q) \in \mathcal{R}$ and σ, ρ and B such that $(\sigma, \rho) \triangleright_B \mathbf{se}$.

There are P_0 and Q_0 such that $P_0 \sim_{\text{OH}}^{\mathbf{se}} Q_0$ and $P >_o P_0$ and $Q >_o Q_0$.

By Corollary 9, since $P >_o P_0$ and $Q >_o Q_0$ we have $(*) P\sigma >_o P_0\sigma$ and $Q\rho >_o Q_0\rho$.

– Assume that $P \xrightarrow[c_1]{\tau} P'$ with $\mathbf{e}(c_1\sigma) = \mathbf{true}$.

By Theorem 14, we have $P\sigma \xrightarrow[\mathbf{nc}(c_1\sigma)]{\tau} P''$ with $P'\sigma >_o^e P''$, i.e. $P'\sigma >_o P''$.

By Theorem 13 and $(*)$, we thus have $P_0\sigma \xrightarrow[\mathbf{nc}(c_1\sigma)]{\tau} P_1$ with $P'' >_o^= P_1$, i.e. $P'' >_o P_1$.

Since $P_0 \sim_{\text{OH}}^{\mathbf{se}} Q_0$, we have $Q_0\rho \xrightarrow[S_2]{\tau} Q_1$ with $P_1 \sim_{\text{OH}}^{\mathbf{se}'} Q_1$ and where $\mathbf{se}' := \mathbf{se}_B^{(\sigma, \rho)} +_c(\mathbf{nc}(c_1\sigma), S_2)$.

So by Theorem 13 and $(*)$, we have $Q\rho \xrightarrow[S_2]{\tau} Q''$ with $Q'' >_o^= Q_1$, i.e. $Q'' >_o Q_1$.

Hence by Theorem 14, $Q \xrightarrow[c_2]{\tau} Q'$ with $\mathbf{e}(c_2\rho) = \mathbf{true}$, $\mathbf{nc}(c_2\rho) = S_2$ and $Q'\rho >_o^e Q''$, i.e. $Q'\rho >_o Q''$.

Thus $P'\sigma >_o P_1$ and $Q'\rho >_o Q_1$.

Clearly $\text{fn}(P'\sigma) \subseteq \mathbf{n}(\pi_1(\mathfrak{H}(\mathbf{se}')))$ and $\text{fn}(Q'\rho) \subseteq \mathbf{n}(\pi_2(\mathfrak{H}(\mathbf{se}')))$.

So $(\mathbf{se}', P'\sigma, Q'\rho) \in \mathcal{R}$.

– Assume that $P \xrightarrow[c_1]{E} (x)P'$ (with $x \notin \mathbf{n}(\pi_1(\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)}))))$, $\mathbf{e}(c_1\sigma) = \mathbf{true}$ and $(\mathbf{e}_c(E\sigma), b) \in \mathcal{S}(\mathcal{I}(\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)})))$.

By Theorem 14, we have $P\sigma \xrightarrow[\mathbf{nc}(c_1\sigma)]{\mathbf{e}_c(E\sigma)} (x)P''$ with $((x)P')\sigma >_o^e (x)P''$, i.e. $P'\sigma >_o P''$.

By Theorem 13 and (*), we thus have $P_0\sigma \xrightarrow[\mathbf{nc}(c_1\sigma)]{\mathbf{e}_c(E\sigma)} (x)P_1$ with $(x)P'' >_{\circ}^{\bar{=}} (x)P_1$, i.e. $P'' >_{\circ} P_1$.

Since $P_0 \sim_{\text{OH}}^{\text{se}} Q_0$, we have $Q_0\rho \xrightarrow[S_2]{b} (y)Q_1$

(with $y \notin \mathbf{n}(\pi_2(\mathfrak{H}(\mathbf{se}_B^{(\sigma,\rho)})))$)

and $P_1 \sim_{\text{OH}}^{\text{se}'} Q_1$ where $\text{se}' := \mathbf{se}_B^{(\sigma,\rho)} +_i(x, y) +_c(\mathbf{nc}(c_1\sigma), S_2)$.

So by Theorem 13 and (*), $Q\rho \xrightarrow[S_2]{b} (y)Q''$ with $(y)Q'' >_{\circ}^{\bar{=}} (y)Q_1$, i.e. $Q'' >_{\circ} Q_1$.

Hence by Theorem 14, $Q \xrightarrow[c_2]{E'} (y)Q'$ with $\mathbf{e}(c_2\rho) = \mathbf{true}$,

$\mathbf{nc}(c_2\rho) = S_2$, $\mathbf{e}_c(E'\rho) = b$ and $((y)Q')\rho >_{\circ}^{\mathbf{e}} (y)Q''$, i.e. $Q'\rho >_{\circ} Q''$.

Thus $P'\sigma >_{\circ} P_1$ and $Q'\rho >_{\circ} Q_1$.

So $(\text{se}', P'\sigma, Q'\rho) \in \mathcal{R}$.

– Assume that $P \xrightarrow[c_1]{\bar{E}} (v\bar{c})\langle F \rangle P'$ (with $\{\bar{c}\} \cap \mathbf{n}(\pi_1(\mathfrak{H}(\mathbf{se}_B^{(\sigma,\rho)}))) = \emptyset$) and $\mathbf{e}(c_1\sigma) = \mathbf{true}$ and $(\mathbf{e}_c(E\sigma), b) \in \mathcal{S}(\mathcal{I}(\mathfrak{H}(\mathbf{se}_B^{(\sigma,\rho)})))$.

By Theorem 14, we have $P\sigma \xrightarrow[\mathbf{nc}(c_1\sigma)]{\overline{\mathbf{e}_c(E\sigma)}} (v\bar{c})\langle M \rangle P''$ with

$((v\bar{c})\langle F \rangle P')\sigma >_{\circ}^{\mathbf{e}} (v\bar{c})\langle M \rangle P''$ i.e. $\mathbf{e}_c(F\sigma) = M$ and $P'\sigma >_{\circ} P''$.

By Theorem 13 and (*), we thus have $P_0\sigma \xrightarrow[\mathbf{nc}(c_1\sigma)]{\overline{\mathbf{e}_c(E\sigma)}} (v\bar{c})\langle M \rangle P_1$ with $(v\bar{c})\langle M \rangle P'' >_{\circ}^{\bar{=}} (v\bar{c})\langle M \rangle P_1$ i.e. $P'' >_{\circ} P_1$.

Since $P_0 \sim_{\text{OH}}^{\text{se}} Q_0$, we have $Q_0\rho \xrightarrow[S_2]{\bar{b}} (v\bar{d})\langle N \rangle Q_1$

(with $\{\bar{d}\} \cap \mathbf{n}(\pi_2(\mathfrak{H}(\mathbf{se}_B^{(\sigma,\rho)}))) = \emptyset$)

and $P_1 \sim_{\text{OH}}^{\text{se}'} Q_1$ where $\text{se}' := \mathbf{se}_B^{(\sigma,\rho)} +_{\circ}(M, N) +_c(\mathbf{nc}(c_1\sigma), S_2)$.

So by Theorem 13 and (*), we have $Q\rho \xrightarrow[S_1]{\bar{b}} (v\bar{d})\langle N \rangle Q''$ with $(v\bar{d})\langle N \rangle Q'' >_{\circ}^{\bar{=}} (v\bar{d})\langle N \rangle Q_1$, i.e. $Q'' >_{\circ} Q_1$.

Hence by Theorem 14, $Q \xrightarrow[c_2]{E'} (v\bar{d})\langle F' \rangle Q'$ with $\mathbf{e}(c_2\rho) = \mathbf{true}$,

$\mathbf{nc}(c_2\rho) = S_2$, $\mathbf{e}_c(\bar{E}'\rho) = \bar{b}$ and $((v\bar{d})\langle F' \rangle Q')\rho >_{\circ}^{\mathbf{e}} (v\bar{d})\langle N \rangle Q''$, i.e. $\mathbf{e}_c(F'\rho) = N$ and $Q'\rho >_{\circ} Q''$.

Thus $P'\sigma >_{\circ} P_1$ and $Q'\rho >_{\circ} Q_1$.

So $(\mathbf{se}', P'\sigma, Q'\rho) \in \mathcal{R}$.

Hence \mathcal{R} is a symbolic open hedged bisimulation.

$$\Leftarrow \text{Let } \mathcal{R} = \left\{ (\mathbf{se}, P, Q) \mid \left\{ \begin{array}{l} P' \sim_{\text{SOH}}^{\mathbf{se}} Q' \\ P' >_o P \wedge Q' >_o Q \end{array} \right\} \right\}.$$

We show that \mathcal{R} is an open hedged bisimulation.

Clearly \mathcal{R} is a symmetric consistent open hedged relation.

Let $(\mathbf{se}, P, Q) \in \mathcal{R}$ and σ, ρ and B such that $(\sigma, \rho) \triangleright_B \mathbf{se}$.

There are P_0 and Q_0 such that $P_0 \sim_{\text{SOH}}^{\mathbf{se}} Q_0$ and $P_0 >_o P$ and $Q_0 >_o Q$.

By Corollary 9, since $P_0 >_o P$ and $Q_0 >_o Q$ we have (*) $P_0\sigma >_o P\sigma$ and $Q_0\rho >_o Q\rho$.

– Assume that $P\sigma \xrightarrow[S_1]{\tau} P'$.

By Theorem 13, we have $P_0\sigma \xrightarrow[S_1]{\tau} P_1$ with $P_1 >_o^= P'$, i.e. $P_1 >_o P'$.

Thus by Theorem 14, we have $P_0 \xrightarrow[c_1]{\tau} P'_0$ with $\mathbf{e}(c_1\sigma) = \mathbf{true}$, $\mathbf{nc}(c_1\sigma) = S_1$ and $P'_0\sigma >_o^e P_1$, i.e. $P'_0\sigma >_o P_1$.

Since $P_0 \sim_{\text{SOH}}^{\mathbf{se}} Q_0$, we have $Q_0 \xrightarrow[c_2]{\tau} Q'_0$ with $\mathbf{e}(c_2\rho) = \mathbf{true}$ and $P'_0\sigma \sim_{\text{SOH}}^{\mathbf{se}'} Q'_0\rho$ where $\mathbf{se}' := \mathbf{se}_B^{(\sigma, \rho)} +_c(\mathbf{nc}(c_1\sigma), \mathbf{nc}(c_2\rho))$.

Thus, by Theorem 14, we have $Q_0\rho \xrightarrow[\mathbf{nc}(c_2\rho)]{\tau} Q_1$ with $Q'_0\rho >_o^e Q_1$, i.e. $Q'_0\rho >_o Q_1$.

So by Theorem 13, $Q\rho \xrightarrow[\mathbf{nc}(c_2\rho)]{\tau} Q'$ with $Q_1 >_o^= Q'$, i.e. $Q_1 >_o Q'$.

So $P'_0\sigma >_o P'$ and $Q'_0\rho >_o Q'$.

So $(\mathbf{se}', P', Q') \in \mathcal{R}$.

– Assume that $P\sigma \xrightarrow[S_1]{a} (x)P'$

(with $x \notin \mathbf{n}(\pi_1(\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)}))))$

and $(a, b) \in \mathcal{S}(\mathcal{I}(\mathfrak{H}(\mathbf{se}_B^{(\sigma, \rho)}))))$.

By Theorem 13, we have $P_0\sigma \xrightarrow[S_1]{a} (x)P_1$ with $(x)P_1 >_o^= (x)P'$, i.e. $P_1 >_o P'$.

Thus by Theorem 14, we have $P_0 \xrightarrow[c_1]{E} (x)P'_0$ with $\mathbf{e}(c_1\sigma) = \mathbf{true}$, $\mathbf{nc}(c_1\sigma) = S_1$, $\mathbf{e}_c(E\sigma) = a$ and $((x)P'_0)\sigma >_o^e (x)P_1$, i.e. $P'_0\sigma >_o P_1$.

Since $P_0 \sim_{\text{SOH}}^{\text{se}} Q_0$, we have $Q_0 \xrightarrow{E'}_{c_2} (y)Q'_0$

(with $y \notin \mathfrak{n}(\pi_2(\mathfrak{H}(\text{se}_B^{(\sigma, \rho)})))$), $\mathbf{e}(c_2\rho) = \mathbf{true}$, $\mathbf{e}_c(E'\rho) = b$

and $P'_0\sigma \sim_{\text{SOH}}^{\text{se}'} Q'_0\rho$ where

$\text{se}' := \text{se}_B^{(\sigma, \rho)} +_i(x, y) +_c(\mathbf{nc}(c_1\sigma), \mathbf{nc}(c_2\rho))$.

Thus, by Theorem 14, $Q_0\rho \xrightarrow[\mathbf{nc}(c_2\rho)]{b} (y)Q_1$ with $((y)Q'_0)\rho >_c^\varepsilon$

$(y)Q_1$, i.e. $Q'_0\rho >_o Q_1$.

So by Theorem 13, $Q\rho \xrightarrow[\mathbf{nc}(c_2\rho)]{b} (y)Q'$ with $(y)Q_1 >_o^\varepsilon (y)Q'$, i.e.

$Q_1 >_o Q'$.

So $P'_0\sigma >_o P'$ and $Q'_0\rho >_o Q'$.

So $(\text{se}', P', Q') \in \mathcal{R}$.

– Assume that $P\sigma \xrightarrow[S_1]{\bar{a}} (v\bar{c}) \langle M \rangle P'$

(with $\{\bar{c}\} \cap \mathfrak{n}(\pi_1(\mathfrak{H}(\text{se}_B^{(\sigma, \rho)}))) = \emptyset$) and

$(a, b) \in \mathcal{S}(\mathcal{I}(\mathfrak{H}(\text{se}_B^{(\sigma, \rho)})))$.

By Theorem 13, we have that $P_0\sigma \xrightarrow[S_1]{\bar{a}} (v\bar{c}) \langle M \rangle P_1$

with $(v\bar{c}) \langle M \rangle P_1 >_o^\varepsilon (v\bar{c}) \langle M \rangle P'$, i.e. $P_1 >_o P'$.

Thus by Theorem 14, $P_0 \xrightarrow{E'}_{c_1} (v\bar{c}) \langle F \rangle P'_0$ with $\mathbf{e}(c_1\sigma) = \mathbf{true}$,

$\mathbf{nc}(c_1\sigma) = S_1$, $\mathbf{e}_c(E\bar{\sigma}) = \bar{a}$ and $((v\bar{c}) \langle F \rangle P'_0)\sigma >_c^\varepsilon (v\bar{c}) \langle M \rangle P_1$,
i.e. $\mathbf{e}_c(F\sigma) = M$ and $P'_0\sigma >_o P_1$.

Since $P_0 \sim_{\text{SOH}}^{\text{se}} Q_0$, we have $Q_0 \xrightarrow{E'}_{c_2} (v\bar{d}) \langle F' \rangle Q'_0$

(with $\{\bar{d}\} \cap \mathfrak{n}(\pi_2(\mathfrak{H}(\text{se}_B^{(\sigma, \rho)}))) = \emptyset$), $\mathbf{e}(c_2\rho) = \mathbf{true}$, $\mathbf{e}_c(E'\rho) = b$

and $P'_0\sigma \sim_{\text{SOH}}^{\text{se}'} Q'_0\rho$

where $\text{se}' := \text{se}_B^{(\sigma, \rho)} +_o(\mathbf{e}_c(F\sigma), \mathbf{e}_c(F'\rho)) +_c(\mathbf{nc}(c_1\sigma), \mathbf{nc}(c_2\rho))$.

Thus, by Theorem 14, we have $Q_0\rho \xrightarrow[\mathbf{nc}(c_2\rho)]{\bar{b}} (v\bar{d}) \langle N \rangle Q_1$ with

$((v\bar{d}) \langle F' \rangle Q'_0)\rho >_c^\varepsilon (v\bar{d}) \langle N \rangle Q_1$, i.e. $\mathbf{e}_c(F'\rho) = N$ and $Q'_0\rho >_o Q_1$.

So by Theorem 13, $Q\rho \xrightarrow[\mathbf{nc}(c_2\rho)]{\bar{b}} (v\bar{d}) \langle N \rangle Q'$ with $(v\bar{d}) \langle N \rangle Q_1 >_o^\varepsilon$

$(v\bar{d}) \langle N \rangle Q'$, i.e. $Q_1 >_o Q'$.

So $P'_0\sigma >_o P'$ and $Q'_0\rho >_o Q'$.

So $(\text{se}', P', Q') \in \mathcal{R}$.

Hence \mathcal{R} is an open hedged bisimulation. ■

5.4.4 Towards mechanisation

Ultimately, we seek mechanisable (efficiently computable) ways to perform equivalence checks. Hüttel [86] showed decidability of bisimilarity checking by giving a “brute-force” decision algorithm for framed bisimulation in a language of only finite processes. However, this algorithm is not practically implementable, generating $\gg 2^{20}$ branches for each input of the Wide-mouthed Frog protocol of [9].

To this matter, Theorem 15 is an important step towards mechanisation of open hedged bisimulation because Definition 89 clarifies which pairs of respectful substitutions have to be considered to enable transitions.

Another crucial point is the infinite quantifications in the definition of environment consistency. We are confident that we can reuse some ideas that were implemented in our prototype tool [38] for checking symbolic bisimilarity of [39]. In this latter case, and as in [33], it turned out to be sufficient to check a finite subset of the environment-respecting substitution pairs: the minimal elements of a refinement preorder. However, the presence of consistency makes for a significant difference in the refinement relation. Since our environments are slightly simpler than those of symbolic bisimilarity (and so is also the definition of consistency), it seems reasonable to think that similar ideas will apply to our setting. Another interesting related work is [57] where a procedure is given for computing a finite set of most general solutions for problems of so called *simultaneous construction*.

In the sequel, we detail a procedure to compute a finite set of most general solutions of transition constraints as derived by the symbolic transition system. We postpone to future work the definition and proof of correctness of a decision procedure for checking open hedged bisimilarity on the fragment of finite spi calculus terms. Note that it was shown in [86] that *finite control spi calculus* is Turing complete.

Solving transition constraints In the following, we assume to have an infinite set of *variables* $\mathcal{V} \subseteq N$ such that $N \setminus \mathcal{V}$ is still infinite. Every substitution considered in this part is assumed to have its support included in \mathcal{V} .

Definition 90 (solution of a constraint).

If c is a transition constraint and σ is a substitution, we say that σ is a *solution* of c if $\mathbf{e}(c\sigma) = \mathbf{true}$.

We note $\mathcal{S}(c)$ the set of solutions of c .

We say that c is *satisfiable* if $\mathcal{S}(c) \neq \emptyset$.

It is clear that if σ is a solution of c then so is the restriction $\sigma|_{\text{fn}(c)}$.

Before addressing the problem of computing $\mathcal{S}(c)$, we consider the problem of solving constraints $c = (\nu \bar{z})\Phi$ where \bar{z} is empty and Φ contains only matching guards.

A *matching problem* is a finite set $S = \{[E_1 = F_1], \dots, [E_n = F_n]\}$ where $E_i, F_i \in \mathbf{E}$.

A substitution σ is a solution of S if for all $1 \leq i \leq n$, $\mathbf{e}([E_i\sigma = F_i\sigma]) = \mathbf{true}$, i.e. for all $1 \leq i \leq n$, $\mathbf{e}_c(E_i\sigma) = \mathbf{e}_c(F_i\sigma) \in \mathbf{M}$.

Note that if for all i , $E_i \in \mathbf{M}$ and $F_i \in \mathbf{M}$, then a matching problem is just a unification problem since in this case, by Lemma 19 and Lemma 22, we have that $\mathbf{e}_c(E_i\sigma) = E_i\sigma \in \mathbf{M}$ and $\mathbf{e}_c(F_i\sigma) = F_i\sigma \in \mathbf{M}$. We thus show how to transform a general matching problem in a matching problem where only messages are involved. Before this, we need some more definitions.

A substitution σ is *more general* than a substitution σ' on $X \subseteq V$, and we write $\sigma \lesssim_X \sigma'$, if there exists a substitution δ such that for all $x \in X$, $x\sigma' = x\sigma\delta$. When $X = V$, we omit to mention X , and so simply write $\sigma \lesssim \sigma'$ and say that σ is more general than σ' . Note that due to our above assumption, we have $\text{supp}(\delta) \subseteq V$.

Observe that if $\sigma \lesssim_X \sigma'$ and $\sigma' \lesssim_{X'} \sigma''$ with $X \subseteq X'$ then $\sigma \lesssim_X \sigma''$. Note also that if $\sigma \lesssim_X \sigma'$ then $\sigma \lesssim_{X'} \sigma'$ for any $X' \subseteq X$. Finally, since only the variables of X are relevant, if σ and σ' agrees on X then $\sigma \lesssim_X \sigma'$.

If c is a matching problem, a complete set of solutions of c is any set \mathcal{S} such that for any solution $\sigma' \in \mathcal{S}(c)$ there exists $\sigma \in \mathcal{S}$ such that $\sigma \lesssim_{n(S)} \sigma'$. The case where \mathcal{S} is finite is particularly interesting.

We are going to define a reduction \implies between matching problems so that solvable normal forms involve only messages. The presence of asymmetric cryptography requires to generalise our setting to finite sets of matching problems.

Indeed, the matching problem $\{[\text{Dec}_y^a x = a]\}$ where $a \in N$ has two more general solutions: $\{\text{Enc}_{\text{pub}(z)}^a / x, \text{priv}(z) / y\}$ and $\{\text{Enc}_{\text{priv}(z)}^a / x, \text{pub}(z) / y\}$, z being a fresh new variable.

A finite set of matching problems is denoted by \mathfrak{S} . We say that σ is a solution of \mathfrak{S} if there exists a matching problem $S \in \mathfrak{S}$ such that σ is a solution of S .

We shall now define \implies . To avoid writing symmetric rules, we define a predicate on expressions which tells if the root node is a deconstructor and will orient matchings accordingly.

Definition 91 (head deconstructor form).

Let $E \in E$. We say that E is in *head deconstructor form*, and write $hd(E)$ if E is of the form $\pi_1(E')$, $\pi_2(E')$, $\text{Dec}_{E_2}^s E_1$ or $\text{Dec}_{E_2}^a E_1$.

Definition 92 (\implies).

The reduction \implies relates a matching problem to a finite set of matching problems. Its definition is given in Table 5.6. In this setting, the \cup symbol should be understood as *disjoint union* when on left side of \implies .

We extend the definition of \implies to finite set of matching problems by defining that $\mathfrak{S} \implies \mathfrak{S}'$ if there exists $S \in \mathfrak{S}$ and \mathfrak{S}'' such that $S \implies \mathfrak{S}''$ and $\mathfrak{S}' := (\mathfrak{S} \setminus S) \cup \mathfrak{S}''$.

Example 19

Consider the following matching problem

$$S := \{[\text{Dec}_k^s \text{Enc}_l^s \pi_1(m) = \text{Dec}_k^s \text{Enc}_l^s \pi_1(m)], [\pi_1(a) = b]\}$$

and assume $S \cap \mathcal{V} = \{a, l, m\}$.

We have

$$\begin{aligned} \{S\} &\implies \{[a = (z_1 . z_2)], [z_1 = b], [\text{Dec}_k^s \text{Enc}_l^s \pi_1(m) = \text{Dec}_k^s \text{Enc}_l^s \pi_1(m)]\} \\ &\implies \left\{ \begin{array}{l} [a = (z_1 . z_2)], [z_1 = b], [\text{Enc}_l^s \pi_1(m) = \text{Enc}_{z_4}^s z_3], \\ [z_4 = k], [z_3 = \text{Dec}_k^s \text{Enc}_l^s \pi_1(m)] \end{array} \right\} \\ &\implies \left\{ \begin{array}{l} [a = (z_1 . z_2)], [z_1 = b], [\pi_1(m) = z_3], [l = z_4], \\ [z_4 = k], [z_3 = \text{Dec}_k^s \text{Enc}_l^s \pi_1(m)] \end{array} \right\} \\ &\implies \left\{ \begin{array}{l} [a = (z_1 . z_2)], [z_1 = b], [\pi_1(m) = z_3], [l = z_4], \\ [z_4 = k], [\text{Dec}_k^s \text{Enc}_l^s \pi_1(m) = z_3] \end{array} \right\} \\ &\implies \left\{ \begin{array}{l} [a = (z_1 . z_2)], [z_1 = b], [m = (z_5 . z_6)], [z_5 = z_3], [l = z_4], \\ [z_4 = k], [\text{Dec}_k^s \text{Enc}_l^s \pi_1(m) = z_3] \end{array} \right\} \\ &\implies \left\{ \begin{array}{l} [a = (z_1 . z_2)], [z_1 = b], [m = (z_5 . z_6)], [z_5 = z_3], [l = z_4], \\ [z_4 = k], [\text{Enc}_k^s \pi_1(m) = \text{Enc}_{z_8}^s z_7], [z_8 = k], [z_7 = z_3] \end{array} \right\} \\ &\implies \left\{ \begin{array}{l} [a = (z_1 . z_2)], [z_1 = b], [m = (z_5 . z_6)], [z_5 = z_3], [l = z_4], \\ [z_4 = k], [\pi_1(m) = z_7], [l = z_8], [z_8 = k], [z_7 = z_3] \end{array} \right\} \\ &\implies \left\{ \begin{array}{l} [a = (z_1 . z_2)], [z_1 = b], \\ [m = (z_5 . z_6)], [z_5 = z_3], [l = z_4], [z_4 = k], \\ [m = (z_9 . z_{10})], [z_9 = z_7], [l = z_8], [z_8 = k], [z_7 = z_3] \end{array} \right\} \end{aligned}$$

$$\begin{array}{l}
1 \frac{\neg hd(E) \quad hd(F)}{\{[E=F]\} \cup S} \Longrightarrow \{[F=E]\} \cup S \quad 2 \frac{\neg hd(E) \quad E \notin \mathcal{V} \quad y \in \mathcal{V}}{\{[E=y]\} \cup S} \Longrightarrow \{[y=E]\} \cup S \\
3 \frac{}{\{[\text{op}(E) = \text{op}(F)]\} \cup S} \Longrightarrow \{[E=F]\} \cup S \quad \text{op} \in \{\text{pub}, \text{priv}, \text{H}\} \\
4 \frac{}{\{[(E_1 \cdot E_2) = (F_1 \cdot F_2)]\} \cup S} \Longrightarrow \{[E_1 = F_1], [E_2 = F_2]\} \cup S \\
5 \frac{}{\{[\text{Enc}_{E_2}^s E_1 = \text{Enc}_{F_2}^s F_1]\} \cup S} \Longrightarrow \{[E_1 = F_1], [E_2 = F_2]\} \cup S \\
6 \frac{}{\{[\text{Enc}_{E_2}^a E_1 = \text{Enc}_{F_2}^a F_1]\} \cup S} \Longrightarrow \{[E_1 = F_1], [E_2 = F_2]\} \cup S \\
7 \frac{\{z_1, z_2\} \subseteq \mathcal{V} \setminus n(E, F, S)}{\{[\pi_1(E) = F]\} \cup S} \Longrightarrow \{[E = (z_1 \cdot z_2)], [z_1 = F]\} \cup S \\
8 \frac{\{z_1, z_2\} \subseteq \mathcal{V} \setminus n(E, F, S)}{\{[\pi_2(E) = F]\} \cup S} \Longrightarrow \{[E = (z_1 \cdot z_2)], [z_2 = F]\} \cup S \\
9 \frac{\{z_1, z_2\} \subseteq \mathcal{V} \setminus n(E_1, E_2, F, S)}{\{[\text{Dec}_{E_2}^s E_1 = F]\} \cup S} \Longrightarrow \{[E_1 = \text{Enc}_{z_2}^s z_1], [z_2 = E_2], [z_1 = F]\} \cup S \\
10 \frac{\{z_1, z_2\} \subseteq \mathcal{V} \setminus n(E_1, E_2, F, S)}{\{[\text{Dec}_{E_2}^a E_1 = F]\} \cup S} \Longrightarrow \left\{ \begin{array}{l} [E_1 = \text{Enc}_{\text{pub}(z_2)}^a z_1], [\text{priv}(z_2) = E_2], [z_1 = F] \\ [E_1 = \text{Enc}_{\text{priv}(z_2)}^a z_1], [\text{pub}(z_2) = E_2], [z_1 = F] \end{array} \right\} \cup S \\
11 \frac{\text{op}(F) \notin \mathbf{M} \quad x \in \mathcal{V} \quad \{z\} \subset \mathcal{V} \setminus n(x, F, S)}{\{[x = \text{op}(F)]\} \cup S} \Longrightarrow \{[x = \text{op}(z)], [z = F]\} \cup S \quad \text{op} \in \{\text{pub}, \text{priv}, \text{H}\} \\
12 \frac{(F_1 \cdot F_2) \notin \mathbf{M} \quad x \in \mathcal{V} \quad \{z_1, z_2\} \subset \mathcal{V} \setminus n(x, F_1, F_2, S)}{\{[x = (F_1 \cdot F_2)]\} \cup S} \Longrightarrow \{[x = (z_1 \cdot z_2)], [z_1 = F_1], [z_2 = F_2]\} \cup S \\
13 \frac{\text{Enc}_{F_2}^s F_1 \notin \mathbf{M} \quad x \in \mathcal{V} \quad \{z_1, z_2\} \subset \mathcal{V} \setminus n(x, F_1, F_2, S)}{\{[x = \text{Enc}_{F_2}^s F_1]\} \cup S} \Longrightarrow \{[x = \text{Enc}_{z_2}^s z_1], [z_1 = F_1], [z_2 = F_2]\} \cup S \\
14 \frac{\text{Enc}_{F_2}^a F_1 \notin \mathbf{M} \quad x \in \mathcal{V} \quad \{z_1, z_2\} \subset \mathcal{V} \setminus n(x, F_1, F_2, S)}{\{[x = \text{Enc}_{F_2}^a F_1]\} \cup S} \Longrightarrow \{[x = \text{Enc}_{z_2}^a z_1], [z_1 = F_1], [z_2 = F_2]\} \cup S
\end{array}$$

Table 5.6: Definition of \Longrightarrow

where $\{z_i \mid 1 \leq i \leq 10\} \subseteq \mathcal{V}$.

Let S' be the last matching problem above. Then S' has a most general unifier, which is $\sigma : z_1 \mapsto b$

$$\begin{aligned} z_5 &\mapsto z_3 \\ z_4 &\mapsto k \\ z_7 &\mapsto z_3 \\ z_8 &\mapsto k \\ z_9 &\mapsto z_7 \\ a &\mapsto (b . z_2) \\ l &\mapsto k \\ m &\mapsto (z_3 . z_6) \\ z_{10} &\mapsto z_6 \end{aligned}$$

Observe that the restriction $\rho = \sigma_{\uparrow n(S)} = \{(b . z_2)/a, k/l, (z_3 . z_6)/m\}$ is a solution of S . *

We show that the reduction process \implies is sound and complete, i.e. that if $\mathfrak{G} \implies \mathfrak{G}'$ then any solution of \mathfrak{G}' is a solution of \mathfrak{G} and conversely that any solution of \mathfrak{G} can be lifted into a solution of \mathfrak{G}' . For the completeness result, note that it is needed to extend the solution of \mathfrak{G} to the fresh new variables introduced in \mathfrak{G}' .

Soundness is a consequence of the following lemma.

Lemma 81:

If $S \implies \mathfrak{G}$ then any solution of \mathfrak{G} is a solution of S .

PROOF

By case distinction on the rule for deriving $S \implies \mathfrak{G}$.

Let σ a solution of \mathfrak{G} .

We just show the result for the following cases:

- If $S = \left\{ [\text{Dec}_{E_2}^a E_1 = F] \right\} \cup S', \{z_1, z_2\} \subseteq \mathcal{V} \setminus n(S)$ and $\mathfrak{G} = \{S_1, S_2\}$ with

$$S_1 = \left\{ [E_1 = \text{Enc}_{\text{pub}(z_2)}^a z_1], [\text{priv}(z_2) = E_2], [z_1 = F] \right\} \cup S'$$

$$S_2 = \left\{ [E_1 = \text{Enc}_{\text{priv}(z_2)}^a z_1], [\text{pub}(z_2) = E_2], [z_1 = F] \right\} \cup S'$$

By symmetry, assume that σ is a solution of S_1 . Then, it is a solution of S' .

Moreover, $\mathbf{e}_c(\text{priv}(z_2\sigma)) = \mathbf{e}_c(E_2\sigma) \in \mathbf{M}$, $\mathbf{e}_c(z_1\sigma) = \mathbf{e}_c(F\sigma) \in \mathbf{M}$ and $\mathbf{e}_c(E_1\sigma) = \mathbf{e}_c(\text{Enc}_{\text{pub}(z_2\sigma)}^a z_1\sigma)$.

Since $z_1\sigma \in \mathbf{M}$ and $z_2\sigma \in \mathbf{M}$, we have $\mathbf{e}_c(E_1\sigma) = \text{Enc}_{\text{pub}(z_2\sigma)}^a z_1\sigma$, $\mathbf{e}_c(E_2\sigma) = \text{priv}(z_2\sigma)$ and $\mathbf{e}_c(F\sigma) = z_1\sigma$.

Thus $\mathbf{e}_c(\text{Dec}_{E_2\sigma}^a E_1\sigma) = z_1\sigma = \mathbf{e}_c(F\sigma)$. Hence σ is a solution of S .

- If $S = \{[x = (F_1 . F_2)]\} \cup S'$, $x \in \mathcal{V}$, $\{z_1, z_2\} \subseteq \mathcal{V} \setminus \mathbf{n}(S)$ and $\mathfrak{S} = \{\{[x = (z_1 . z_2)], [z_1 = F_1], [z_2 = F_2]\} \cup S'\}$.

Since σ is a solution of \mathfrak{S} , we have $\mathbf{e}_c(x\sigma) = \mathbf{e}_c((z_1\sigma . z_2\sigma)) \in \mathbf{M}$, $\mathbf{e}_c(z_1\sigma) = \mathbf{e}_c(F_1\sigma) \in \mathbf{M}$, $\mathbf{e}_c(z_2\sigma) = \mathbf{e}_c(F_2\sigma) \in \mathbf{M}$ and σ is a solution of S' .

Thus $x\sigma = (z_1\sigma . z_2\sigma)$, $z_1\sigma = \mathbf{e}_c(F_1\sigma)$, and $z_2\sigma = \mathbf{e}_c(F_2\sigma)$.

Hence $\mathbf{e}_c((F_1\sigma . F_2\sigma)) = (z_1\sigma . z_2\sigma) = x\sigma = \mathbf{e}_c(x\sigma)$ and σ is a solution of S . \blacksquare

Completeness is a consequence of the following lemma.

Lemma 82:

If $S \implies \mathfrak{S}$ and σ is a solution of S then there exists a solution ρ of \mathfrak{S} that agrees with σ on $\mathbf{n}(S)$.

PROOF

By case distinction on the rule for deriving $S \implies \mathfrak{S}$.

Since we are only interested in the names of S , we may assume that $\text{supp}(\sigma) \subseteq V \cap \mathbf{n}(S)$.

We just show the result for the following cases:

- If $S = \{[\text{Dec}_{E_2}^a E_1 = F]\} \cup S'$, $\{z_1, z_2\} \subseteq \mathcal{V} \setminus \mathbf{n}(S)$ and $\mathfrak{S} = \{S_1, S_2\}$ with

$$S_1 = \{[E_1 = \text{Enc}_{\text{pub}(z_2)}^a z_1], [\text{priv}(z_2) = E_2], [z_1 = F]\} \cup S'$$

$$S_2 = \{[E_1 = \text{Enc}_{\text{priv}(z_2)}^a z_1], [\text{pub}(z_2) = E_2], [z_1 = F]\} \cup S'$$

We have $\{z_1, z_2\} \cap \text{supp}(\sigma) = \emptyset$.

Since σ is a solution of S then σ is a solution of S' and there exists $M, N' \in \mathbf{M}$ such that $\mathbf{e}_c(E_1\sigma) = \text{Enc}_{N'}^a M$, $\mathbf{e}_c(E_2\sigma) = \text{inv}(N')$ and $\mathbf{e}_c(F\sigma) = M$.

There are two cases: either $N' = \text{pub}(N)$ for some message N or $N' = \text{priv}(N)$ for some message N .

By symmetry, assume that $N' = \text{pub}(N)$ for some $N \in \mathbf{M}$.

Then $\mathbf{e}_c(E_1\sigma) = \text{Enc}_{\text{pub}(N)}^a M$, $\mathbf{e}_c(E_2\sigma) = \text{priv}(N)$ and $\mathbf{e}_c(F\sigma) = M$.

Let ρ defined by $x\rho = x\sigma$ if $x \notin \{z_1, z_2\}$, $z_1\rho = M$ and $z_2\rho = N$.

Then clearly ρ is a solution of S_1 which agrees with σ on $\mathbf{n}(S)$.

- If $S = \{[x = (F_1 . F_2)]\} \cup S'$, $x \in \mathcal{V}$, $\{z_1, z_2\} \subseteq \mathcal{V} \setminus \mathbf{n}(S)$ and $\mathfrak{S} = \{[x = (z_1 . z_2)], [z_1 = F_1], [z_2 = F_2]\} \cup S'$.

Since σ is a solution of S , it is also a solution of S' . Moreover $\mathbf{e}_c(x\sigma) = \mathbf{e}_c((F_1\sigma . F_2\sigma)) \in \mathbf{M}$. But $\mathbf{e}_c(x\sigma) = x\sigma$ since $x\sigma \in \mathbf{M}$.

We define ρ by $x\rho = x\sigma$ if $x \notin \{z_1, z_2\}$, $z_1\rho = F_1\sigma$, $z_2\rho = F_2\sigma$.

Clearly ρ is a solution of \mathfrak{S} that agrees with σ on $\mathbf{n}(S)$. ■

Example 20

Back to Example 19, we have that $\{\sigma\}$ is a complete set of solutions of the matching problem

$$S = \{[\text{Dec}_k^s \text{Enc}_l^s \pi_1(m) = \text{Dec}_k^s \text{Enc}_l^s \pi_1(m)], [\pi_1(a) = b]\}$$

where $\sigma = \{(b.z_2)/a, \quad k/l, \quad (z_3.z_6)/m\}$. *

We now show that normal forms of \implies are matching problems that have no solution or that involve only messages (opposed to arbitrary expressions).

Lemma 83:

If S is in normal form then either S involves only messages or S has no solution.

PROOF

Assume that S is in normal form.

If S involves only messages, we are done.

Otherwise, assume that $[E = F] \in S$ where $E \notin \mathbf{M}$ or $F \notin \mathbf{M}$.

We make a case distinction on $hd(E)$ and $hd(F)$.

If $hd(E)$ then depending on E , one of the rules 7,8,9 or 10 can be applied: contradiction.

If $\neg hd(E)$ and $hd(F)$ then rule 1 can be applied: contradiction.

So the remaining case is $\neg hd(E)$ and $\neg hd(F)$.

If $E \in \mathcal{V}$ then $F \notin \mathcal{M}$ and then one of the rules 11,12,13 or 14 can be applied: contradiction.

If $E \notin \mathcal{V}$ and $F \in \mathcal{V}$ then rule 2 can be applied: contradiction.

So assume that $E \notin \mathcal{V}$ and $F \notin \mathcal{V}$.

In $E \in \mathcal{N}$ then $F \notin \mathcal{M}$ so $F \notin \mathcal{N}$ and $[E=F]$ has no solution. Hence S has no solution.

If $F \in \mathcal{N}$ then similarly $[E=F]$ has no solution. Hence S has no solution.

So assume that $E \notin \mathcal{N}$ and $F \notin \mathcal{N}$. Then, since $\neg hd(E)$ and $\neg hd(F)$, the root node of E and the root node of F is a constructor. Moreover, since neither of the rules 3, 4, 5 nor 6 can be applied, the root node of E and the root node of F differs. Clearly, there is no substitution σ such that $\mathbf{e}_c(E\sigma) = \mathbf{e}_c(F\sigma)$ with $\mathbf{e}_c(E\sigma) \in \mathcal{M}$. So $[E=F]$ has no solution. Hence S has no solution. ■

Finally, we show that \implies is well-founded.

Lemma 84:

\implies is well-founded.

PROOF

First, we define the *weight* $weight(E)$ of an expression E such that any name has a weight of 0, any constructor adds 1 to the weight and any deconstructor adds 3 to the weight. We define the weight of a matching problem $S = \{[E_1=F_1], \dots, [E_n=F_n]\}$ to be the finite multiset of the weight of each expressions in S , i.e.

$$weight(S) = \{\{weight(E_i), weight(F_i) \mid 1 \leq i \leq n\}\}$$

If S is a matching problem, we define n_S to be the number of matching in S of the form $[E=F]$ where $\neg hd(E)$ and $(hd(F)$ or $F \in \mathcal{V})$.

The measure $m(S)$ of a matching problem S is the pair $(weight(s), n_S)$.

It is easy to see that if $S \implies \mathfrak{S}$ then for any $S' \in \mathfrak{S}$, we have $m(S') < m(S)$ where $<$ is the lexicographic order on the pairs composed of a finite multiset of integers and an integer, which is well-founded. Indeed, rules 1-2 keep the first projection but make decrease the second one whereas rules 3-14 make the first projection decrease.

So \implies is well-founded. ■

Thus, given a matching problem S , there exists a finite set of matching problems \mathfrak{S} such that $\{S\} \implies^* \mathfrak{S}$ and \mathfrak{S} is in normal form. According to the previous results, any solution of \mathfrak{S} is a solution of S and any solution of S can be extended to be a solution of \mathfrak{S} . Moreover, any matching problem

of \mathfrak{S} that involves more than messages has no solution. The other matching problems, involving only messages, can be solved with a unification algorithm. Each satisfiable problem yields one most general unifier σ . By construction, we know that $\sigma_{\uparrow n(S)}$ is a solution of S and that any solution ρ of S , we have $\sigma \lesssim_{n(S)} \rho$. In other words, we have shown the following theorem.

Theorem 16:

If S is a matching problem, there exists a complete finite set of solutions of S . \diamond

Hence, we have a constructive way to compute a complete finite set of solutions when a transition guard is of the form $c = S$, where S is a matching problem $\{[E_1 = F_1], \dots, [E_n = F_n]\}$.

If c has some restricted names, i.e. $c = (\nu \vec{z}) S$ where S is a matching problem, then we can always α -rename c so that $\{\vec{z}\} \cap V = \emptyset$ (since $N \setminus V$ is infinite). Then, we solve S as previously. This yields a complete finite set of solutions \mathcal{S} of S .

Then it is easy to see that if $\sigma \in \mathcal{S}$ and $n(\sigma) \cap \{\vec{z}\} \neq \emptyset$ then any instance of σ is not a solution of c . Conversely, if ρ is a solution of c , then ρ is a solution of S so there exists $\sigma \in \mathcal{S}$ such that $\sigma \lesssim_{n(S)} \rho$. Since ρ is a solution of c , we have $n(\rho) \cap \{\vec{z}\} = \emptyset$ and necessarily $n(\sigma) \cap \{\vec{z}\} = \emptyset$. Thus $\mathcal{S}' := \{\sigma \in \mathcal{S} \mid n(\sigma) \cap \{\vec{z}\} = \emptyset\}$ is a complete finite set of solutions of c .

Example 21

Consider $c := (\nu k) S$ where S has been defined in Example 19.

By Example 20, we know that $\{\sigma\}$ is a complete set of solutions of S where $\sigma = \{(b \cdot z_2)/a, k/l, (z_3 \cdot z_6)/m\}$.

Since $k \in n(\sigma)$, σ is not a solution of c . Hence c has no solution. $*$

Finally, if c is a general transition constraint, i.e. $c = (\nu \vec{z}) (S \cup \Phi)$ where S is a matching problem and $\Phi = \{[G_1 : N], \dots, [G_n : N]\}$, we have that $\mathcal{S}(c) = \mathcal{S}(c')$ where $c' = (\nu \vec{z}) (S' \cup \Phi)$ where $S' = S \cup \{[G_i : M] \mid 1 \leq i \leq n\}$. Let \mathcal{S} be a finite complete set of solutions of $(\nu \vec{z}) S'$. Then it is easy to see that $\mathcal{S}' = \{\sigma \in \mathcal{S} \mid \mathbf{e}_c(G_i \sigma) \in N\}$ is a finite complete set of solutions of c .

Conclusion

We have achieved our goal to find an open-style definition of bisimulation in the spi calculus. We have shown that our proposal is an extension of K-open bisimulation, which in turn was shown to coincide with Sangiorgi's

open bisimulation. We have also shown that open hedged bisimulation is a sound proof technique for proving late hedged bisimilarity. Finally, we have given a symbolic characterisation of open hedged bisimulation which constitutes a promising step towards mechanisation.

As future work, there are several interesting paths to follow:

1. On the theoretical side, we would like to investigate whether a converse result to Proposition 4 (page 144) can be proven. Even if our conjecture of [50] finally appeared to be false (for the reasons we have explained), we strongly believe that an analogous result holds. Our current idea is to define the spi translation of pi calculus terms by adding a guard $[x:N]$ after each input prefix binding a name x . We then conjecture that if two pi calculus terms are K-open bisimilar then their spi translations are open hedged bisimilar. Having such a result will show that not only open hedged bisimulation is an extension of K-open bisimulation but also that this extension is *conservative*.
2. Still on the theoretical side, another interesting question is, in how far the congruence properties of K-open bisimulation carry over from the pi calculus to the spi calculus. As noted by Boreale and Gorla in [37], a major difficulty for congruence properties in the spi calculus is the case of parallel composition, where a naive formulation is just false. We could likely reuse a number of ideas of [37] for studying the congruence properties of open hedged bisimulation. However, it is yet unclear to us whether the distinction between input variables and freshly created names will equally help us to formulate more refined congruence properties.
3. On the practical side, we intend to formalise a decision procedure for checking open hedged bisimilarity (for finite terms) and prove its correctness. As mentioned, we have good hope to benefit from our experience acquired when working on the *symbolic bisimulation checker* [38]. Since open hedged bisimilarity is sound w.r.t. to late hedged bisimilarity but is probably coarser than symbolic bisimilarity of [39], this would yield an interesting tool for checking behavioural equivalence of finite spi calculus processes.

An interesting work is the recent proposal of a trace based bisimulation for the spi calculus [138]. In this work, Tiu gives an other formulation of an open-style definition for the spi calculus, which is sound w.r.t. to late hedged bisimulation. It is also shown that the underlying notion of

bisimilarity is a congruence on finite spi processes. The major difference between our proposal and Tiu's proposal is the representation of environments. Roughly speaking, Tiu's environments resemble the alternative view we have given of our growing S-environments (which can be directly interpreted as traces, hence the name). To this matter, our representation seems to be more concise, probably very close to an implementation, but at the cost of being maybe less intuitive. Another subtle difference is that Tiu's environments do not take care of the dynamic type of names (our γ 's in the S-environments); we thus believe that our proposal is coarser than Tiu's one.

To overcome the infinite quantification on process inputs of the bisimulation clause, symbolic techniques [79, 35] have been developed. The idea has been exploited to implement bisimulation checking algorithms for the pi calculus. For instance, [142, 42] both implement open bisimulation of Sangiorgi [127] and rely on the symbolic characterisation lemma.

Another mean to study spi calculus, or more generally a pi calculus with algebraic data terms (defined by constructors and destructors), is to reuse directly pi calculus theory and tools by finding a suitable encoding. Following this direction, a fully abstract encoding w.r.t. may-testing equivalence has been proposed in [19, 20]. Despite being very interesting from a theoretical point of view, it is unclear to us whether these encodings could be of practical use, even more since may-testing equivalence is not necessarily the right notion of equivalence to consider when dealing with security properties.

Chapter 6

A Formalisation in `coq`

Theorem provers are becoming mature enough to be used broadly for formalising and validating mathematical theories. The most emblematic example is probably the recent formalisation of the 4 colour theorem in the `coq` proof assistant [75]. Another interesting experiment is the POPLMARK challenge [16] which is a set of benchmarks designed to evaluate the usability of theorem provers and proof assistants for reasoning about the meta-theory of programming languages.

One major contribution of this thesis is certainly the formalisation in the proof assistant `coq` [135] of almost all of the results presented previously. Our own motivation for doing such a formalisation was firstly to validate our theory of bisimulation for the `spi` calculus and secondly to be able to extract [91] a certified bisimulation checker thanks to the Curry-Howard isomorphism. Contrary to the usual approach of theorem proving where the theory is first developed on paper and then is optionally validated within a theorem prover, we have taken a rather opposite approach where we have validated the theory while elaborating it: our feeling is that theorem provers should not be considered as an option but instead should be the commonplace for accompanying fundamental research in the field of computer science. Working this way has been very fruitful because we have been able to identify, understand and fix small and not so small bugs in the theory (definitions or theorems) instantaneously. It is interesting to note that our feeling is that some of these bugs would have probably otherwise been uncaught. However, our approach had the disadvantage to slow down the development of the theory: formalising and formally proving in a proof assistant is sometimes a very time-consuming task. Thus, we have not been able to extract a fully certified bisimulation checker since there remains some theoretical work to do about this.

In this chapter, we give an overview of the `coq` formalisation we have

led. We show the difficulties that arise when doing such a work and explain how we have overcome them. We then enter into the details of several proofs that we have omitted in the previous chapters. The complete archive can be found at <http://lamp.epfl.ch/~sbriaais>.

6.1 The formalisation

6.1.1 Representation of binders

One main technical difficulty when formalising programming languages (such as the (s)pi calculus or $F_{<}$, like in the POPLMARK challenge) is the representation of *binders* and how *bound variables* and α -conversion are handled. This is difficult because today's proof assistants lack of support for treating quotient sets (working up to α -conversion just means working in the quotient set of terms by the equivalence relation $=_\alpha$). We give a brief account of several representations of binders that has been proposed during the last 40 years. These topics are discussed on the wiki of the POPLMARK challenge and in some contributions (e.g. [90]) to the challenge.

In the sequel, we illustrate our explanation with the example of the λ -calculus [21] and its possible representation in the `ocaml` language [137]. Recall that the set Λ of λ -terms are built upon a set of variables in the following way:

$$\begin{array}{l} t, u ::= x \quad \text{variables} \\ \quad | \lambda x.t \quad \text{abstraction} \\ \quad | (t \ u) \quad \text{application} \end{array}$$

λ -terms are naturally represented in `ocaml` by an algebraic data-type defined as follows:

```
type term = Var of variable (* variables *)
          | Abs of abstraction (* abstraction *)
          | App of term * term (* application *)
and type variable = ...
and type abstraction = ...
```

The matter is how to define the types `variable` and `abstraction`.

Nominal representation

This representation is the closest to the usual mathematical practise where bound variables are identified with names and terms are considered up

to α -conversion of bound names. Nominal logic of Pitts [116] is a first-order logic that internalises this approach by offering primitives for renaming, for freshness of names and for name-binding. Support for nominal logic within proof assistant is still marginal but is making significant and promising progress [140].

Support for nominal representation is not provided by `ocaml` but there exist extensions of `ocaml` that implement the idea of nominal logic [130, 59]. Nonetheless, with this approach, we would have

```
(* ... *)
and type variable = string
and type abstraction = string * term
```

For instance the λ -term $t = \lambda x.(x \lambda y.(x (y a)))$ would be represented by:

```
let t = Abs("x", App(Var("x"),
                    Abs("y", App(Var("x"),
                                App(Var("y"),
                                    Var("a"))))))))
```

de Bruijn indices

In this representation, variables are not represented by names but by the so called *de Bruijn indices* [65]. Thus, each variable is encoded by an index (i.e. an integer) that denotes the number of binders that are in scope between that occurrence and its corresponding binder. A variable n is bound if it is in the scope of at least n binders; otherwise it is free. Free variables are thus also indexed.

With this approach, we would have the following definitions:

```
(* ... *)
and type variable = int
and type abstraction = term
```

Assuming that a has index 0, the λ -term $t = \lambda x.(x \lambda y.(x (y a)))$ would be defined by:

```
let t = Abs(App(Var(0),
               Abs(App(Var(1),
                       App(Var(0),
                           Var(2))))))
```

The strength of this representation is that α -equivalent terms have the same de Bruijn notation. The disadvantage of this notation is that the meaning of de Bruijn indices depends on the context; thus many lifting and relocation operators *pollute* the statement of theorems. However, this notation is very popular and is for instance used internally by the `coq` proof assistant to represent terms.

Locally nameless representation

Roughly speaking, this representation is a blending of the nominal representation and the de Bruijn notation. The idea is to represent bound variables by their de Bruijn indices and free variables by names [51].

Thus, we would have the following definitions:

```
(* ... *)
and type variable = B(*ound*) of int
                  | F(*ree*) of string
and type abstraction = term
```

The λ -term $t = \lambda x.(x \lambda y.(x (y a)))$ would be defined by:

```
let t = Abs(App(Var(B(0)),
               Abs(App(Var(B(1)),
                       App(Var(B(0)),
                           Var(F("a"))))))))
```

This representation has the advantage of the de Bruijn notation where α -equivalent terms have the same representation and the advantage of nominal representation where the statement of theorems are close to usual mathematical practise. Compared to the de Bruijn notation, lifting functions are not needed anymore because the meaning of indices is not anymore context dependent. Indeed, a crucial invariant is that representations of terms never contain free de Bruijn indices. Thus, one should take care that the considered terms are always well-formed (i.e. closed), for instance by defining a well-formedness predicate. Compared to the de Bruijn approach, one difficulty is how to handle the crossing of a binder (that appears for example in RES), since open terms are not considered. There exist several solutions to this problem but it is not yet clear which one is the right way to proceed (see final remarks of [90] for instance).

Higher-order abstract syntax

This representation uses the functions provided by the logic (or the programming language) to represent binders. The advantages of this approach are that α -conversion and substitution of bound names come for free because they are handled automatically by the logic. The downside is that depending on the expressiveness of the logic, it can lead to exotic terms. To avoid these, it is possible to define well-formedness predicates [120] but this kind of representation can be considered as a bit difficult to manipulate.

In `ocaml`, this would correspond to:

```
(* ... *)
and type variable = string
and type abstraction = term -> term
```

It is important to understand that `term -> term` represents the type of functions from `term` to `term`. While looking innocent at first, this quantification over terms of type `term` actually quantifies over arbitrary (well-typed) `ocaml` terms because the type `'a -> term` is inhabited.

The λ -term $t = \lambda x.(x \lambda y.(x (y a)))$ would be defined by:

```
let t = Abs(fun x ->
             App(x, Abs(fun y ->
                        App(x, App(y, Var("a"))))))
```

6.1.2 The de Bruijn representation

The choice of the `coq` proof assistant was guided by the wish to extract a certified tool at the end. This choice has led us to consider either the de Bruijn notation or the locally nameless paradigm for formalising the (s)pi calculus. We finally chose the de Bruijn representation because it seemed to us better suited to represent and handle open terms and open-style definitions.

We now describe the general spirit of our formalisation of the spi calculus in `coq`. An inspiring work has been a formalisation of the polyadic pi calculus within an older version of `coq` [82].

Names are identified with de Bruijn indices (i.e. integers) as expected. We have naturally defined several inductive types to represent messages, expressions, guards, processes, and agents. For instance, processes are defined by:

```

Inductive process : Set :=
| Zero : process
| Par : process -> process -> process
| Sum : process -> process -> process
| Bang : process -> process
| New : process -> process
| Tau : process -> process
| Input : expression -> process -> process
| Output : expression -> expression
              -> process -> process
| IfThen : guard -> process -> process.

```

To keep the terms readable, we use a notation close to the mathematical notation instead of using their concrete `coq` values. Moreover, for the examples, we use free variables in the range $\{a, b, \dots, z\}$ and assume that these are numbered starting from 0.

Thus the process $a(x).[Dec_k^s x : M](vl) \bar{b}(l). \mathbf{0}$ is represented by the `coq` process $0\lambda.[Dec_{11}^s 0 : M]v\bar{3}(0). \mathbf{0}$ (see also Table 6.1).

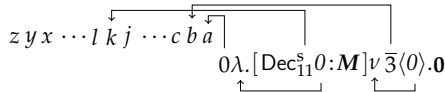


Table 6.1: Representation of $a(x).[Dec_k^s x : M](vl) \bar{b}(l). \mathbf{0}$ using de Bruijn notation

Note that since with de Bruijn notation binders' names are irrelevant, we just indicate that there is a binder in a term either with λ for input prefix or v for restriction. We also use λ for abstractions so that $\langle x \rangle P$ is written $\lambda.P$. Moreover, concretions $(vz_1 \dots z_k) \langle E \rangle P$ are written $v^k \langle E \rangle P$. When $k = 1$, we simply write $v \langle E \rangle P$ and when $k = 0$, we write $\langle E \rangle P$.

It is convenient to define several operations for manipulating de Bruijn indices. We explain and illustrate these operations by showing how to implement several operations on agents. All of these operations take as a parameter the binding depth of the term they are working on: this represents the number of binders in the enclosing context of the term. Going through a λ or a v increases this depth.

1. Clearly, a function is needed to check whether a given de Bruijn index is free in a term or not. The function $\text{mem}_d(i, t)$ returns **true** if the index i is free in term t (at binding depth d) and **false** otherwise.

On names, it is simply defined by:

$$\begin{aligned} \text{mem}_d(i, n) &:= \mathbf{true} && \text{if } n = d + i \\ &:= \mathbf{false} && \text{otherwise} \end{aligned}$$

Then, this operation is lifted to messages, expressions, guards, ... For instance, the definition for processes is given Table 6.2.

$$\begin{aligned} \text{mem}_d(i, \mathbf{0}) &:= \mathbf{false} \\ \text{mem}_d(i, E\lambda.P) &:= \mathbf{true} && \text{if } \text{mem}_d(i, E) = \mathbf{true} \\ &:= \text{mem}_{d+1}(i, P) && \text{otherwise} \\ \text{mem}_d(i, \bar{E}\langle F \rangle.P) &:= \mathbf{true} && \text{if } \text{mem}_d(i, E) = \mathbf{true} \\ &:= \mathbf{true} && \text{if } \text{mem}_d(i, F) = \mathbf{true} \\ &:= \text{mem}_d(i, P) && \text{otherwise} \\ \text{mem}_d(i, \phi P) &:= \mathbf{true} && \text{if } \text{mem}_d(i, \phi) = \mathbf{true} \\ &:= \text{mem}_d(i, P) && \text{otherwise} \\ \text{mem}_d(i, \tau.P) &:= \text{mem}_d(i, P) \\ \text{mem}_d(i, P \mid Q) &:= \mathbf{true} && \text{if } \text{mem}_d(i, P) = \mathbf{true} \\ &:= \text{mem}_d(i, Q) && \text{otherwise} \\ \text{mem}_d(i, P + Q) &:= \mathbf{true} && \text{if } \text{mem}_d(i, P) = \mathbf{true} \\ &:= \text{mem}_d(i, Q) && \text{otherwise} \\ \text{mem}_d(i, !P) &:= \text{mem}_d(i, P) \\ \text{mem}_d(i, \nu P) &:= \text{mem}_{d+1}(i, P) \end{aligned}$$

Table 6.2: Definition of $\text{mem}_d(i, P)$ lifted on processes

2. Recall that the parallel composition of an agent and a process is defined by

$$\begin{aligned} ((x)P) \mid Q &:= (x)(P \mid Q) && \text{if } x \notin \text{fn}(Q) \\ ((\nu \tilde{z}) \langle F \rangle P) \mid Q &:= (\nu \tilde{z}) \langle F \rangle (P \mid Q) && \text{if } \{\tilde{z}\} \cap \text{fn}(Q) = \emptyset \end{aligned}$$

With de Bruijn notation, such side conditions are irrelevant. This kind of conditions is treated in a systematic way since they can always be satisfied by using α -conversion. Thus, an operator $\text{lift}_d(k, t)$ is defined to update the free de Bruijn indices in t to reflect an addition of k binders to the enclosing context of t .

On names, it is simply defined by

$$\begin{aligned} \text{lift}_d(k, n) &:= n && \text{if } n < d \\ &:= n + k && \text{otherwise} \end{aligned}$$

As before, this operation is naturally lifted to other data types.

The parallel composition of an agent and a process is then defined by

$$\begin{aligned} (\lambda.P) | Q &:= \lambda.(P | \text{lift}_0(1, Q)) \\ (\nu^k \langle F \rangle P) | Q &:= \nu^k \langle F \rangle (P | \text{lift}_0(k, Q)) \end{aligned}$$

For instance, consider the abstraction $F := (x)\bar{a}\langle x \rangle. \mathbf{0}$ and the process $Q \text{ Defa}(z).\bar{z}\langle x \rangle. \mathbf{0}$. Their de Bruijn representations are $F = \lambda.\bar{1}\langle 0 \rangle. \mathbf{0}$ and $Q = 0\lambda.\bar{0}\langle 24 \rangle. \mathbf{0}$ and the parallel composition $F | Q$ is equal to $\lambda.(\bar{1}\langle 0 \rangle. \mathbf{0} | 1\lambda.\bar{0}\langle 25 \rangle. \mathbf{0})$ (see also Table 6.3).

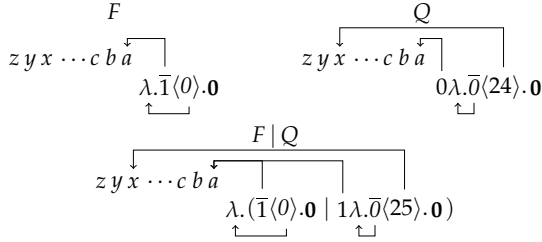


Table 6.3: Parallel composition of an abstraction and a process using de Bruijn representation

3. Recall that the restriction of an agent is defined by

$$\begin{aligned} (\nu y)((x)P) &:= (x)(\nu y)P && \text{if } y \neq x \\ (\nu y)((\nu \bar{z})\langle F \rangle P) &:= (\nu y\bar{z})\langle F \rangle P && \text{if } y \notin \{\bar{z}\} \text{ and } y \in \mathfrak{n}(F) \\ (\nu y)((\nu \bar{z})\langle F \rangle P) &:= (\nu \bar{z})\langle F \rangle (\nu y)P && \text{if } y \notin \{\bar{z}\} \text{ and } y \notin \mathfrak{n}(F) \end{aligned}$$

As before, the side condition $y \neq x$ or $y \notin \{\bar{z}\}$ can be satisfied via α -conversion and is thus trivially handled with de Bruijn notation.

The interesting cases are restriction of an abstraction and restriction of a concretion when $y \notin \mathfrak{n}(F)$. Since the explanation of the former is subsumed by the explanation of the latter, we focus on the restriction of a concretion with $y \notin \mathfrak{n}(F)$. In this case, the name y goes through the binders \bar{z} . So a reordering of de Bruijn indices in P is needed. Moreover, the binding name y is removed from the context of F . Hence indices of F should also be updated (decremented by one). To update terms, we introduce two operations: $\text{swap}_d(k, t)$

that makes a circular permutation of the first k free indices of t and $\text{low}_d(t)$ that decrements by one the free indices of t . This latter operation fails if the index 0 is free in the term (considered at depth d).

Formally, these operations are defined on names by

$$\begin{aligned} \text{swap}_d(i, n) &:= n && \text{if } n < d \\ &:= n + 1 && \text{if } d \leq n < d + k \\ &:= d && \text{if } n = d + k \\ &:= n && \text{if } n > d + k \\ \\ \text{low}_d(n) &:= n && \text{if } n < d \\ &:= n - 1 && \text{if } n > d \end{aligned}$$

Restriction of agents is then defined by

$$\begin{aligned} \mathbf{v}(\lambda.P) &:= \lambda.v \text{swap}_0(1, P) \\ \mathbf{v}(v^k \langle F \rangle P) &:= v^{k+1} \langle F \rangle P && \text{if } \text{mem}_k(0, F) = \mathbf{true} \\ &:= v^k \langle \text{low}_k(F) \rangle v \text{swap}_0(k, P) && \text{otherwise} \end{aligned}$$

4. The pseudo-application is defined by

$$((x)P) \bullet ((v\tilde{z}) \langle F \rangle Q) := (v\tilde{z})(P\{F/x\} \mid Q)$$

with the side condition that $\{\tilde{z}\} \cap \text{fn}(P) = \emptyset$. As before, this requirement can be satisfied via an α -conversion.

To handle the substitution, we define an operation $\text{lsubst}_d(k, E, P)$ that substitutes the index 0 in P by the expression E . The indices of E should be updated according to the binding depth of the place where it is substituted. The other free indices are decremented by one (corresponding to the removal of the bound name that is being substituted) and incremented by k , k being the number of binders in the context of E .

This is defined on names by

$$\begin{aligned} \text{lsubst}_d(k, E, n) &:= n && \text{if } n < d \\ &:= \text{lift}_0(d, E) && \text{if } n = d \\ &:= n - 1 + k && \text{if } n > d \end{aligned}$$

Thus, pseudo-application is defined by

$$(\lambda.P) \bullet (v^k \langle F \rangle Q) := v^k(\text{lsubst}_0(k, F, P) \mid Q)$$

One tedious but necessary task to do when working with de Bruijn representation is to prove several (about 60) facts about the above defined operations (mem, lift, swap, low and lsubst). For instance, swap and lift “commutes” in the following way:

$$\forall d, d', k, k', x : \text{swap}_{d+k+d'}(k', \text{lift}_d(k, x)) = \text{lift}_d(k, \text{swap}_{d+d'}(k', x))$$

When formalising a calculus like the spi calculus, the number of these technical results literally explodes because each of the result should be proved for each data type since there are as many instances of the operators above as the number of data types (i.e. one instance for names, one instance for messages, one instance for expressions, ...) If this problem is not handled carefully, we could have to prove about 60 facts * 6 data types = 360 facts at least. Our approach to overcome this explosion of facts has been to factor the work extensively by finding suitable abstractions.

Hence, we have tried to characterise what it means for a type to use a de Bruijn representation. For convenience reasons, we have split the five basic operations into three groups. The first group of operations is composed of lift, swap and low. The definitions of these operations as well as their properties are gathered into a record type. Thus a value of type `deBruijn_base t` provides these operations for type `t`. The record type is defined as follows:

```
Record deBruijn_base (t:Set) : Set := {
  lift : nat -> nat -> t -> t;
  swapR : nat -> nat -> t -> t;
  lowl : nat -> t -> t;
  (* lift, swapR and lowl properties *)
  ...
  swapR_lift_comm :
    forall (d d' k k':nat)(x:t),
      (swapR (d+k+d') k' (lift d k x)) =
        (lift d k (swapR (d+d') k' x));
  ...
}.
```

Having a record of type `deBruijn_base t` and a compositional map function of type `(nat->t->t)->nat->t'->t'`, it is possible to define a record of type `deBruijn_base t'` therefore providing de Bruijn operations for type `t'`. Eligible map functions are characterised by a few (precisely four) reasonable properties that should be satisfied.

Given a type `t` and a record of type `deBruijn_base t`, we define a new record type that provides the mem operations and its properties:

```

Record deBruijn_fn
  (t:Set)(deBruijn_t:deBruijn_base t) : Set := {
  mem_fn : nat -> nat -> t -> bool;
  (* properties of mem_fn *)
  ...
}

```

Again, it is possible to gently lift this operation to a type t' thanks to a `fold` function that satisfies a few reasonable properties.

Finally, to provide the `lsubst` operation and its properties, we define another record type:

```

Record deBruijn_subst
  (t:Set)(deBruijn_t:deBruijn_base t)
  (to:Set)(deBruijn_to:deBruijn_base to) : Set := {
  low_subst : nat -> nat -> list to -> t -> t;
  (* low_subst properties *)
  ...
}

```

Actually, there is a need to define two kinds of substitutions depending on the return type: either subjects of type t are substituted by objects of type t_0 yielding a term of type t or subjects of type t are substituted by objects of type t_0 yielding a term of type t_0 . The former case is used more often and corresponds to the record type given above. However, the latter case happens when defining the substitution of a name by an expression (yielding back an expression). As for the other operations, given a reasonable `map` function, it is possible to lift this operation to another type t' .

Note that for convenience reasons, we have defined directly a “generalised” substitution, i.e. an operation $lsubst_i(k, es, t)$ that substitutes the n first indices in t by the corresponding expression in the list es , n being the length of this list.

To sum up, the architecture of our formalisation is the following:

- a module `deBruijnNat` implements every basic operation (i.e. `lift`, `swap`, `low`, `mem` and `lsubst`) for the simple type `nat`. The technical results about these operations are also proved in this module.
- a module `deBruijnType` defines the several record types described above and provides several functions to lift `deBruijn` operations from a type to another.

It also defines eligible `map` and `fold` functions for going from type t to type `list t` or from types t_1 and t_2 to type $t_1 * t_2$.

- a module name embeds the operations on nat into operations on names. The type name is defined by

```
Inductive name : Set := Ref : nat -> name.
```

We do this embedding to avoid confusions between natural numbers and de Bruijn indices.

- a module message implements the type of messages. Suitable map and fold functions are given. Then a “lifter” provided by module deBruijnType is used to define de Bruijn operations on messages (and all the technical results that come with them) in terms of de Bruijn operations on names (defined in module name). This task is fast and easy to achieve.
- as for messages, a module for expressions, guards, processes and agents is defined and de Bruijn operations are implemented in the same way.

For instance, the map function on processes is defined by:

```
Fixpoint t_map
  (fe:nat->E.t->E.t)
  (ff:nat->F.t->F.t) (d:nat) (p:t)
  {struct p} : t :=
match p with
| Zero => Zero
| Par p q => Par (t_map fe ff d p)
              (t_map fe ff d q)
| Sum p q => Sum (t_map fe ff d p)
               (t_map fe ff d q)
| Bang p => Bang (t_map fe ff d p)
| New p => New (t_map fe ff (S d) p)
| Tau p => Tau (t_map fe ff d p)
| Input e p => Input (fe d e)
                (t_map fe ff (S d) p)
| Output e f p => Output (fe d e)
                       (fe d f)
                       (t_map fe ff d p)
| IfThen phi p => IfThen (ff d phi)
                       (t_map fe ff d p)
end.
```

Note that it takes two functions as arguments, one for applying operations on expressions (of type $E.t$) and one for applying operations on guards (of type $F.t$). Note also how the meaning of binders is given by incrementing the binding depth d in the case of `New` or `Input`.

This architecture is summarised in Table 6.4. An arrow indicates the use of a “lifter”.

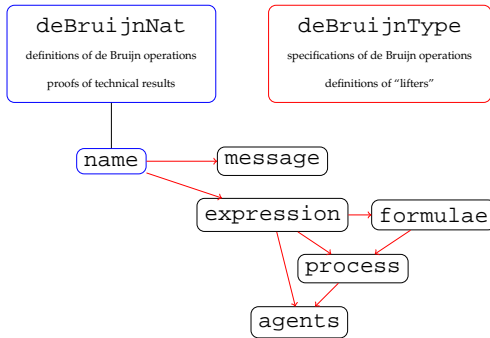


Table 6.4: Architecture of the `coq` formalisation (de Bruijn operations)

6.1.3 Abstracting from the semantics

Now that we have explained the general architecture of the formalisation of terms, we focus our attention on the formalisation of the various labelled semantics.

One may notice that the definitions of the labelled semantics presented previously in Table 3.8, Table 3.9 and Table 5.2 are quite similar. Moreover, they share some properties, for instance see Theorem 4. So to avoid duplication of definitions and proofs, we have used an abstraction to formalise the labelled semantics.

Our approach is to postpone the definition of the labelled semantics until the definition of the semantics of actions. Hence, we assume to have a set of actions \mathcal{A} over which several functions are provided to create and handle them:

- `mkSil` : \mathcal{A} is a silent action.

- $\text{mkInp} : E \rightarrow \mathcal{A} \cup \{\perp\}$ is used for creating input actions. $\text{mkInp}(E)$ creates an action ($\alpha \in \mathcal{A}$) or fails (\perp) depending on the expression E appearing in channel position.
- $\text{mkOutp} : E \times E \rightarrow (\mathcal{A} \times E) \cup \{\perp\}$ is used for creating output actions. $\text{mkOutp}(E, F)$ creates a pair of an action and an expression $((\alpha, M) \in \mathcal{A} \times E)$ or fails (\perp) depending on the expression E appearing in channel position and the expression F appearing in data position (M represents the transmitted data).
- $\text{mkRes} : \mathcal{A} \rightarrow \mathcal{A} \cup \{\perp\}$ is used to restrict a name in an action. $\text{mkRes}(\alpha)$ tries to restrict the first de Bruijn index from α : it may fail (\perp) or give an action ($\beta \in \mathcal{A}$).
- $\text{mkIf} : F \times \mathcal{A} \rightarrow \mathcal{A} \cup \{\perp\}$ is used to guard an action. $\text{mkIf}(\phi, \alpha)$ either gives a guarded action ($\beta \in \mathcal{A}$) or fails.
- $\text{mkInt} : \mathcal{A} \times \mathcal{A} \rightarrow \mathcal{A} \cup \{\perp\}$ is used to make actions interact. Depending on actions α and β , $\text{mkInt}(\alpha, \beta)$ either succeeds or fails.

The labelled semantics is then defined by an inductive predicate as shown by Table 6.5 (as before, we have omitted the symmetric variants of rules CLOSE-L, PAR-L and SUM-L).

It is possible to prove several properties on the labelled semantics of Table 6.5. For instance assume that each action is given a *kind* by the function $\text{kind} : \mathcal{A} \rightarrow \{\mathbf{In}, \mathbf{Out}, \mathbf{Silent}\}$ and assume that the following properties hold:

- $\text{kind}(\text{mkSil}) = \mathbf{Silent}$
- $\forall E \in E, \alpha \in \mathcal{A} : \text{mkInp}(E) = \alpha \implies \text{kind}(\alpha) = \mathbf{In}$
- $\forall E, F \in E, \alpha \in \mathcal{A}, M \in E :$

$$\text{mkOutp}(E, F) = (\alpha, M) \implies \text{kind}(\alpha) = \mathbf{Out}$$
- $\forall \phi \in F, \alpha, \beta \in \mathcal{A} : \text{mkIf}(\phi, \alpha) = \beta \implies \text{kind}(\alpha) = \text{kind}(\beta)$
- $\forall \alpha, \beta \in \mathcal{A} : \text{mkRes}(\alpha) = \beta \implies \text{kind}(\alpha) = \text{kind}(\beta)$
- $\forall \alpha, \beta, \gamma \in \mathcal{A} : \text{mkInt}(\alpha, \beta) = \gamma \implies \text{kind}(\gamma) = \mathbf{Silent}$

Then, one can prove that if $P \xrightarrow{\alpha} A$ and

1. if $\text{kind}(\alpha) = \mathbf{Silent}$ then A is a process Q .

$$\begin{array}{c}
\text{SILENT} \frac{}{\tau.P \xrightarrow{\text{mkSil}} P} \qquad \text{INPUT} \frac{\text{mkInp}(E) = \alpha \in \mathcal{A}}{E\lambda.P \xrightarrow{\alpha} \lambda.P} \\
\\
\text{OUTPUT} \frac{\text{mkOutp}(E, F) = (\alpha, M) \in \mathcal{A} \times E}{\bar{E}\langle F \rangle.P \xrightarrow{\alpha} \langle M \rangle P} \\
\\
\text{CLOSE-L} \frac{P \xrightarrow{\alpha} F \quad Q \xrightarrow{\beta} C \quad \text{mkInt}(\alpha, \beta) = \gamma \in \mathcal{A}}{P | Q \xrightarrow{\gamma} F \bullet C} \\
\\
\text{RES} \frac{P \xrightarrow{\alpha} A \quad \text{mkRes}(\alpha) = \beta \in \mathcal{A}}{\nu P \xrightarrow{\beta} \nu A} \\
\\
\text{IFTHEN} \frac{P \xrightarrow{\alpha} P' \quad \text{mkIf}(\phi, \alpha) = \beta \in \mathcal{A}}{\phi P \xrightarrow{\beta} P'} \qquad \text{PAR-L} \frac{P \xrightarrow{\alpha} A}{P | Q \xrightarrow{\alpha} A | Q} \\
\\
\text{SUM-L} \frac{P \xrightarrow{\alpha} A}{P + Q \xrightarrow{\alpha} A} \qquad \text{REP-ACT} \frac{P \xrightarrow{\alpha} A}{!P \xrightarrow{\alpha} A | !P} \\
\\
\text{REP-CLOSE} \frac{P \xrightarrow{\alpha} F \quad P \xrightarrow{\beta} C \quad \text{mkInt}(\alpha, \beta) = \gamma \in \mathcal{A}}{!P \xrightarrow{\gamma} (F \bullet C) | !P}
\end{array}$$

Table 6.5: Formalisation of the labelled semantics in coq

2. if $\text{kind}(\alpha) = \mathbf{In}$ then A is an abstraction F .
3. if $\text{kind}(\alpha) = \mathbf{Out}$ then A is a concretion C .

Table 6.6 shows the concrete implementation of actions corresponding to Table 3.8. In this setting, actions are defined in the module `ActionSpi` by:

```

Inductive action : Set :=
| Tau : action
| Inp : Name.t -> action
| Outp : Name.t -> action.

(* type for "standard" actions *)
Definition t := action.

      mkSil :=  $\tau$ 
      mkInp( $E$ ) :=  $n$            if  $\mathbf{e}_c(E) = n \in N$ 
                        :=  $\perp$            otherwise
mkOutp( $E, F$ ) :=  $(\bar{n}, M)$    if  $\mathbf{e}_c(E) = n \in N$ 
                        and  $\mathbf{e}_c(F) = M \in M$ 
                        :=  $\perp$            otherwise
      mkIf( $\phi, \alpha$ ) :=  $\alpha$    if  $\mathbf{e}(\phi) = \mathbf{true}$ 
                        :=  $\perp$            otherwise
      mkRes( $\tau$ ) :=  $\tau$ 
      mkRes( $n$ ) :=  $\text{low}_0(n)$    if  $\text{mem}_0(0, n) = \mathbf{false}$ 
                        :=  $\perp$            otherwise
      mkRes( $\bar{n}$ ) :=  $\overline{\text{low}_0(n)}$  if  $\text{mem}_0(0, n) = \mathbf{false}$ 
                        :=  $\perp$            otherwise
      mkInt( $a, \bar{b}$ ) :=  $\tau$      if  $a = b$ 
      mkInt( $\bar{b}, a$ ) :=  $\tau$      if  $a = b$ 
      mkInt( $\alpha, \beta$ ) :=  $\perp$    in all other cases

```

Table 6.6: Implementation of actions for the standard labelled semantics of the spi calculus

The type of actions with type constraints is defined in `ActionSpiNC` by:

```

(* type for "type constraints" *)
Definition constraint := list Name.t.

```

```
(* type for action "with type constraints" *)
Definition t := ActionSpi.action * constraint.
```

The type of symbolic actions is defined in `ActionSpiSymb` by:

```
(* type for constraints *)
Definition constraint := nat * list FormulaeSpi.t.
```

```
Inductive action : Set :=
  | Tau : action
  | Inp : ExpressionSpi.t -> action
  | Outp : ExpressionSpi.t -> action.
```

```
(* type for symbolic action *)
Definition t := action * constraint.
```

The integer in `type constraint` encodes the number of restricted names in the underlying constraint.

6.2 Proof sketches

Hitherto, we have explained the spirit of our formalisation of the spi calculus in `coq`. We now give the proof sketches of some lemmas and theorems stated in the previous chapters. We then quickly illustrate how one can formally reason in `coq` about cryptographic protocols using our framework by showing a perfect encryption equation.

In order to keep proofs readable, we use the mathematical notation but one should keep in mind that all of the following results have been validated in `coq`. We also remind the reader that results about hedges have already been sketched in Chapter 4 but have also been validated in `coq`.

6.2.1 Proof of Theorem 4

Theorem 4 states that:

1. If $P \equiv Q$ and $P \xrightarrow{\mu} A$ then there exists B such that $A \equiv B$ and $Q \xrightarrow{\mu} B$.
2. If $P \equiv Q$ and $P \xrightarrow[S]{\mu} A$ then there exists B such that $A \equiv B$ and $Q \xrightarrow[S]{\mu} B$.

As a test suite for our formalisation, we have thus proved this well-known result in `coq`.

Before sketching the proof, it is interesting to note that it does not hold for the symbolic transitions system. Indeed, consider

$$\begin{aligned} P &:= (\nu x) x(z). \mathbf{0} \\ Q &:= \bar{y}(k). \mathbf{0} \end{aligned}$$

Then $P \mid Q$ cannot perform any internal transition whereas by scope extrusion, we have

$$P \mid Q \equiv (\nu x) (x(z). \mathbf{0} \mid \bar{y}(k). \mathbf{0}) \xrightarrow[\text{scope extrusion}]{\tau} (\nu x) (\mathbf{0} \mid \mathbf{0})$$

This negative result is not particularly annoying because this phenomenon happens only with “false” transitions (i.e. symbolic transitions with unsatisfiable constraints) however it is worth to note it.

As explained previously, we have abstracted the semantics over actions in order to avoid duplication of work. So the theorem proved is rather

Theorem 17:

If the set of actions \mathcal{A} and the functions $\text{mkSil}, \text{mkInp}, \dots$ satisfy some conditions, then if $P \equiv Q$ and $P \xrightarrow{\alpha} A$ then there exist β and B such that $A \equiv B$, $Q \xrightarrow{\beta} B$ and $\alpha = \beta$. \diamond

Note that it is convenient to abstract over the equality of actions because using `coq` equality can sometimes be too strong. That is why we write $\alpha = \beta$ in the above theorem.

Among the required conditions that should be satisfied by actions, the following is needed to prove the case of scope extrusion:

$$\forall \alpha, \alpha', \beta, \gamma'' \in \mathcal{A} : \begin{cases} \text{mkRes}(\alpha) = \alpha' \\ \text{mkInt}(\alpha', \beta) = \gamma'' \end{cases} \implies \begin{cases} \exists \gamma, \gamma' \in \mathcal{A} : \begin{cases} \gamma' = \gamma'' \\ \text{mkInt}(\alpha, \text{lift}_0(1, \beta)) = \gamma \\ \text{mkRes}(\gamma) = \gamma' \end{cases} \end{cases}$$

The proof (of the theorem) proceeds in several steps. Firstly, some technical results about structural congruence and operations on agents have to be proved. For instance, for any abstraction F , concretion C and process P , we have that $(F \bullet C) \mid P \equiv F \bullet (C \mid P)$. Secondly we prove that the theorem holds for each axiom of the structural congruence and thirdly, we show that the result is preserved by process contexts.

We refer the interested reader to the `coq` files to see the full details of the definition of structural congruence and the proof of this theorem.

6.2.2 Proof of Theorem 13

PROOF (OF LEMMA 77)

The goal is to prove that

$$\forall E, F \in \mathbf{E} : E >_o F \implies (\forall M \in \mathbf{M} : \mathbf{e}_c(E) = M \iff \mathbf{e}_c(F) = M)$$

We show the two implications separately, i.e. we show that

1. $\forall E, F \in \mathbf{E} : E >_o F \implies (\forall M \in \mathbf{M} : \mathbf{e}_c(E) = M \implies \mathbf{e}_c(F) = M)$
2. $\forall E, F \in \mathbf{E} : E >_o F \implies (\forall M \in \mathbf{M} : \mathbf{e}_c(F) = M \implies \mathbf{e}_c(E) = M)$

The two proofs proceed by rule induction on $E >_o F$.

To illustrate this, we give the proof of the second implication for the case when $E >_o F$ is deduced from the axiom $\pi_1((E_1.E_2)) >_o E_1$ if $\mathbf{e}_c(E_2) \neq \perp$.

In this case, we have $E = \pi_1((F.G))$ with $\mathbf{e}_c(G) \neq \perp$.

By hypothesis, $\mathbf{e}_c(F) = M$. Since $\mathbf{e}_c(G) \neq \perp$, there exists $N \in \mathbf{M}$ such that $\mathbf{e}_c(G) = N$.

We have thus $\mathbf{e}_c((F.G)) = (M.N)$.

So $\mathbf{e}_c(E) = M\hat{A}$. ■

It is possible to generalise Lemma 22.

Lemma 85:

Let $E \in \mathbf{E}$ and $F_1, \dots, F_n \in \mathbf{E}$ such that for all $1 \leq i \leq n$, $\mathbf{e}_c(F_i) = N_i \neq \perp$. If $\mathbf{e}_c(E) = M \in \mathbf{M}$ then $\mathbf{e}_c(E [F_1/x_1, \dots, F_n/x_n]) = M [N_1/x_1, \dots, N_n/x_n]$.

PROOF

By induction on E . ■

Likewise, it is possible to generalise Lemma 78.

Lemma 86:

1. Let $E, E' \in \mathbf{E}$ such that $E >_o E'$. Let $F_1, \dots, F_n \in \mathbf{E}$ such that for all $1 \leq i \leq n$, $\mathbf{e}_c(F_i) \neq \perp$. Then $E [F_1/x_1, \dots, F_n/x_n] >_o E' [F_1/x_1, \dots, F_n/x_n]$.
2. Let $E \in \mathbf{E}$. Let $F_1, \dots, F_n \in \mathbf{E}$ and $G_1, \dots, G_n \in \mathbf{E}$ such that for all $1 \leq i \leq n$, $F_i >_o G_i$. Then $E [F_1/x_1, \dots, F_n/x_n] >_o E [G_1/x_1, \dots, G_n/x_n]$.
3. Let $E, E' \in \mathbf{E}$ such that $E >_o E'$. Let $F_1, \dots, F_n \in \mathbf{E}$ and $G_1, \dots, G_n \in \mathbf{E}$ such that for all $1 \leq i \leq n$, $F_i >_o G_i$ and $\mathbf{e}_c(F_i) \neq \perp$. Then $E [F_1/x_1, \dots, F_n/x_n] >_o E' [G_1/x_1, \dots, G_n/x_n]$.

PROOF

1. By induction on $E >_o E'$ and thanks to Lemma 85.
2. By induction on E .
3. A simple corollary of the two previous results. ■

Clearly, Lemma 86 can be generalised to the case of guards and processes.

Corollary 9:

1. Let $\phi, \phi' \in F$ such that $\phi >_o \phi'$. Let $F_1, \dots, F_n \in E$ and $G_1, \dots, G_n \in E$ such that for all $1 \leq i \leq n$, $F_i >_o G_i$ and $\mathbf{e}_c(F_i) \neq \perp$. Then $\phi [^{F_1/x_1, \dots, F_n/x_n}] >_o \phi' [^{G_1/x_1, \dots, G_n/x_n}]$.
2. Let $P, P' \in P$ such that $P >_o P'$. Let $F_1, \dots, F_n \in E$ and $G_1, \dots, G_n \in E$ such that for all $1 \leq i \leq n$, $F_i >_o G_i$ and $\mathbf{e}_c(F_i) \neq \perp$. Then $P [^{F_1/x_1, \dots, F_n/x_n}] >_o P' [^{G_1/x_1, \dots, G_n/x_n}]$. ♠

Lemma 87:

1. Let A, B two agents such that $A >_o^= B$. Then $(\mathbf{v}x) A >_o^= (\mathbf{v}x) B$.
2. Let A, B two agents such that $A >_o^= B$ and P, Q two processes such that $P >_o Q$. Then $A | P >_o^= B | Q$.
3. Let F_1, F_2 two abstractions such that $F_1 >_o^= F_2$ and C_1, C_2 two concretions such that $C_1 >_o^= C_2$. Then $F_1 \bullet C_1 >_o F_2 \bullet C_2$.

PROOF

1. By induction on $A >_o^= B$.
2. By induction on $A >_o^= B$.
3. We have $F_1 =_\alpha (x)P_1, F_2 =_\alpha (x)P_2$ with $P_1 >_o P_2$.

Similarly, we have $C_1 =_\alpha (v\bar{z}) \langle M \rangle Q_1$ and $C_2 =_\alpha (v\bar{z}) \langle M \rangle Q_2$ with $\{\bar{z}\} \cap \text{fn}(P_i) = \emptyset$ for $i \in \{1, 2\}$.

By definition, we have

$$\begin{aligned} F_1 \bullet C_1 &= (v\bar{z}) (P_1 \{^M/x\} | Q_1) \\ F_2 \bullet C_2 &= (v\bar{z}) (P_2 \{^M/x\} | Q_2) \end{aligned}$$

So to show the result, it is sufficient to show that $P_1 \{^M/x\} >_o P_2 \{^M/x\}$ which is given by Corollary 9. ■

PROOF (OF THEOREM 13)

The goal is to prove that:

$$\forall P, Q \in \mathcal{P} : P >_o Q \implies$$

$$1. \forall S, A : P \xrightarrow[S]{\mu} A \implies \exists B : A >_o^= B \wedge Q \xrightarrow[S]{\mu} B$$

$$2. \forall S, B : Q \xrightarrow[S]{\mu} B \implies \exists A : A >_o^= B \wedge P \xrightarrow[S]{\mu} A$$

1. By rule induction on $P \xrightarrow[S]{\mu} A$. We give the proof for the following cases:

NC-INPUT Assume that $P = E(x).P' \xrightarrow[\{a\}]{a} (x)P'$ where $\mathbf{e}_c(E) = a \in N$.

Since $P >_o Q$, we have $Q = {}_\alpha F(x).Q'$ with $E >_o F$ and $P' >_o Q'$.
By Lemma 77, we have $\mathbf{e}_c(F) = a$.

Thus, by NC-ALPHA and NC-INPUT, $Q = {}_\alpha F(x).Q' \xrightarrow[\{a\}]{a} (x)Q'$
and $(x)P' >_o^= (x)Q'$.

NC-OUTPUT Assume that $P = \overline{E}_1 \langle E_2 \rangle . P' \xrightarrow[\{a\}]{\bar{a}} \langle M \rangle P'$ with $\mathbf{e}_c(E_1) = a \in N$
and $\mathbf{e}_c(E_2) = M \in M$.

Since $P >_o Q$, we have $Q = \overline{F}_1 \langle F_2 \rangle . Q'$ with $E_1 >_o F_1$, $E_2 >_o F_2$
and $P' >_o Q'$.

By Lemma 77, we have $\mathbf{e}_c(F_1) = a$ and $\mathbf{e}_c(F_2) = M$.

Thus, by NC-OUTPUT, $Q = \overline{F}_1 \langle F_2 \rangle . Q' \xrightarrow[\{a\}]{\bar{a}} \langle M \rangle Q'$ and we have
 $\langle M \rangle P' >_o^= \langle M \rangle Q'$.

NC-GUARD Assume that $P = \phi P_1$ with $P_1 \xrightarrow[S]{\mu} A$, $\mathbf{e}(\phi) = \mathbf{true}$ and $P =$

$$\phi P_1 \xrightarrow[S \cup \mathbf{nc}(\phi)]{\mu} P'.$$

Since $P >_o Q$, we have $Q = \psi Q_1$ with $\phi >_o \psi$ and $P_1 >_o Q_1$.

By induction, $Q_1 \xrightarrow[S]{\mu} B$ with $A >_o^= B$.

By Corollary 7, we have $\mathbf{e}(\psi) = \mathbf{true}$.

Thus, by NC-GUARD, we have $Q = \psi Q_1 \xrightarrow[S \cup \mathbf{nc}(\psi)]{\mu} B$.

Again by Corollary 7, we have $\mathbf{nc}(\psi) = \mathbf{nc}(\phi)$ so $Q \xrightarrow[S \cup \mathbf{nc}(\phi)]{\mu} B$
and $A >_o^= B$.

NC-CLOSE-L Assume that $P = P_1 | P_2$ with $P_1 \xrightarrow{a}_S F_1$, $P_2 \xrightarrow{\bar{a}}_{S'} C_1$ and $P \xrightarrow{\tau}_{SUS'} A = F_1 \bullet C_1$.
 Since $P >_o Q$, we have $Q = Q_1 | Q_2$ with $P_1 >_o Q_1$ and $P_2 >_o Q_2$.
 By induction, we have $Q_1 \xrightarrow{a}_S F_2$ and $Q_2 \xrightarrow{\bar{a}}_{S'} C_2$ with $F_1 >_o F_2$ and $C_1 >_o C_2$.
 Thus by NC-CLOSE-L $Q_1 | Q_2 \xrightarrow{\tau}_{SUS'} F_2 \bullet C_2 = B$.
 By Lemma 87, we have $A >_o B$, i.e. $A >_o B$.

2. By rule induction on $Q \xrightarrow{\mu}_S B$. We give the proof for the following cases:

NC-INPUT Assume that $Q = F(x).Q' \xrightarrow{a}_{\{a\}} (x)Q'$ with $\mathbf{e}_c(F) = a \in N$.

Since $P >_o Q$, we have $P =_\alpha E(x).P'$ with $E >_o F$ and $P' >_o Q'$.
 By Lemma 77, we have $\mathbf{e}_c(E) = a$.

So, by NC-ALPHA and NC-INPUT, we have $P =_\alpha E(x).P' \xrightarrow{a}_{\{a\}} (x)P'$ and $(x)P' >_o (x)Q'$.

NC-CLOSE-L Assume that $Q = Q_1 | Q_2$ with $Q_1 \xrightarrow{a}_S F_2$, $Q_2 \xrightarrow{\bar{a}}_{S'} C_2$ and $Q \xrightarrow{\tau}_{SUS'} B = F_2 \bullet C_2$.

Since $P >_o Q$, we have $P = P_1 | P_2$ with $P_1 >_o Q_1$ and $P_2 >_o Q_2$.

By induction, we have $P_1 \xrightarrow{a}_S F_1$ and $P_2 \xrightarrow{\bar{a}}_{C_1}$ with $F_1 >_o F_2$ and $C_1 >_o C_2$.

So by NC-CLOSE-L, we have $P = P_1 | P_2 \xrightarrow{\tau}_{SUS'} A = F_1 \bullet C_1$.

Moreover, by Lemma 87, we have $A >_o B$ so $A >_o B$.

NC-RES Assume that $Q = (\nu z)Q'$ with $Q' \xrightarrow{\mu}_S B'$ with $z \notin n(\mu)$ and

$Q \xrightarrow{\mu}_{S \setminus \{z\}} B = (\nu z)B'$.

Since $P >_o Q$, we have $P =_\alpha P'$ with $P' >_o Q'$.

By induction we thus have $P' \xrightarrow{\mu}_S A'$ with $A' >_o B'$.

Since $z \notin n(\mu)$, we have by NC-RES and NC-ALPHA, $P =_\alpha (\nu z)P' \xrightarrow{\mu}_{S \setminus \{z\}} A = (\nu z)A'$.

Moreover, by Lemma 87, we have $(\nu z) A' >_o^\circ (\nu z) B'$ i.e. $A >_o^\circ B$. \blacksquare

6.2.3 Proof of Theorem 14

PROOF (OF LEMMA 79)

The goal is to prove that

$$\forall E \in E, \forall \sigma : N \rightarrow M : \mathbf{e}_a(\mathbf{e}_a(E)\sigma) = \mathbf{e}_a(E\sigma)$$

The proof proceeds by induction on E . We give the proof for the inductive case $E = \pi_1(F)$.

Assume $E = \pi_1(F)$ and the result holds for F .

By induction, we have (*) $\mathbf{e}_a(\mathbf{e}_a(F)\sigma) = \mathbf{e}_a(F\sigma)$.

We have two cases:

1. if $\mathbf{e}_a(F) = (F_1 \cdot F_2)$ for some F_1, F_2 .

Then $\mathbf{e}_a(E) = F_1$.

So $\mathbf{e}_a(\mathbf{e}_a(E)\sigma) = \mathbf{e}_a(F_1\sigma)$.

Rewriting (*) gives $\mathbf{e}_a((F_1 \cdot F_2)\sigma) = \mathbf{e}_a(F\sigma)$, i.e. $\mathbf{e}_a((F_1\sigma \cdot F_2\sigma)) = \mathbf{e}_a(F\sigma)$.

By definition, $\mathbf{e}_a((F_1\sigma \cdot F_2\sigma)) = (\mathbf{e}_a(F_1\sigma) \cdot \mathbf{e}_a(F_2\sigma))$.

So $\mathbf{e}_a(F\sigma) = (\mathbf{e}_a(F_1\sigma) \cdot \mathbf{e}_a(F_2\sigma))$.

So $\mathbf{e}_a(E\sigma) = \mathbf{e}_a(\pi_1(F)\sigma) = \mathbf{e}_a(\pi_1(F\sigma)) = \mathbf{e}_a(F_1\sigma)$ by definition.

Hence $\mathbf{e}_a(\mathbf{e}_a(E)\sigma) = \mathbf{e}_a(E\sigma)$.

2. otherwise

Then by definition, $\mathbf{e}_a(E) = \pi_1(\mathbf{e}_a(F))$.

Thus, $\mathbf{e}_a(E)\sigma = \pi_1(\mathbf{e}_a(F)\sigma)$.

So $\mathbf{e}_a(\mathbf{e}_a(E)\sigma) = \mathbf{e}_a(\pi_1(\mathbf{e}_a(F)\sigma))$.

We have then two subcases:

- (a) if $\mathbf{e}_a(\mathbf{e}_a(F)\sigma) = (F_1 \cdot F_2)$

By (*), we have then $\mathbf{e}_a(F\sigma) = (F_1 \cdot F_2)$.

By definition, we have $\mathbf{e}_a(\mathbf{e}_a(E)\sigma) = F_1$.

And by definition, we have $\mathbf{e}_a(\pi_1(F\sigma)) = F_1$.

But $\mathbf{e}_a(\pi_1(F\sigma)) = \mathbf{e}_a(\pi_1(F)\sigma) = \mathbf{e}_a(E\sigma)$.

So $\mathbf{e}_a(\mathbf{e}_a(E)\sigma) = \mathbf{e}_a(E\sigma)$.

(b) otherwise

So by definition, we have $\mathbf{e}_a(\mathbf{e}_a(E)\sigma) = \pi_1(\mathbf{e}_a(\mathbf{e}_a(F)\sigma))$ which is equal to $\pi_1(\mathbf{e}_a(F\sigma))$ by (*).

By (*), we know that $\mathbf{e}_a(F\sigma)$ is not a pair so $\mathbf{e}_a(\pi_1(F\sigma)) = \pi_1(\mathbf{e}_a(F\sigma))$.

But $\mathbf{e}_a(\pi_1(F\sigma)) = \mathbf{e}_a(E\sigma)$.

So $\mathbf{e}_a(\mathbf{e}_a(E)\sigma) = \mathbf{e}_a(E\sigma)$. ■

PROOF (OF LEMMA 80)

The goal is to prove

$$\forall E \in \mathbf{E}, \forall \sigma : \mathbf{N} \rightarrow \mathbf{M} : \mathbf{e}_c(E\sigma) = M \in \mathbf{M} \implies \mathbf{e}_c(\mathbf{e}_a(E)\sigma) = M$$

The proof proceeds by induction on E . We give the proof for the inductive case $E = \pi_1(F)$.

Assume $E = \pi_1(F)$ and the result holds for F

We have $\mathbf{e}_c(E\sigma) = \mathbf{e}_c(\pi_1(F\sigma)) = M \in \mathbf{M}$.

Necessarily, $\mathbf{e}_c(F\sigma) = (M.N)$ for some $N \in \mathbf{M}$.

By induction, we have thus $\mathbf{e}_c(\mathbf{e}_a(F)\sigma) = (M.N)$.

We have two cases:

1. if $\mathbf{e}_a(F) = (F_1.F_2)$

Then $\mathbf{e}_a(E) = F_1$.

We have $\mathbf{e}_c(\mathbf{e}_a(F)\sigma) = \mathbf{e}_c((F_1\sigma.F_2\sigma)) = (M.N)$.

So necessarily, $\mathbf{e}_c(F_1\sigma) = M$ and $\mathbf{e}_c(F_2\sigma) = N$.

Hence the result, since $\mathbf{e}_c(E\sigma) = M = \mathbf{e}_c(F_1\sigma) = \mathbf{e}_c(\mathbf{e}_a(E)\sigma)$.

2. otherwise

We have $\mathbf{e}_a(E) = \pi_1(\mathbf{e}_a(F))$.

So $\mathbf{e}_c(\mathbf{e}_a(E)\sigma) = \mathbf{e}_c(\pi_1(\mathbf{e}_a(F)\sigma))$.

Since $\mathbf{e}_c(\mathbf{e}_a(F)\sigma) = (M.N)$, we have $\mathbf{e}_c(\pi_1(\mathbf{e}_a(F)\sigma)) = M$ by definition.

Hence $\mathbf{e}_c(\mathbf{e}_a(E)\sigma) = \mathbf{e}_c(E\sigma)$. ■

PROOF (OF LEMMA 76)

The goal is to prove that

$$\forall E \in \mathbf{E} : \mathbf{e}_c(E) = M \in \mathbf{M} \implies E >_{\circ} M$$

The proof proceeds by a simple induction on E .

For instance, we give the proof for the inductive case $E = \pi_1(F)$.

Assume that $E = \pi_1(F)$ and the result holds for F .

Since $\mathbf{e}_c(E) = M \in \mathbf{M}$, necessarily $\mathbf{e}_c(F) = (M.N) \in \mathbf{M}$ for some N .

By induction, we have $F >_o (M.N)$.

Thus, by definition of $>_o$, we have $\pi_1(F) >_o \pi_1((M.N))$, i.e. $E >_o \pi_1((M.N))$.

Moreover, by definition of $>_o$, we have $\pi_1((M.N)) >_o M$, because $\mathbf{e}_c(N) \neq \perp$ by Lemma 19 since $N \in \mathbf{M}$.

So, by transitivity of $>_o$, we conclude that $E >_o M$. \blacksquare

Before proving Theorem 14, we need results analogous to Lemma 87 involving $>_o^e$. The case for the restriction is technically difficult and requires some auxiliary definitions.

Definition 93.

For $x \in N$, we define the inductive predicate $x \triangleleft E$ on expressions as shown in Table 6.7.

$x \triangleleft x$	$\frac{x \triangleleft E_1}{x \triangleleft (E_1 . E_2)}$	$\frac{x \triangleleft E_2}{x \triangleleft (E_1 . E_2)}$	$\frac{x \triangleleft E_1}{x \triangleleft \text{Enc}_{E_2}^s E_1}$	$\frac{x \triangleleft E_2}{x \triangleleft \text{Enc}_{E_2}^s E_1}$
$\frac{x \triangleleft E_1}{x \triangleleft \text{Enc}_{E_2}^a E_1}$	$\frac{x \triangleleft E_2}{x \triangleleft \text{Enc}_{E_2}^a E_1}$	$\frac{x \triangleleft E}{x \triangleleft \text{op}(E)} \text{ op} \in \{\text{pub}, \text{priv}, \text{H}\}$		

Table 6.7: Definition of $x \triangleleft E$

Intuitively, $x \triangleleft E$ is a stronger condition than $x \in n(E)$ because it requires the name x to appear in only certain subexpressions of E . It will help us to better control the case for restriction when relating the symbolic LTS to the concrete one.

Lemma 88:

1. If $E \in \mathbf{E}$ then $x \triangleleft E \implies x \in n(E)$.
2. If $M \in \mathbf{M}$ then $x \in n(M) \iff x \triangleleft M$.
3. If $E \in \mathbf{E}$ and $\sigma : N \rightarrow \mathbf{M}$ is a substitution such that $x \notin n(\sigma)$, then if $x \triangleleft E$ then $x \triangleleft E\sigma$.

PROOF

1. By induction on $x \triangleleft E$.

2. By induction on $M \in \mathbf{M}$.

3. By induction on $x \triangleleft E$. ■

Lemma 89:

Let $E \in \mathbf{E}$ and $\sigma : \mathbf{N} \rightarrow \mathbf{M}$ a substitution such that $x \notin \mathfrak{n}(\sigma)$. If $x \triangleleft E$ and $\mathbf{e}_c(E\sigma) = M \in \mathbf{M}$ then $x \in \mathfrak{n}(M)$.

PROOF

By induction on $x \triangleleft E$. ■

Definition 94.

We define a measure $\sharp(E)$ on expressions as shown in Table 6.8.

$\sharp(a)$:= 0	if $a \in \mathbf{N}$
$\sharp((E_1 . E_2))$:= $\sharp(E_1) + \sharp(E_2)$	
$\sharp(\text{Enc}_{E_2}^s E_1)$:= $\sharp(E_1) + \sharp(E_2)$	
$\sharp(\text{Enc}_{E_2}^a E_1)$:= $\sharp(E_1) + \sharp(E_2)$	
$\sharp(\text{op}(E))$:= $\sharp(E)$	$\text{op} \in \{\text{pub}, \text{priv}, \text{H}\}$
$\sharp(\pi_1(E))$:= $1 + \sharp(E)$	if $E = (E_1 . E_2)$
	:= $\sharp(E)$	otherwise
$\sharp(\pi_2(E))$:= $1 + \sharp(E)$	if $E = (E_1 . E_2)$
	:= $\sharp(E)$	otherwise
$\sharp(\text{Dec}_F^s E)$:= $1 + \sharp(E) + \sharp(F)$	if $E = \text{Enc}_{E_2}^s E_1$
	:= $\sharp(E) + \sharp(F)$	otherwise
$\sharp(\text{Dec}_F^a E)$:= $1 + \sharp(E) + \sharp(F)$	if $E = \text{Enc}_{E_2}^a E_1$
	:= $\sharp(E) + \sharp(F)$	otherwise

Table 6.8: Definition of $\sharp(E)$

Intuitively, $\sharp(E)$ is the number of potential redexes in E . It helps to characterise precisely those expressions E such that $\mathbf{e}_a(E) = E$.

Lemma 90:

Let $E \in \mathbf{E}$. Then $\mathbf{e}_a(E) = E \iff \sharp(E) = 0$.

PROOF

\Rightarrow We show by induction on E that $\sharp(\mathbf{e}_a(E)) = 0$. Thus, if $\mathbf{e}_a(E) = E$, then $\sharp(\mathbf{e}_a(E)) = \sharp(E) = 0$.

⇐ By induction on E . ■

Lemma 91:

Let $E \in \mathbf{E}$, $\sigma : N \rightarrow M$ a substitution such that $x \notin n(\sigma)$. If $\mathbf{e}_c(E\sigma) = M \in \mathbf{M}$, $\sharp(E) = 0$ and $x \in n(E)$ then $x \triangleleft E$.

PROOF

By induction on E . We give the proof for several cases:

- $E = a \in N$

Necessarily, $a = x$ thus $z \triangleleft E$.

- $E = (E_1 \cdot E_2)$ and the result holds for E_1 and E_2

Clearly we have $\sharp(E_1) = \sharp(E_2) = 0$.

Since $n(E) = n(E_1) \cup n(E_2)$, we have $x \in n(E_1)$ or $x \in n(E_2)$.

Moreover, since $\mathbf{e}_c(E\sigma) \neq \perp$, we have necessarily $\mathbf{e}_c(E_1\sigma) \neq \perp$ and $\mathbf{e}_c(E_2\sigma) \neq \perp$.

If $x \in n(E_1)$ then by induction $x \triangleleft E_1$. Hence $x \triangleleft E$.

If $x \in n(E_2)$ then by induction $x \triangleleft E_2$. Hence $x \triangleleft E$.

In both cases, we have $x \triangleleft E$.

- $E = \text{Dec}_C^S F$ and the result holds for F and G

Since $\sharp(E) = 0$, we have $F \neq \text{Enc}_{F_2}^S F_1$, $\sharp(F) = 0$ and $\sharp(G) = 0$.

Since $\mathbf{e}_c(E\sigma) \neq \perp$, we have $\mathbf{e}_c(F\sigma) = \text{Enc}_{M_2}^S M_1$ and $\mathbf{e}_c(G\sigma) = M_2$ for some $M_1, M_2 \in \mathbf{M}$. Thus $\mathbf{e}_c(F\sigma) \neq \perp$ and $\mathbf{e}_c(G\sigma) \neq \perp$.

Since $n(E) = n(F) \cup n(G)$ we have $x \in n(F)$ or $x \in n(G)$.

If $x \in n(F)$, then by induction $x \triangleleft F$. By case analysis on $x \triangleleft F$ and since $\mathbf{e}_c(F\sigma) = \text{Enc}_{M_2}^S M_1$ and $x \notin n(\sigma)$, we necessarily have that $F = \text{Enc}_{F_2}^S F_1$ for some $F_1, F_2 \in \mathbf{E}$. This is a contradiction.

So necessarily $x \in n(G)$. By induction we have $x \triangleleft G$. Since $\mathbf{e}_c(G\sigma) \neq \perp$ and $x \triangleleft G$ we have by Lemma 89 that $x \in n(\mathbf{e}_c(G\sigma)) = n(M_2)$. So, since $\mathbf{e}_c(F\sigma) = \text{Enc}_{M_2}^S M_1$, we have $x \in n(\mathbf{e}_c(F\sigma))$. Since $x \notin n(\sigma)$, we necessarily have that $x \in n(F)$. This leads to a contradiction.

This case is thus impossible. ■

Corollary 10:

Let $E \in E$ and $\sigma : N \rightarrow M$ a substitution such that $x \notin n(\sigma)$. If $e_c(E\sigma) = M \in M$, $\sharp(E) = 0$ then $x \in n(E) \iff x \in n(M)$.

PROOF

\Rightarrow By Lemma 91, we have $x \triangleleft E$.

So by Lemma 89, we have $x \in n(M)$.

\Leftarrow Trivial. ■

Definition 95.

We extend the definition of \sharp to symbolic actions and symbolic agents as follows:

$$\begin{array}{llll} \sharp(\tau) & := & 0 & \sharp(P) & := & 0 \\ \sharp(E) & := & \sharp(E) & \sharp((x)P) & := & 0 \\ \sharp(\bar{E}) & := & \sharp(E) & \sharp((v\bar{z})\langle F \rangle P) & := & \sharp(F) \end{array}$$

Lemma 92:

If $P \xrightarrow[c]{\mu} A$ then $\sharp(\mu) = \sharp(A) = 0$.

PROOF

By induction on $P \xrightarrow[c]{\mu} A$ and thanks to Lemma 90. ■

The next lemma is the counterpart of Lemma 87 for $>_0^e$.

Lemma 93:

1. Let A, B two agents such that $A >_0^e B$, and $P, Q \in \mathbf{P}$ such that $P >_0 Q$. Then $A | P >_0^e B | Q$.
2. Let F_1, F_2 two abstractions such that $F_1 >_0^e F_2$ and C_1, C_2 two concretions such that $C_1 >_0^e C_2$. Then $F_1 \bullet C_1 >_0^e F_2 \bullet C_2$.
3. Let A, B two agents such that $A >_0^e B$.
 - if both A and B are processes then $(\nu x) A >_0^e (\nu x) B$
 - if both A and B are abstractions then $(\nu x) A >_0^e (\nu x) B$
 - if $A = (v\bar{z})\langle E \rangle P$ and $B = (v\bar{z})\langle M \rangle Q$ and $x \in n(E) \iff x \in n(M)$ then $(\nu x) A >_0^e (\nu x) B$.
4. Let A, B two agents and $\sigma : N \rightarrow M$ a substitution such that $A\sigma >_0^e B$ and $\sharp(A) = 0$. Let $x \notin n(\sigma)$. Then $(\nu x) (A\sigma) >_0^e (\nu x) B$.

PROOF

1. By induction on $A >_0^e B$.
2. We have $F_1 =_{\alpha} (x)P$, $F_2 =_{\alpha} (x)P'$, $C_1 =_{\alpha} (v\bar{z}) \langle E \rangle Q$ and $C_2 =_{\alpha} (v\bar{z}) \langle M \rangle Q'$ with $\{\bar{z}\} \subseteq n(E)$, $\{\bar{z}\} \subseteq n(M)$, $\mathbf{e}_c(E) = M$, $P >_0 P'$, $Q >_0 Q'$ and $\{\bar{z}\} \cap \text{fn}(P) = \{\bar{z}\} \cap \text{fn}(P') = \emptyset$.

By definition,

$$\begin{aligned} F_1 \bullet C_1 &= (v\bar{z}) (P\{^E/x\} | Q) \\ F_2 \bullet C_2 &= (v\bar{z}) (P'\{^M/x\} | Q') \end{aligned}$$

To show the result, it is sufficient to show that $P\{^E/x\} >_0 P'\{^M/x\}$. This is a simple consequence of Corollary 9, since $\mathbf{e}_c(E) = M$ and thus $E >_0 M$.

3. The non trivial case is when both A and B are concretions.

In this case, we have $A = (v\bar{z}) \langle E \rangle P$ and $B = (v\bar{z}) \langle M \rangle Q$ with $P >_0 Q$, $\mathbf{e}_c(E) = M$, $\{\bar{z}\} \subseteq n(E)$ and $\{\bar{z}\} \subseteq n(M)$. Since $x \in n(E) \iff x \in n(M)$, we have two cases.

- if $x \in n(E)$. Then $x \in n(M)$.

By definition,

$$\begin{aligned} (vx) A &= (vx\bar{z}) \langle E \rangle P \\ (vx) B &= (vx\bar{z}) \langle M \rangle Q \end{aligned}$$

Thus clearly $(vx) A >_0^e (vx) B$.

- if $x \notin n(E)$. Then $x \notin n(M)$.

By definition,

$$\begin{aligned} (vx) A &= (v\bar{z}) \langle E \rangle (vx) P \\ (vx) B &= (v\bar{z}) \langle M \rangle (vx) Q \end{aligned}$$

Thus $(vx) A >_0^e (vx) B$.

4. The non trivial case is when both A and B are concretions.

Then $A = (v\bar{z}) \langle E \rangle P$ and $B = (v\bar{z}) \langle M \rangle Q$ with $\mathbf{e}_c(E\sigma) = M$, $P\sigma >_0 Q$, $\{\bar{z}\} \subseteq n(E)$, $\{\bar{z}\} \subseteq n(M)$ and $\sharp(E) = 0$.

Since $\sharp(E) = 0$, $\mathbf{e}_c(E\sigma) = M$ and $x \notin n(\sigma)$, by Corollary 10 we have $x \in n(E\sigma) \iff x \in n(M)$. Thus, thanks to the previous result, we have $(vx) (A\sigma) >_0^e (vx) B$. ■

Lemma 94:

1. Let c_1, c_2 be two constraints. Then $\mathbf{nc}(c_1 \ \& \ c_2) = \mathbf{nc}(c_1) \cup \mathbf{nc}(c_2)$
2. Let c be a constraint. Then $\mathbf{nc}(v_+(x, c)) = \mathbf{nc}(c) \setminus \{x\}$. □

PROOF (OF THEOREM 14)

The goal is to prove that

If $P, Q \in \mathbf{P}$ and $\sigma : N \rightarrow M$ is a substitution then:

1. If $P \xrightarrow[c]{\mu_s} A$ and $\mathbf{e}(c\sigma) = \mathbf{true}$ then $P\sigma \xrightarrow[\mathbf{nc}(c\sigma)]{\mathbf{e}_c(\mu_s\sigma)} B$ with $A\sigma >_0^{\mathbf{e}} B$
 2. If $P\sigma \xrightarrow[S]{\mu} B$ then $P \xrightarrow[c]{\mu_s} A$ with $\mathbf{e}(c\sigma) = \mathbf{true}$, $\mathbf{nc}(c\sigma) = S$, $\mathbf{e}_c(\mu_s\sigma) = \mu$ and $A\sigma >_0^{\mathbf{e}} B$
1. By induction on $P \xrightarrow[c]{\mu_s} A$. We give the proof for the following cases:

S-INPUT Assume $P = E(x).P' \xrightarrow[\{\{E:N\}\}]{\mathbf{e}_a(E)} (x)P' = A$. We have $c = \{\{E:N\}\}$ and $\mu_s = \mathbf{e}_a(E)$.

Since $\mathbf{e}(c\sigma) = \mathbf{true}$, there exists $a \in N$ such that $\mathbf{e}_c(E\sigma) = a$.

Thus, by NC-INPUT, $P\sigma \xrightarrow[\{a\}]{a} (x)P'\sigma$.

By Lemma 80, we have $\mathbf{e}_c(\mathbf{e}_a(E)\sigma) = a$. Moreover, it is clear that $\mathbf{nc}(c\sigma) = \{a\}$ and $A\sigma >_0^{\mathbf{e}} (x)P'\sigma$. Hence the result.

S-OUTPUT Assume that $P = \overline{E}\langle F \rangle.P' \xrightarrow[\{\{E:N\}, \{F:M\}\}]{\overline{\mathbf{e}_a(E)}} A = \langle \mathbf{e}_a(F) \rangle P'$. We have $c = \{\{E:N\}, \{F:M\}\}$ and $\mu_s = \overline{\mathbf{e}_a(E)}$.

Since $\mathbf{e}(c\sigma) = \mathbf{true}$, there exists $a \in N$ and $M \in M$ such that $\mathbf{e}_c(E\sigma) = a$ and $\mathbf{e}_c(F\sigma) = M$.

Thus, by NC-OUTPUT, $P\sigma \xrightarrow[\{a\}]{\overline{a}} \langle M \rangle P'\sigma$.

By Lemma 80, we have $\mathbf{e}_c(\overline{\mathbf{e}_a(E)\sigma}) = \overline{a}$ and $\mathbf{e}_c(\mathbf{e}_a(F)\sigma) = M$. Thus $A\sigma >_0^{\mathbf{e}} \langle M \rangle P'\sigma$.

Moreover, it is clear that $\mathbf{nc}(c\sigma) = \{a\}$.

Hence the result.

S-CLOSE-L Assume that $P = P_1 \mid P_2$, $P_1 \xrightarrow{E}_{c_1} F$, $P_2 \xrightarrow{E'}_{c_2} C$ and $P \xrightarrow{\tau}_{\{[E=E']\} \& c_1 \& c_2} A = F \bullet C$.

We have $c = \{[E=E']\} \& c_1 \& c_2$. Since $\mathbf{e}(c\sigma) = \mathbf{true}$, we have $\mathbf{e}(c_1\sigma) = \mathbf{e}(c_2\sigma) = \mathbf{true}$ and there exists $M \in \mathcal{M}$ such that $\mathbf{e}_c(E\sigma) = \mathbf{e}_c(E'\sigma) = M$.

By induction, we have $P_1\sigma \xrightarrow[\mathbf{nc}(c_1\sigma)]{\mathbf{e}_c(E\sigma)} F'$ and $P_2\sigma \xrightarrow[\mathbf{nc}(c_2\sigma)]{\mathbf{e}_c(E'\sigma)} C'$ with $F\sigma >_0^e F'$ and $C\sigma >_0^e C'$.

Since $\mathbf{e}_c(E\sigma) = M$ is a concrete action, we have $M = a \in \mathcal{N}$.

Thus, by NC-CLOSE-L, $P\sigma = P_1\sigma \mid P_2\sigma \xrightarrow[\mathbf{nc}(c_1\sigma) \cup \mathbf{nc}(c_2\sigma)]{\tau} F' \bullet C'$.

By Lemma 93, we have $F\sigma \bullet C\sigma >_0 F' \bullet C'$ so $F\sigma \bullet C\sigma >_0^e F' \bullet C'$.

Since $(F \bullet C)\sigma = F\sigma \bullet C\sigma$ we thus have $A\sigma >_0^e F' \bullet C'$.

Moreover, we have by Lemma 94 $\mathbf{nc}(c\sigma) = \mathbf{nc}(\{[E=E']\}\sigma) \cup \mathbf{nc}(c_1\sigma) \cup \mathbf{nc}(c_2\sigma)$, i.e. $\mathbf{nc}(c\sigma) = \mathbf{nc}(c_1\sigma) \cup \mathbf{nc}(c_2\sigma)$.

Hence the result.

S-RES Assume that $P = (\nu z)P'$ with $P' \xrightarrow{\mu_s}_c A'$, $P = (\nu z)P' \xrightarrow{\mu_s}_{\nu_+(z,c)} (\nu z)A$ and $z \notin \mathbf{n}(\mu_s)$.

We have $\mathbf{e}(\nu_+(z,c)) = \mathbf{e}(c) = \mathbf{true}$.

So by induction, we have $P'\sigma \xrightarrow[\mathbf{nc}(c\sigma)]{\mathbf{e}_c(\mu\sigma)} B$ with $A'\sigma >_0^e B$.

By Lemma 92, we have that $\sharp(\mu_s) = \sharp(A') = 0$.

Since $\mathbf{e}_c(\mu_s\sigma) \neq \perp$, $\sharp(\mu_s) = 0$ and $z \notin \mathbf{n}(\sigma)$ (by convention, bound names are chosen different from free names), a simple consequence of Lemma 91 and $z \notin \mathbf{n}(\mu_s)$ is that $z \notin \mathbf{n}(\mathbf{e}_c(\mu_s\sigma))$.

So, by NC-RES, we have $(\nu z)P'\sigma \xrightarrow[\mathbf{nc}(c\sigma) \setminus \{z\}]{\mathbf{e}_c(\mu\sigma)} (\nu z)B$.

By Lemma 94, we have $\mathbf{nc}(\nu_+(z,c)) = \mathbf{nc}(c) \setminus \{z\}$. Moreover, by Lemma 93, we have $A\sigma = (\nu z)(A'\sigma) >_0^e (\nu z)B$. Hence the result.

2. By induction on $P\sigma \xrightarrow{\mu}_S B$. We give the proof for the following cases:

NC-INPUT Assume that $P\sigma = E\sigma(x).P'\sigma \xrightarrow[\{a\}]{a} B = (x)P'\sigma$ with $\mathbf{e}_c(E\sigma) = a \in \mathcal{N}$.

We have $P = E(P)'$. By S-INPUT, we have $P \xrightarrow[\{[E:N]\}]{\mathbf{e}_a(E)} (x)P'$.

By Lemma 80, we have $\mathbf{e}_c(\mathbf{e}_a(E)\sigma) = a$.

Thus $\mathbf{e}(\{[E:N]\}\sigma) = \mathbf{true}$ and $\mathbf{nc}(\{[E:N]\}\sigma) = \{a\}$.

Moreover $A\sigma = ((x)P')\sigma >_0^{\mathbf{e}} B$.

Hence the result.

NC-IFTHEN Assume that $P\sigma = \phi\sigma P'\sigma$ with $P'\sigma \xrightarrow[S]{\mu} B$, $\mathbf{e}(\phi\sigma) = \mathbf{true}$ and

$$P \xrightarrow[S \cup \mathbf{nc}(\phi\sigma)]{\mu} B.$$

We have $P = \phi P'$.

By induction, we have $P' \xrightarrow[c]{\mu_s} A$ with $\mathbf{e}(c\sigma) = \mathbf{true}$, $\mathbf{nc}(c\sigma) = S$, $\mathbf{e}_c(\mu_s\sigma) = \mu$ and $A\sigma >_0^{\mathbf{e}} B$.

By S-IFTHEN, we have $P \xrightarrow[c \& \{\phi\}]{\mu_s} A$.

Since $\mathbf{e}(\phi\sigma) = \mathbf{true}$ and $\mathbf{e}(c\sigma) = \mathbf{true}$, we clearly have $\mathbf{e}((c \& \{\phi\})\sigma) = \mathbf{e}(c\sigma \& \{\phi\sigma}) = \mathbf{true}$.

By Lemma 94, we have $\mathbf{nc}((c \& \{\phi\})\sigma) = \mathbf{nc}(c\sigma \& \{\phi\sigma}) = \mathbf{nc}(c\sigma) \cup \mathbf{nc}(\phi\sigma) = S \cup \mathbf{nc}(\phi\sigma)$.

Hence the result.

NC-PAR-L Assume that $P\sigma = P_1\sigma | P_2\sigma$ with $P_1\sigma \xrightarrow[S]{\mu} B'$ and $P_2\sigma \xrightarrow[S]{\mu} B = B' | P_2\sigma$.

By induction, we have $P_1 \xrightarrow[c]{\mu_s} A'$ with $\mathbf{e}(c\sigma) = \mathbf{true}$, $\mathbf{nc}(c\sigma) = S$, $\mathbf{e}_c(\mu_s\sigma) = \mu$ and $A'\sigma >_0^{\mathbf{e}} B'$.

By S-PAR-L, we have $P = P_1 | P_2 \xrightarrow[c]{\mu_s} A = A' | P_2$.

By Lemma 93, we have $A\sigma >_0^{\mathbf{e}} B$ since $A'\sigma >_0^{\mathbf{e}} B'$ and $P_2 >_0^{\mathbf{e}} P_2$.

Hence the result. \blacksquare

6.2.4 Late hedged bisimilarity in coq

Once hedges, spi calculus terms and labelled semantics are defined in coq, it is easy to define the notion of late hedged bisimulation.

Our formal definition in coq follows closely Definition 69. A hedged relation is thus a predicate on tuples (h, P, Q) where h is a hedge h and P and Q are two processes. We define the notion of well-formedness, symmetry and consistency for such predicates. A late hedged bisimulation is

thus a well-formed, symmetric and consistent hedged relation that satisfies the bisimulation clauses.

We illustrate how bisimulations can be exploited in coq through two small examples inspired by [8].

Example 1

Define $P(c, M) := (\nu k) \bar{c}(\text{Enc}_k^s M). \mathbf{0}$ where $k \notin \{c\} \cup \text{n}(M)$.

We show that for any c, M and N we have $P(c, M) \sim_{\text{LH}}^h P(c, N)$ where $h = \mathcal{I}(\{(c, c), (M, M), (N, N)\})$.

Intuitively, this means that $P(c, M)$ and $P(c, M')$ do not reveal M and M' respectively.

With de Bruijn representation, $P(c, M)$ is defined by

$$P(c, M) := \overline{\nu \text{lift}_0(1, c)}(\text{Enc}_0^s \text{lift}_0(1, M)). \mathbf{0}$$

Note in particular how the condition $k \notin \{c\} \cup \text{n}(M)$ is satisfied by definition thanks to the $\text{lift}_0(1, \cdot)$ operation.

We define

$$\begin{aligned} h_0(c, M, N) &:= \mathcal{I}(\{(c, c), (M, M), (N, N)\}) \\ h_1(c, M, N, k, l) &:= h_0 \cup \{(\text{Enc}_k^s M, \text{Enc}_k^s N)\} \end{aligned}$$

It is easy to prove that $h_0(c, M, N)$ is consistent for any c, M and N because it is the irreducible part of a reflexive hedge (i.e. a hedge h such that whenever $(M, N) \in h$ we have $M = N$). $h_0(c, M, N)$ is also a reflexive hedge and we have $h_0(c, M, N)^{-1} = h_0(c, N, M) = h_0(c, M, N)$.

Moreover, if $k \notin \text{n}(\pi_1(h_0(c, M, N)))$ and $l \notin \text{n}(\pi_2(h_0(c, M, N)))$ (i.e. k and l are fresh) then $h_1(c, M, N, k, l)$ is consistent. In this case, by Theorem 9, we have $\mathcal{I}(h_1(c, M, N, k, l)) = h_1(c, M, N, k, l)$.

Moreover we have obviously that $h_1(c, M, N, k, l)^{-1} = h_1(c, N, M, l, k)$.

Let $c \in \mathcal{N}$ and $M, N \in \mathcal{M}$. We define

$$\begin{aligned} \mathcal{R} := & \{(h_0(c, M, N), P(c, M), P(c, N))\} \\ & \cup \{(h_1(c, M, N, k, k), \mathbf{0}, \mathbf{0}) \mid k \notin \text{n}(h_0(c, M, N))\} \\ & \cup \{(h_0(c, N, M), P(c, N), P(c, M))\} \\ & \cup \{(h_1(c, N, M, k, k), \mathbf{0}, \mathbf{0}) \mid k \notin \text{n}(h_0(c, N, M))\} \end{aligned}$$

We check that \mathcal{R} is a late hedged bisimulation.

Thus, we have $P(c, M) \sim_{\text{LH}}^{h_0(c, M, N)} P(c, N)$.

Example 2

As a small variant, consider $Q(c, M, M') = (\nu k) \bar{c} \langle \text{Enc}_k^s M \rangle . \bar{c} \langle \text{Enc}_k^s M' \rangle . \mathbf{0}$ where $k \notin \{c\} \cup n(M, M')$.

We show that for any c, M, N and N' , if $N \neq N'$ then there is no hedge h such that $(c, c) \in h$ (i.e. the channel c is known by the attacker) and $Q(c, M, M) \sim_{\text{LH}}^h Q(c, N, N')$.

Intuitively, this means that even if two cyphertexts cannot be decrypted, they can still be compared.

Let $c \in N, M, N, N' \in M$ such that $N \neq N'$.

The proof proceeds by contradiction. Assume that there is h such that $(c, c) \in h$ and $Q(c, M, M) \sim_{\text{LH}}^h Q(c, N, N')$. We show that necessarily $N = N'$.

Assume that \mathcal{R} is a late hedged bisimulation such that

$$(h, Q(c, M, M), Q(c, N, N')) \in \mathcal{R}$$

Since \mathcal{R} is a bisimulation, we have that

$$\begin{aligned} (\mathcal{I}(h \cup \{\text{Enc}_k^s M, \text{Enc}_l^s N\}), \bar{c} \langle \text{Enc}_k^s M \rangle . \mathbf{0}, \bar{c} \langle \text{Enc}_l^s N' \rangle . \mathbf{0}) &\in \mathcal{R} \\ (\mathcal{I}(\mathcal{I}(h \cup \{\text{Enc}_k^s M, \text{Enc}_l^s N\}) \cup \{\text{Enc}_k^s M, \text{Enc}_l^s N'\}), \mathbf{0}, \mathbf{0}) &\in \mathcal{R} \end{aligned}$$

for some $k \notin n(\pi_1(h))$ and $l \notin n(\pi_2(h))$.

Define

$$\begin{aligned} h' &:= \mathcal{I}(h \cup \{\text{Enc}_k^s M, \text{Enc}_l^s N\}) \\ h'' &:= \mathcal{I}(h' \cup \{\text{Enc}_k^s M, \text{Enc}_l^s N'\}) \end{aligned}$$

Since \mathcal{R} is a consistent hedged relation, we necessarily have that h'' is consistent and thus is left consistent.

Hence, by Lemma 60, we have

$$\forall (M, N), (M', N') \in \mathcal{S}(h'') : M' = M \implies N' = N$$

We now show that $(\text{Enc}_k^s M, \text{Enc}_l^s N) \in \mathcal{S}(h'')$ and $(\text{Enc}_k^s M, \text{Enc}_l^s N') \in \mathcal{S}(h'')$. It will follow that $\text{Enc}_l^s N = \text{Enc}_l^s N'$ and so $N = N'$.

By Lemma 45, we have $h' \cup \{\text{Enc}_k^s M, \text{Enc}_l^s N'\} <_{\mathbf{H}} h''$.

So clearly $(\text{Enc}_k^s M, \text{Enc}_l^s N') \in \mathcal{S}(h'')$.

Still by Lemma 45, we have $h \cup \{\text{Enc}_k^s M, \text{Enc}_l^s N\} <_{\mathbf{H}} h'$. Moreover by Corollary 1, we have $h' <_{\mathbf{H}} h' \cup \{\text{Enc}_k^s M, \text{Enc}_l^s N'\}$. Hence, by transitivity of $<_{\mathbf{H}}$, we have $h \cup \{\text{Enc}_k^s M, \text{Enc}_l^s N\} <_{\mathbf{H}} h''$.

So clearly $(\text{Enc}_k^s M, \text{Enc}_l^s N) \in \mathcal{S}(h'')$.

Hence $N = N'$. This is a contradiction.

Actually, when $N \neq N'$, $Q(c, M, M)$ and $Q(c, N, N')$ are not even testing equivalent. Indeed, the following environment distinguishes these two processes.

$$R := c(x).c(y).[x=y]\bar{c}\langle c \rangle.$$

Conclusion

In this chapter, we have presented our formalisation of the spi calculus in `coq`.

Related works include a formalisation of the polyadic pi calculus in `coq` using de Bruijn indices [81, 82], a formalisation of the pi calculus in Isabelle using the higher-order abstract syntax approach [121] and a recent formalisation of the pi calculus in Isabelle using the nominal approach [24].

To ease the manipulation of de Bruijn terms and to keep small the part devoted to show technical results about de Bruijn indices, we have sketched a `coq` library to handle de Bruijn terms. This was technically difficult to achieve because of some limitations of the `coq` module system: we finally got rid of these limitations by representing modules as records (like in the object-oriented world). We think this contribution is important and it will certainly be interesting to evaluate the usability of our de Bruijn library for formalising other programming languages or for instance for contributing to the POPLMARK challenge.

We have also sketched some other abstractions for defining the labelled semantics. Hence, we have proved once and for all the theorem stating that structural congruence preserves the semantics under suitable assumptions. As a corollary, we have thus formally proved in `coq` Theorem 4 for the late semantics of the pi calculus and the late semantics of the spi calculus with or without type constraints. Indeed, we have not only formalised the spi calculus but also the monadic pi calculus, both being an instance of an abstract pi calculus (where expressions E and guards ϕ remain abstract) which is defined by:

$$\begin{array}{l}
 P, Q ::= \mathbf{0} \mid P + Q \\
 \quad \mid P \mid Q \mid !P \\
 \quad \mid (\nu x)P \mid \tau.P \\
 \quad \mid E(x).P \mid \bar{E}\langle F \rangle.P \\
 \quad \mid \phi P
 \end{array}$$

It would be interesting to see if we can easily adapt our definitions to also model the applied pi calculus [6].

It is worth noting that the use of the abstraction/concretion presentation has also greatly simplified the definition of the semantics and the presentation of the proofs because it naturally enforces to separate technical lemmas from main results.

To catch a glimpse of our `coq` formalisation, we have listed below all the modules together with a quick description and the weight in terms of lines of `coq` code. It is composed of 41 modules for a total of about 33000 lines of `coq` code. It takes approximately 8 minutes to compile on a recent computer (Core 2 Duo at 2.16 GHz) with the latest version of `coq`.

Even if we have only partially reached our goal (i.e. extract a certified bisimulation checker), we think this work was worth the effort because we have been able to catch and repair some subtle bugs in the definitions or in the handwritten paper proofs. Moreover, it already provides a proof environment for reasoning formally about cryptographic protocols using the spi calculus framework.

As a first step towards the extraction of a fully certified bisimulation checker, we have been able to extract a tool for computing the transitions of pi calculus or spi calculus terms. This tool offers the choice of the used semantics among the following: standard late semantics of the pi calculus, open semantics of the pi calculus, standard late semantics of the spi calculus, late semantics with type constraints of the spi calculus, symbolic semantics of the spi calculus. For having written several labelled semantics in `ocaml` (e.g. [42, 38]), we can say that this is an invaluable first step because even this simple part can be error-prone. As obvious future work, we intend to complete our formalisation of the spi calculus by defining open hedged and symbolic open hedged bisimulation and study them. We then intend to focus on the decision algorithm for checking open bisimulation on finite terms.

Name	Description	Lines
<code>tactics</code>	Definitions of some useful naive tactics.	47
<code>missing</code>	Auxiliary results that are missing in the <code>coq</code> standard library.	512

Name	Description	Lines
deBruijnNat	Implementation of de Bruijn operations on natural numbers and proofs of technical results.	910
deBruijnType	Specification of de Bruijn operations and definitions of de Bruijn structure "lifter"	1846
name	Embedding of de Bruijn indices into an inductive type.	443
substLemma	The substitution lemma for de Bruijn indices.	263

Name	Description	Lines
expressionType	The abstract type of expressions.	44
formulaeType	The abstract type of guards.	43
processType	The abstract type of abstract processes.	108
process	Implementation of abstract processes as a functor.	587
agentType	The abstract type of agents.	135
agent	Implementation of abstract agents over abstract processes.	780
congruence	Definition and properties of the structural congruence over abstract processes and agents.	1838

Name	Description	Lines
expressionPi	Implementation of pi calculus expressions.	174
formulaePi	Implementation of pi calculus guards.	202

Name	Description	Lines
opSpi	Definition of operators (i.e. {pub, priv, H}) for spi calculus messages/expressions.	93
messageSpi	Implementation of spi calculus messages.	393
expressionSpi	Implementation of spi calculus expressions.	435
formulaeSpi	Implementation of spi calculus guards.	216

Name	Description	Lines
calculi	Instantiation of the pi calculus and the spi calculus.	34

actionType	The abstract type of actions for defining LTS.	186
reduction	Definition of the LTS as a functor parametrised by a calculus and its actions. Definition of a certified function to compute transitions.	1127
redcong	Generic proof of Theorem 4.	1700

Name	Description	Lines
actionPi	Actions for the monadic pi calculus	552
reductionPi	Instantiation of the late semantics of the pi calculus.	41

Name	Description	Lines
evalSpi	Definition and properties of evaluation functions of the spi calculus.	1043
actionSpi	Actions for the spi calculus	582
actionSpiNC	Actions with type constraints for the spi calculus	954
actionSpiSymb	Symbolic actions for the spi calculus	1218
reductionSpi	Instantiation of the three semantics of the spi calculus and proofs of some auxiliary results. Proof of Theorem 3.	341

Name	Description	Lines
workAround	Auxiliary results to get rid of coq type system problems.	145
substSpi	Proof of Lemma 80 and Lemma 79.	437
redSpiNCprops	Proof of Lemma 24.	588
roSpi	Definition of $>_o$ and $>_o^=$. Proof of Theorem 13.	1388
fnInMsg	Auxiliary results for proving Theorem 14	449
redSpiSymProps	Definition of $>_o^e$. Proof of Theorem 14.	1990

Name	Description	Lines
<code>ordmissing</code>	Auxiliary results that are missing in the <code>FSet coq</code> library.	1976
<code>set_order</code>	Definition of the well-founded order used to prove the existence of the analysis of a hedge.	638
<code>hedges</code>	Definitions and properties related to hedges (see Chapter 4).	6913
Name	Description	Lines
<code>spienv</code>	Auxiliary definitions for defining late hedged bisimulation.	370
<code>LHbisimulation</code>	Definition of late hedged bisimulation. Proof of a perfect encryption equation.	1417

Chapter 7

From Protocol Narrations to Spi Calculus

Protocol narrations are a widely-used informal means to describe, in an idealistic manner, the functioning of cryptographic protocols as a single intended sequence of cryptographic message exchanges among the protocol's participants. Protocol narrations have also been informally "turned into" a number of formal protocol descriptions, e.g., using the spi calculus. In this chapter, we propose a direct formal operational semantics for protocol narrations that fixes a particular and, as we argue, well-motivated interpretation on how the involved protocol participants are supposed to execute. Based on this semantics, we explain and formally justify a natural and precise translation of narrations into spi calculus. An optimised translation has been implemented in `ocaml`, and we report on case studies that we have carried out using the tool.

7.1 Introduction

7.1.1 The setting

As we mentioned in Chapter 1, in the cryptographic protocol literature, protocols are usually expressed as narrations [58, 95]. The protocol in Table 7.1 is a typical example of this style. Two principals A and B are both connected to the server S with whom they share the secret keys k_{AS} and k_{BS} , respectively. The protocol tells the story where A wants to establish a secret connection (a shared key k_{AB}) with B via the common server S : first, A should contact S , then S forwards the key k_{AB} to B . Finally, A uses this key to exchange secret data with B . In this chapter, we focus our in-

$$\begin{aligned}
A \rightarrow S &: (A . \text{Enc}_{k_{AS}}^s (B . k_{AB})) \\
S \rightarrow B &: \text{Enc}_{k_{BS}} (A . (B . k_{AB})) \\
A \rightarrow B &: \text{Enc}_{k_{AB}} m
\end{aligned}$$

Table 7.1: Wide-Mouthed Frog protocol

terest on the bare operational content of the description technique of narrations.

Our own motivation for the interest in a formal semantics for narrations is that we had implemented a “straightforward” translator [74] from protocol narrations into the spi calculus. We then wanted to formally prove our translator correct but faced the problem that there was no formal intended semantics to compare to. This lacking semantics is what we provide within this chapter. Indeed, it turns out that the attempt to properly formalise narrations brings one already much closer to spi-like executable descriptions, but there are a number of insightful observations along the way, on which we report here as well.

7.1.2 The challenge

Despite being rather intuitive, the description technique of protocol narrations contains lots of implicit concepts. Looking for a formal semantics, these need to be rendered explicit. For example, Abadi [2] pointed out that “informal protocol narrations” need to be complemented with explanations of some either implicitly assumed facts or additional information to remove ambiguities. He raised four tasks that need to be pursued:

1. One should make explicit what is known (public, private) before a protocol run, and what is to be generated freshly during a protocol run.
2. One should make explicit which checks the individual principals are expected to carry out on the reception of messages.
3. Principals act concurrently, in contrast to the apparently sequential idealised execution of a run according to a narration.
4. Concurrency occurs also at the level of different protocol sessions, which may happen to be executed simultaneously while sharing principals across.

(Interestingly, Abadi used these requirements to motivate the use of the spi calculus as a description technique for “formal protocol narrations”.)

The first item above should be clear: data is missing otherwise. To this aim, narrations usually come with a bit of explanation in natural language on the spirit of the protocol and on the assumptions made. Essentially, these assumptions consist of expliciting the *pieces of data* known in advance by the agents¹ and those that are to be *freshly generated* during the course of a protocol run.

The second item above results from the too high level of abstraction of message exchanges, noted as $A \rightarrow B : M$. There are a number of problems connected to the fact that message M is usually transmitted from A to B by passing through an asynchronous insecure network where a potential intruder can interfere [68]. Thus, once B receives a message, it may be just the expected one according to the protocol, but it may also be an intended message received at the wrong moment and, worse, it may be an unintended message forged by some malicious attacker. So, B needs to perform some informative checks. But precisely which ones? For example, when B receives M it must first check in how far, at this very moment, it “understands” M (with respect to possible encryptions). Then, if B acquires new knowledge by this analysis, it must ensure that this new knowledge is consistent with its previously acquired knowledge. Some careful analysis is due, requiring a suitable representation of knowledge.

The third item above looks innocent at first, but once the non-atomic passage of messages through the network is properly taken into account, some surprising effects arise due to parts of *later* message exchanges (referring to the order of exchanges in a narration) possibly occurring before *earlier* message exchanges have completed or even started.

The fourth item above is again intuitively straightforward, but the description technique of narrations completely ignores the problem.

7.1.3 Our approach

In this chapter, we present solutions to the first three items, leaving the fourth for future work (see Section 7.8) Concerning the first item, we simply add a declaration part to narrations (Section 7.2). Here, we are no different from competing approaches (see the paragraph on Related work in Section 7.8). On item two, we propose (in Section 7.3) to compile exchanges of the form $A \rightarrow B : M$ into three separate syntactic parts, corresponding to:

¹We use the terms principal and agent interchangeably.

- (i) A asynchronously sends M towards B ,
- (ii) B receives some message (intended to be M), and
- (iii) finally B checks that the message it just received indeed has the expected properties (associated with M , from the point of view of B).

With respect to the required checks, our approach is to automatically generate the maximum of checks derivable from the static information of protocol narrations. We call the resulting refined notion of narrations *executable*, because it will allow us to formalise an operational semantics of narrations, which would not be possible with an atomic, or synchronous, interpretation of message exchanges.

Concerning the third item, we profit from the above decomposition of message transmission and introduce a natural structural equivalence relation on executable narrations that may bring any of the (con-)currently enabled actions to top-level. On this basis, we provide a labelled transition semantics (Section 7.5).

Finally, we rewrite executable narrations within the spi calculus, which is then only a minor, albeit insightful, remaining step (Section 7.6). We then establish a straightforward formal operational correspondence between the two semantics.

7.1.4 Tool support.

We have implemented the previous developments in `ocaml` (see Section 7.7). Due to the overly big size of the generated formulae, we studied possible simplification strategies. To this end, we have implemented naive ideas such as removing duplicated atoms, or removing atoms like $[E:M]$ when E is a message or when it appears as a sub-expression of the remaining formula. We also perform some rewriting inside formulae, which according to our experience gives good results in practise.

7.2 Extending protocol narrations

Like in the competing approaches on the representation of protocol narrations, we extend narrations with a header that declares the initial knowledge of each agent, the names generated by them and also the names that are assumed to be initially only known by the system (this last point permits to simulate a first pass where shared keys are securely distributed among some agents).

Hence, an extended protocol narration is composed of two parts: a sequence of *declarations* followed by the *narration* itself. The agents are picked among a countably infinite set A of *agent names* ranged over by A, B, C, \dots, S, \dots and the messages are built upon a countably infinite set N_\circ of *names* ranged over by $a, b, c, \dots, k, l, m, n, \dots$. For sake of simplicity, we assume that $A \cap N_\circ = \emptyset$.

We implicitly assume that all agents involved in the protocol know each other; this can be generalised by explicit declarations. The syntax of messages and protocol narrations is given in Table 7.2.

messages M_\circ		
$M, N ::= a$		name
A		agent name
$H(M)$		hashing
$\text{pub}(M)$		public key
$\text{priv}(M)$		private key
(M, N)		pair
$\text{Enc}_N^s M$		shared-key encryption
$\text{Enc}_N^a M$		asymmetric encryption
exchanges		
$T ::= A \rightsquigarrow B : M$		exchanges
narrations		
$L ::= \epsilon$		empty narration
$T ; L$		non-empty narration
declarations		
$D ::= A \text{ knows } M$		initial knowledge
$A \text{ generates } n$		fresh name generation
private k		private name
protocol narration D		
$P ::= D ; P$		sequence of declarations
L		narration

Table 7.2: Protocol narrations

This calls several remarks. Firstly, we write message exchanges in narrations $A \rightsquigarrow B : M$ instead of $A \rightarrow B : M$. Secondly, names to construct messages can be of two kinds: “pure” names or agent names. This is only a minor difference compared to the messages used in Chapter 3. Thirdly, as before, to handle asymmetric cryptography, we define the inverse key $\text{inv}(M)$ of a message M to be $\text{pub}(M')$ if $M = \text{priv}(M')$, $\text{priv}(M')$ if $M = \text{pub}(M')$ and \perp otherwise. Note that any participant has the power

to verify if two messages M_1 and M_2 are inverse keys one from each other, simply by trying to decrypt with M_2 a message encrypted with M_1 and conversely.

The meaning of **private** k is that k is a name which is initially only available for the agents involved in the protocol. Typically, it is useful to simulate that an agent A and a server S initially share a secret key k_{AS} . The meaning of A **knows** M is simply that, initially, agent A knows the message M . Finally the meaning of A **generates** n is that A will generate a fresh name n (typically a nonce). For the sake of clarity, we enforce fresh generated names to be declared explicitly. Table 7.3 shows the Wide-Mouthed Frog protocol using our framework.

$$\begin{aligned} & \mathbf{private} \ k_{AS} ; A \ \mathbf{knows} \ k_{AS} ; S \ \mathbf{knows} \ k_{AS} ; \\ & \mathbf{private} \ k_{BS} ; B \ \mathbf{knows} \ k_{BS} ; S \ \mathbf{knows} \ k_{BS} ; \\ & A \ \mathbf{generates} \ k_{AB} ; A \ \mathbf{knows} \ m \\ & A \rightsquigarrow S : (A . \text{Enc}_{k_{AS}}(B . k_{AB})) ; \\ & S \rightsquigarrow B : \text{Enc}_{k_{BS}}(A . (B . k_{AB})) \\ & A \rightsquigarrow B : \text{Enc}_{k_{AB}} m ; e \end{aligned}$$

Table 7.3: Wide-Mouthed Frog protocol, with formal declarations

It often happens in cryptographic protocols that a secret is shared by several participants. For this reason, we propose to introduce as a macro the construct

$$A_1, \dots, A_n \ \mathbf{share} \ k$$

which is intended to mean that the agents A_1, \dots, A_n share the secret name k . This macro is simply expanded into:

$$\mathbf{private} \ k ; A_1 \ \mathbf{knows} \ k ; \dots ; A_n \ \mathbf{knows} \ k$$

To ease the writing of formal declarations, one can also imagine to introduce the shortcut $A_1, \dots, A_n \ \mathbf{knows} \ M$ to mean $A_1 \ \mathbf{knows} \ M ; \dots ; A_n \ \mathbf{knows} \ M$.

7.3 Compiling protocol narrations

7.3.1 Target syntax.

As motivated in the Introduction, *executable narrations* (set X , as defined in Table 7.4) are to be more explicit about the behaviour of individual agents.

Instead of atomic exchanges of the form $A \rightsquigarrow B : M$ as used in the standard narrations of Table 7.2, we observe four more fine-grained basic actions (nonterminal I in Table 7.4): emission $A:B!E$ of a message expression E (evaluating to M , see below), reception $B:?x$ of a message and binding it to a variable x (see below), check $B:\phi$ for the validity of formula ϕ from the point of view of principal B , and scoping νk , which is reminiscent of the spi calculus and represents the creation and scope of private names. Scoping is decoupled from principals, allowing us to use a single construct for names that are **private** and **generated** according to the declarations of Section 7.2.

Expressions E		
$E, F ::=$	a	name
	A	agent name
	x	variable
	$H(E)$	hashing
	$\text{pub}(E)$	public key
	$\text{priv}(E)$	private key
	$\pi_1(E)$	first projection
	$\pi_2(E)$	second projection
	$(E.F)$	pair
	$\text{Enc}_F^S E$	shared-key encryption
	$\text{Enc}_F^A E$	asymmetric encryption
	$\text{Dec}_F^S E$	shared-key decryption
	$\text{Dec}_F^A E$	asymmetric decryption
Formulae F		
$\phi, \psi ::=$	$[E = F]$	matching
	$[E : M]$	well-formedness test
	$\text{inverse}(E, F)$	inverse key test
	$\phi \wedge \psi$	conjunction
	tt	always true
Simple action		
$I ::=$	νk	fresh name generation
	$A:B!E$	message emission
	$A:?x$	message reception
Executable narrations X		
$X ::=$	ϵ	empty narration
	$I ; X$	non empty narration

Table 7.4: Syntax of executable narrations

In interacting systems, when an agent receives a message, it binds it to a fresh variable for reference in subsequent processing. For this purpose, we introduce a well-founded totally ordered countably infinite set x, y, z, \dots of *variables* \mathbf{V} that we assume to be disjoint from $A \cup N_\circ$. Since an agent does not only handle messages but also variables, we introduce the notion of message *expressions* (E), including operations to construct and deconstruct messages. The process of finding out whether some expression indeed “represents” some particular message, is formalised using the *evaluation function* in Table 7.5.

7.3.2 Evaluation of expressions and formulae.

Definition of $\llbracket \cdot \rrbracket : E \rightarrow \{\perp\} \cup \mathbf{M}_\circ$	
$\llbracket E \rrbracket$	$:= E$ if $E \in N_\circ \cup A$
$\llbracket (E.F) \rrbracket$	$:= (M.N)$ if $\llbracket E \rrbracket = M \in \mathbf{M}_\circ$ and $\llbracket F \rrbracket = N \in \mathbf{M}_\circ$
$\llbracket \pi_1(E) \rrbracket$	$:= M$ if $\llbracket E \rrbracket = (M.N) \in \mathbf{M}_\circ$
$\llbracket \pi_2(E) \rrbracket$	$:= N$ if $\llbracket E \rrbracket = (M.N) \in \mathbf{M}_\circ$
$\llbracket \text{Enc}_F^S E \rrbracket$	$:= \text{Enc}_N^S M$ if $\llbracket E \rrbracket = M \in \mathbf{M}_\circ$ and $\llbracket F \rrbracket = N \in \mathbf{M}_\circ$
$\llbracket \text{Dec}_F^S E \rrbracket$	$:= M$ if $\llbracket E \rrbracket = \text{Enc}_N^S M \in \mathbf{M}_\circ$ and $\llbracket F \rrbracket = N \in \mathbf{M}_\circ$
$\llbracket \text{Enc}_F^A E \rrbracket$	$:= \text{Enc}_N^A M$ if $\llbracket E \rrbracket = M \in \mathbf{M}_\circ$ and $\llbracket F \rrbracket = N \in \mathbf{M}_\circ$
$\llbracket \text{Dec}_F^A E \rrbracket$	$:= M$ if $\llbracket E \rrbracket = \text{Enc}_N^A M \in \mathbf{M}_\circ$ and $\llbracket F \rrbracket = \text{inv}(N) \in \mathbf{M}_\circ$
$\llbracket \text{op}(E) \rrbracket$	$:= \text{op}(M)$ if $\llbracket E \rrbracket = M \in \mathbf{M}_\circ$ for $\text{op} \in \{\text{pub}, \text{priv}, \text{H}\}$
$\llbracket E \rrbracket$	$:= \perp$ in all other cases
Definition of $\llbracket \cdot \rrbracket : F \rightarrow \{\text{true}, \text{false}\}$	
$\llbracket \#t \rrbracket$	$:= \text{true}$
$\llbracket \phi \wedge \psi \rrbracket$	$:= \text{true}$ if $\llbracket \phi \rrbracket = \llbracket \psi \rrbracket = \text{true}$
$\llbracket [E = F] \rrbracket$	$:= \text{true}$ if $\llbracket E \rrbracket = \llbracket F \rrbracket = M \in \mathbf{M}_\circ$
$\llbracket [E : M] \rrbracket$	$:= \text{true}$ if $\llbracket E \rrbracket = M \in \mathbf{M}_\circ$
$\llbracket [\text{inverse}(E, F)] \rrbracket$	$:= \text{true}$ if $\llbracket E \rrbracket = M \in \mathbf{M}_\circ$ and $\llbracket F \rrbracket = \text{inv}(M) \in \mathbf{M}_\circ$
$\llbracket \phi \rrbracket$	$:= \text{false}$ in all other cases

Table 7.5: Evaluation of expressions (can fail, in particular if $v(E) \neq \emptyset$) and formulae

Formulae ϕ on received messages are described by (conjunctions of) three kinds of checks: *equality tests* $[E = F]$ on expressions denote the comparison of two bit-streams of E and F ; *well-formedness tests* $[E : M]$ denote

the verification of whether the projections and decryptions contained in E are likely to succeed; *inversion tests* $\text{inverse}(E, F)$ denote the verification that E and F evaluate to inverse messages. The evaluation function of Table 7.5 is straightforwardly extended to formulae; note that, according to it, $[E:M]$ is just a macro for $[E=E]$. Similarly, $\text{inverse}(E, F)$ can be encoded (for example) as $[\text{Dec}_F^a \text{Enc}_E^a(E.F):M]$ (see also Section 7.7).

Table 7.4 lists the syntax of expressions, formulae and executable narrations. In the following, we will omit the trailing $;$ ϵ of a non-empty executable narration. Moreover, we overload the operator $;$ to also concatenate narrations.

Definition 96.

Let $M \in M_\diamond$, $E \in E$, $\phi \in F$, $x \in V$. We let $n(M)$, $n(E)$, and $n(\phi)$ denote the set of names occurring in M , E , and ϕ , respectively. Similarly, we let $v(E)$ and $v(\phi)$ denote the set of variables occurring in E and ϕ . $E\{M/x\}$ and $\phi\{M/x\}$ denote the substitution of M for x in E and ϕ , respectively.

7.3.3 Knowledge representation.

As motivated in the Introduction, the central point of the actual behaviour of protocols is to find out which checks are to be performed. We further motivated that such checks need to be based on (1) the narration code, which statically spells out the intended message to be received, and (2) the current knowledge at the moment of reception, which imposes constraints on how much the recipient can dynamically learn from the received message and on what other information the newly acquired knowledge must be consistent with.

Instead of accumulating only the dynamically acquired messages (stored in variables x) we propose to tightly connect the (according to the narration) statically intended messages M with the dynamically received actual messages x . For this, we simply use pairs (M, x) . Since consistency checks will then (have to) operate on such pairs, we need to generalise this representation of principal knowledge to finite subsets of $M_\diamond \times E$. The underlying idea is that a pair (M, E) means that the expression E is supposed to be equal (or: has to evaluate) to M .

The following definition introduces knowledge sets, and also some traditionally employed operations on them (see also Chapter 4): *synthesis* reflects the closure of knowledge sets using message constructors; *analysis* reflects the exhaustive recursive decomposition of knowledge pairs as enabled by the currently available knowledge.

$$\begin{array}{c}
\text{SYN-PAIR} \frac{(M, E) \in \mathcal{S}(K) \quad (N, F) \in \mathcal{S}(K)}{((M \cdot N), (E \cdot F)) \in \mathcal{S}(K)} \\
\text{SYN-ENC-S} \frac{(M, E) \in \mathcal{S}(K) \quad (N, F) \in \mathcal{S}(K)}{(\text{Enc}_N^s M, \text{Enc}_F^s E) \in \mathcal{S}(K)} \\
\text{SYN-ENC-A} \frac{(M, E) \in \mathcal{S}(K) \quad (N, F) \in \mathcal{S}(K)}{(\text{Enc}_N^a M, \text{Enc}_F^a E) \in \mathcal{S}(K)} \\
\text{SYN-OP} \frac{(M, E) \in \mathcal{S}(K)}{(\text{op}(M), \text{op}(E)) \in \mathcal{S}(K)} \quad \text{op} \in \{\text{pub}, \text{priv}, \text{H}\}
\end{array}$$

Table 7.6: Synthesis

Definition 97 (Knowledge).

Knowledge sets $K \in \mathbf{K}$ are finite subsets of $M_\circ \times E$.

The set of names occurring in K is denoted by $n(K)$.

The *synthesis* $\mathcal{S}(K)$ of K is the smallest subset of $M_\circ \times E$ containing K and satisfying the SYN-rules in Table 7.6.

The *analysis* $\mathcal{A}(K)$ of K is $\bigcup_{n \in \mathbb{N}} \mathcal{A}_n(K)$ where the sets $\mathcal{A}_i(K)$ are the smallest sets satisfying the ANA-rules in Table 7.7.

For the same reasons as for hedges in Chapter 4, the analysis is defined in two steps. We have defined a finitely stratified hierarchy $(\mathcal{A}_n(K))_{n \in \mathbb{N}}$. Essentially, the index n of an analysis set $\mathcal{A}_n(K)$ approximates the number of proper deconstruction steps that were needed in order to derive its knowledge items (see the rules ANA-INI, ANA-FST, ANA-SND, and ANA-DEC). In contrast to the standard approach, corresponding to $\mathcal{A}_n(K) \subseteq \mathcal{A}_{n+1}(K)$, here only certain items—not all of them—may be propagated from analysis level n to $n+1$ without proper deconstruction step.

As the following example shows, with the notion of knowledge of this chapter the simple rule $\mathcal{A}_n(K) \subseteq \mathcal{A}_{n+1}(K)$ would allow us to possibly analyse the same message several times, in different ways, which would indeed be harmful. Assume that we remove the rules ANA-DEC-REC and ANA-NAM-REC as well as the indices of analysis sets in Table 7.7 (which amounts to admitting $\mathcal{A}_n(K) \subseteq \mathcal{A}_{n+1}(K)$).

If we now analyse the knowledge set $K = \{(k, k), (\text{Enc}_k^s k, x)\}$ according to this “standard” approach then we would first get the pair $(k, \text{Dec}_k^s x)$,

$$\begin{array}{c}
\text{ANA-INI} \frac{(M, E) \in K}{(M, E) \in \mathcal{A}_0(K)} \\
\text{ANA-FST} \frac{((M.N), E) \in \mathcal{A}_n(K)}{(M, \pi_1(E)) \in \mathcal{A}_{n+1}(K)} \quad \text{ANA-SND} \frac{((M.N), E) \in \mathcal{A}_n(K)}{(N, \pi_2(E)) \in \mathcal{A}_{n+1}(K)} \\
\text{ANA-DEC-S} \frac{(\text{Enc}_N^s M, E) \in \mathcal{A}_n(K) \quad (N, F) \in \mathcal{S}(\mathcal{A}_n(K))}{(M, \text{Dec}_F^s E) \in \mathcal{A}_{n+1}(K)} \\
\text{ANA-DEC-S-REC} \frac{(\text{Enc}_N^s M, E) \in \mathcal{A}_n(K) \quad (N, F) \notin \mathcal{S}(\mathcal{A}_n(K))}{(\text{Enc}_N^s M, E) \in \mathcal{A}_{n+1}(K)} \\
\text{ANA-DEC-A} \frac{(\text{Enc}_N^a M, E) \in \mathcal{A}_n(K) \quad (\text{inv}(N), F) \in \mathcal{S}(\mathcal{A}_n(K))}{(M, \text{Dec}_F^a E) \in \mathcal{A}_{n+1}(K)} \\
\text{ANA-DEC-A-REC} \frac{(\text{Enc}_N^a M, E) \in \mathcal{A}_n(K) \quad (\text{inv}(N), F) \notin \mathcal{S}(\mathcal{A}_n(K))}{(\text{Enc}_N^a M, E) \in \mathcal{A}_{n+1}(K)} \\
\text{ANA-NAM-REC} \frac{(M, E) \in \mathcal{A}_n(K) \quad M \in N_\diamond \cup \mathbf{A}}{(M, E) \in \mathcal{A}_{n+1}(K)} \\
\text{ANA-OP} \frac{(\text{op}(M), E) \in \mathcal{A}_n(K)}{(\text{op}(M), E) \in \mathcal{A}_{n+1}(K)} \text{op} \in \{\text{pub, priv, H}\}
\end{array}$$

Table 7.7: Analysis

then the pair $(k, \text{Dec}_{\text{Dec}_k^s}^s x)$, then $(k, \text{Dec}_{\text{Dec}_{\text{Dec}_k^s}^s}^s x)$, etc. The resulting analysis set $\mathcal{A}(K)$ would be of infinite size, and thus not even be a knowledge set², thus prohibiting a finite representation of the knowledge of participants.

Instead, we control the propagation from analysis level n to $n+1$ by the rules ANA-NAM-REC and ANA-DEC-REC. Knowledge items (M, E) can only be propagated to the next level of the analysis if M is not analysable (i.e., deconstructible) with the knowledge of the same level: either M is a pure name (possibly an agent name) or M can *not* be decrypted with knowledge from the same analysis level. Note that when computing the sequence $(\mathcal{A}_n(K))_{n \in \mathbb{N}}$, the rules ANA-FST, ANA-SND and ANA-DEC strictly decrease the size of the messages, so they can only be applied a finite number of times. Thus, it is obvious that the sequence $(\mathcal{A}_n(K))_{n \in \mathbb{N}}$ converges and thus $\mathcal{A}(K)$ is finite.

Example 22

Consider $K_0 = \{(A, A), (B, B), (S, S), (k_{AS}, k_{AS}), (k_{BS}, k_{BS})\}$.

Let $K = K_0 \cup \{((A \cdot \text{Enc}_{k_{AS}}^s(B \cdot k_{AB})), x_0)\}$.

$$\text{Then } \mathcal{A}(K) = K \cup \left\{ \begin{array}{l} \left(\begin{array}{cc} A & , \quad \pi_1(x_0) \\ \text{Enc}_{k_{AS}}^s(B \cdot k_{AB}) & , \quad \pi_2(x_0) \end{array} \right) \\ \left(\begin{array}{cc} (B \cdot k_{AB}) & , \quad \text{Dec}_{k_{AS}}^s \pi_2(x_0) \\ B & , \quad \pi_1(\text{Dec}_{k_{AS}}^s \pi_2(x_0)) \end{array} \right) \\ \left(\begin{array}{cc} k_{AB} & , \quad \pi_2(\text{Dec}_{k_{AS}}^s \pi_2(x_0)) \end{array} \right) \end{array} \right\}^*$$

7.3.4 Generating checks.

The above knowledge representation allows us to generate the checks required on message reception in a justified manner. Recall that these checks must verify (1) in how far the expectations of the recipient on the received message (as expressed statically in the narration) are matched according to the recipient's current knowledge, and (2) in how far the gained knowledge is consistent with previously acquired knowledge.

Thus, obviously necessary checks are due to the *type* of messages: if an expression shall correspond to a pair then it better allows for projections; if an expression shall correspond to an encrypted message, then it better allows for decryption with the appropriate key—but only if it is known by the receiver.

²In contrast, the “standard” analysis of the corresponding (i.e., projected onto the static component) knowledge set $K_1 = \{k, \text{Enc}_k^s k\}$ yields $\mathcal{A}(K_1) = \{k, \text{Enc}_k^s k\}$.

Less obviously required checks result from the following observation: a message (identifier) M may occur more than once in a protocol narration. Thus, it may happen that, in some knowledge set, M is related to two different expressions E_1 and E_2 , via (M, E_1) and (M, E_2) . As M was precisely used in protocol narrations to indicate the *very same* message, such a knowledge set can only be considered consistent if E_1 and E_2 indeed evaluate to the same message. In the context of asymmetric keys, it can also happen that, in some knowledge set, we find a combination of (M_1, E_1) and (M_2, E_2) where $M_1 = \text{inv}(M_2)$. In this case, $\text{inverse}(E_1, E_2)$ should be satisfied.

Let us assume, as it is customary, that agents dispose of some meaningful initial knowledge (usually of the form (M, M) with M representing some initially known key or participant name). Then, the consistency check for repeated occurrences of data implicitly may take care of testing, e.g., whether some received datum was sent by the expected agent.

To formalise these requirements, we generate consistency formulae.

Definition 98 (Consistency formula).

Let K be a knowledge set. Its *consistency formula* $\Phi(K)$ is defined as follows:

$$\begin{aligned} \Phi(K) := & \bigwedge_{(M,E) \in K} [E:\mathbf{M}] \\ & \wedge \bigwedge_{(M,E_i) \in K \wedge (M,E_j) \in \mathcal{S}(K) \wedge E_i \neq E_j} [E_i = E_j] \\ & \wedge \bigwedge_{(M,E_i) \in K \wedge (\text{inv}(M), E_j) \in \mathcal{S}(K)} \text{inverse}(E_i, E_j) \end{aligned}$$

The first conjunction clause checks that all expressions can be evaluated. The second conjunction clause checks that if there are several ways to build a particular message M , then all the corresponding expressions evaluate to the same entity. (Note that the omission of the subclause $E_i \neq E_j$ would make the first clause redundant; we just kept it for clarity of the respective concepts.) The third conjunction clause checks that if it was possible to generate a message M and its inverse $\text{inv}(M)$, then the corresponding expressions must also be mutually inverse.

When generating the above consistency formula, we compare pairs taken from K with pairs taken from $\mathcal{S}(K)$. The following example shows why it does not suffice to compare just the pairs in K . On the other hand, we should not compare any possible combination of pairs taken from $\mathcal{S}(K)$, because this would yield an infinite formula.

Example 23

If $K = \{(m, x), (H(m), y)\}$, we have that

$$\Phi(K) = [x:\mathbf{M}] \wedge [y:\mathbf{M}] \wedge [H(x) = y]$$

Observe that, if the consistency formula did not consider pairs taken from $\mathcal{S}(K)$, then the test $[H(x) = y]$ would not be present. *

7.3.5 Reducing knowledge sets.

Knowledge sets can often be simplified without loss of information by reducing complex elements to their parts. In our case, we can further simplify due to the occurrence of duplicated elements; there is no loss of information once the consistency formula of Definition 98 remembers the duplication.

Definition 99 (Irreducibles).

Let K be a knowledge set.

The set of *irreducibles* $\mathcal{I}(K)$ is defined by

$$\mathcal{I}(K) := \text{irr}(\mathcal{A}(K))$$

where

$$\begin{aligned} \text{irr}(K) := & \left\{ (M, E) \in K \mid M \in \mathbf{N}_\diamond \cup \mathbf{A} \right\} \\ \cup & \left\{ (\text{Enc}_N^s M, E) \in K \mid \begin{array}{l} \forall F_1 : (M, F_1) \notin \mathcal{S}(K) \\ \vee \forall F_2 : (N, F_2) \notin \mathcal{S}(K) \end{array} \right\} \\ \cup & \left\{ (\text{Enc}_N^a M, E) \in K \mid \begin{array}{l} \forall F_1 : (M, F_1) \notin \mathcal{S}(K) \\ \vee \forall F_2 : (N, F_2) \notin \mathcal{S}(K) \end{array} \right\} \\ \cup & \left\{ ((M.N), E) \in K \mid \begin{array}{l} \forall F_1 : (M, F_1) \notin \mathcal{S}(K) \\ \vee \forall F_2 : (N, F_2) \notin \mathcal{S}(K) \end{array} \right\} \\ \cup & \left\{ (\text{op}(M), E) \in K \mid \forall F : (M, F) \notin \mathcal{S}(K) \right\} . \end{aligned}$$

Let \sim denote the equivalence relation on $\mathbf{M}_\diamond \times \mathbf{E}$ induced by

$$(M, E) \sim (N, F) \iff M = N$$

We let $\text{rep}(K)$ denote the result of deterministically selecting³ one representative element for each equivalence class induced by \sim on K .

Example 24

We continue Example 22. We have:

$$\begin{aligned} \mathcal{I}(K) &= K_0 \cup \left\{ (A, \pi_1(x_0)), (B, \pi_1(\text{Dec}_{k_{AS}}^s \pi_2(x_0))), \right. \\ & \quad \left. (k_{AB}, \pi_2(\text{Dec}_{k_{AS}}^s \pi_2(x_0))) \right\} \\ \text{rep}(\mathcal{I}(K)) &= K_0 \cup \left\{ (k_{AB}, \pi_2(\text{Dec}_{k_{AS}}^s \pi_2(x_0))) \right\} \end{aligned}$$

³Choose an arbitrary well-founded total order on expressions and select the smallest expression according to this order.

Here, we assume that the function $\text{rep}(\cdot)$ selected (A, A) instead of $(A, \pi_1(x_0))$, and (B, B) instead of $(B, \pi_1(\text{Dec}_{k_{AS}} \pi_2(x_0)))$.

Moreover, we have $(M, E) \in \mathcal{S}(\text{rep}(\mathcal{I}(K)))$ where

$$\begin{aligned} M &:= \text{Enc}_{k_{BS}}^s(A \cdot (B \cdot k_{AB})) \\ E &:= \text{Enc}_{k_{BS}}^s(A \cdot (B \cdot \pi_2(\text{Dec}_{k_{AS}}^s \pi_2(x_0)))) \quad * \end{aligned}$$

7.3.6 The compilation.

We now have set up all the required ingredients to compile an extended protocol narration into an executable protocol narration. Technically, while traversing the syntax of a given narration, the translation function keeps a record of global information on the used variables and hidden names, as well as local (i.e., participant-dependent) information on their knowledge on generated names.

Definition 100 (Compilation).

The translation $\mathcal{X}[\cdot]^{(v, \omega, \kappa, \nu)} : \mathcal{D} \rightarrow \mathcal{X}$ is defined inductively in Table 7.8, where $v \subset V$ (current set of used variables), $\omega \subset N_\diamond$ (current set of private names), $\kappa : A \rightarrow K$ (partial mapping from agents to their current knowledge), and $\nu : A \rightarrow N_\diamond$ (partial mapping from agents to their current set of generated names).

Let $P \in \mathcal{D}$ be a protocol narration. Let A_P denote the set of agent names appearing in P . Then, $\mathcal{X}[[P]]^{(\mathcal{O}, \mathcal{O}, \kappa_P, \mathcal{O})}$ denotes the *compilation* of P , where the *initial knowledge* κ_P is defined by $\kappa_P(A) := \{(B, B) \mid B \in A_P\}$ for all $A \in A_P$.

We say that P is *well-formed* if its compilation is defined.

For simplicity, the compilation assumes that all agents initially know each other, as expressed in the initial knowledge set κ_P . Checks-on-reception are deduced from the individual knowledge set of a receiver. To avoid to perform the same checks again and again, the compilation keeps the knowledge sets of κ in reduced form, i.e., $\kappa(A) = \text{rep}(\mathcal{I}(\kappa(A)))$. To update $f \in \{\kappa, \nu\}$, we note $f[x \leftarrow y]$ with $f[x \leftarrow y](x) = y$ and $f[x \leftarrow y](z) = f(z)$ for $z \neq x$.

The compilation of **private** k and A **generates** n checks in both cases that the local (or generated) name is fresh, but differs with respect to the addition of the fresh name to the knowledge sets of agents: whereas the construction A **generates** n increases the knowledge of A , the name k of

$$\begin{aligned}
\mathcal{X}[\epsilon]^{(v,\omega,\kappa,\nu)} &:= \epsilon \\
\mathcal{X}[A \text{ knows } M ; P]^{(v,\omega,\kappa,\nu)} &:= \mathcal{X}[P]^{(v,\omega,\kappa',\nu)} \\
&\quad \text{if } \mathbf{n}(M) \cap \bigcup_{A \in A} \nu(A) = \emptyset \\
&\quad \text{where } K'_A := \kappa(A) \cup \{(M, M)\} \\
&\quad \text{and } \kappa' := \kappa[A \leftarrow \text{rep}(\mathcal{I}(K'_A))] \\
\mathcal{X}[\text{private } k ; P]^{(v,\omega,\kappa,\nu)} &:= \nu k ; \mathcal{X}[P]^{(v,\omega \cup \{k\}, \kappa, \nu)} \\
&\quad \text{if } k \notin \omega \cup \bigcup_{A \in A} (\mathbf{n}(\kappa(A)) \cup \nu(A)) \\
\mathcal{X}[A \text{ generates } n ; P]^{(v,\omega,\kappa,\nu)} &:= \nu n ; \mathcal{X}[P]^{(v,\omega,\kappa',\nu')} \\
&\quad \text{if } n \notin \omega \cup \bigcup_{A \in A} (\mathbf{n}(\kappa(A)) \cup \nu(A)) \\
&\quad \text{where } K'_A := \kappa(A) \cup \{(n, n)\} \\
&\quad \text{and } \kappa' := \kappa[A \leftarrow \text{rep}(\mathcal{I}(K'_A))] \\
&\quad \text{and } \nu' := \nu[A \leftarrow \nu(A) \cup \{n\}] \\
\mathcal{X}[A \rightsquigarrow B : M ; P]^{(v,\omega,\kappa,\nu)} &:= A:B!E ; \\
&\quad B:?x ; B:\phi ; \\
&\quad \mathcal{X}[P]^{(v \cup \{x\}, \omega, \kappa', \nu)} \\
&\quad \text{if } A \neq B \text{ and } (M, E) \in \mathcal{S}(\kappa(A)) \\
&\quad \text{where } x := \min(\mathbf{V} \setminus \nu) \\
&\quad \text{and } K'_B := \kappa(B) \cup \{(M, x)\} \\
&\quad \text{and } \kappa' := \kappa[B \leftarrow \text{rep}(\mathcal{I}(K'_B))] \\
&\quad \text{and } \phi := \Phi(\mathcal{A}(K'_B))
\end{aligned}$$

Table 7.8: Definition of $\mathcal{X}[\cdot]$.

private k is not added to any knowledge; this task is deferred to explicit A **knows** k clauses for the intended A .

The compilation of $A \rightsquigarrow B : M$ checks that M can be synthesised by A , picks a new variable x and adds the pair (M, x) to the knowledge of B .⁴ The consistency formula $\Phi(\mathcal{A}(K'_B))$ of the analysis of this updated knowledge K'_B defines the checks ϕ to be performed by B at runtime. Note that this must be done on the non-reduced version. In fact, it is precisely the consistency check that allows us then to continue with the knowledge in reduced form.

Finally, note that our concept of *well-formedness* of a protocol narration corresponds to the notions of *executability* in [54].

Example 25

Let WMF be the Wide-Mouthed Frog protocol presented Table 7.3.

$$\begin{aligned} \text{We have } \kappa_{\text{WMF}} : A &\rightarrow \mathbf{K} \\ A &\mapsto \{(A, A), (B, B), (S, S)\} \\ B &\mapsto \{(A, A), (B, B), (S, S)\} \\ S &\mapsto \{(A, A), (B, B), (S, S)\} \end{aligned}$$

WMF is well-formed and its compilation is

$$\begin{aligned} \mathcal{X}[\text{WMF}]^{(\mathcal{O}, \mathcal{O}, \kappa_{\text{WMF}}, \mathcal{O})} = & \\ & vk_{AS} ; vk_{BS} ; vk_{AB} ; \\ & A:S!(A . \text{Enc}_{k_{AS}}^s(B . k_{AB})) ; \\ & S:?x_0 ; S:\phi_0 ; \\ & S:B!\text{Enc}_{k_{BS}}^s(A . (B . \pi_2 (\text{Dec}_{k_{AS}}^s \pi_2(x_0)))) ; \\ & B:?x_1 ; B:\phi_1 ; \\ & A:B!\text{Enc}_{k_{AB}}^s m ; \\ & B:?x_2 ; B:\phi_2 \end{aligned}$$

where ϕ_0 , ϕ_1 and ϕ_2 are given below.

$$\begin{aligned} \phi_0 &\approx [A = \pi_1(x_0)] \wedge [B = \pi_1(\text{Dec}_{k_{AS}} \pi_2(x_0))] \\ \phi_1 &\approx [A = \pi_1(\text{Dec}_{k_{BS}} x_1)] \wedge [B = \pi_1(\pi_2(\text{Dec}_{k_{BS}} x_1))] \\ \phi_2 &\approx [\text{Dec}_{\pi_2}(\pi_2(\text{Dec}_{k_{BS}} x_1)) x_2 : M] \end{aligned}$$

We refer the reader to Section 7.7.2 where this protocol is studied with our tool. *

⁴Usually, narrations are defined such that the sender A is supposed to statically know the precise name B of the intended receiver. In a dynamic scenario, the compilation would need to check that B is synthesisable by A .

7.4 A detailed example: the ASW protocol

7.4.1 The protocol

The ASW protocol is an optimistic fair-exchange protocol for contract signing, proposed by Asokan, Shoup and Waidner in [15]. Figure 7.1 shows the slightly simplified version of the Exchange Subprotocol of ASW (that we will simply refer afterwards as the ASW protocol) that Caleiro, Viganò and Basin have used in [55] to illustrate that a *direct interpretation* of protocol narrations would be too naive. A direct interpretation simply lists all the *external* actions each participant should commit, but does not explicit the internal checks and does not verify that these external actions are actually feasible. This protocol also shows that it is not sufficient to check received messages just once, immediately after their reception, because the receiving participant might only later on gain further knowledge that would enable it to analyse the structure of the just-received message more deeply.

The goal of the ASW protocol is to establish a valid contract between the two participants A and B . The protocol proceeds in two rounds.

First, the two participants send their respective so-called *public commitments* $H(n_1)$ or $H(n_2)$ with the contract text m they have agreed upon; n_1 and n_2 being nonces generated by the two participants and called their respective *secret commitments* to the contract. For this first round, the respective messages are digitally signed with the participants' private keys. As usual, the signature can be verified by using the corresponding public key.

Then, in the second round, the participants exchange their respective secret commitments so that they can check the public commitment they

$$\begin{aligned}
 &A \text{ knows } m ; A \text{ knows } k_A ; A \text{ knows pub}(k_B) \\
 &B \text{ knows } m ; B \text{ knows } k_B ; B \text{ knows pub}(k_A) ; \\
 &A \text{ generates } n_1 ; \\
 &B \text{ generates } n_2 ; \\
 &A \rightsquigarrow B : \text{Enc}_{\text{priv}(k_A)}^a((\text{pub}(k_A) \cdot \text{pub}(k_B)) \cdot (m \cdot H(n_1))) \\
 &B \rightsquigarrow A : \text{Enc}_{\text{priv}(k_B)}^a(\text{Enc}_{\text{priv}(k_A)}^a((\text{pub}(k_A) \cdot \text{pub}(k_B)) \cdot (m \cdot H(n_1)))) \cdot H(n_2)) \\
 &A \rightsquigarrow B : n_1 \\
 &B \rightsquigarrow A : n_2
 \end{aligned}$$

Figure 7.1: The Exchange Subprotocol of the ASW protocol (simplified)

have received in round one by hashing this value.

At the end of this exchange, both participants have a valid contract of the form indicated in Figure 7.2.

$$\begin{array}{c} \text{Enc}_{\text{priv}(k_A)}^a((\text{pub}(k_A) \cdot \text{pub}(k_B)) \cdot (m \cdot H(n_1))) \\ \text{Enc}_{\text{priv}(k_B)}^a(\text{Enc}_{\text{priv}(k_A)}^a((\text{pub}(k_A) \cdot \text{pub}(k_B)) \cdot (m \cdot H(n_1)))) \cdot H(n_2)) \\ n_1 \\ n_2 \end{array}$$

Figure 7.2: Contract form at the end of the exchange

In this protocol, the participants should in some sense backtrack their analysis once they have received the message of the second round. Indeed, when B first receives $H(n_1)$, it cannot check that this corresponds to the hashing of the nonce n_1 since n_1 is not yet part of B 's knowledge. However, once B receives n_1 in the second round, it is able to check that the hashing of n_1 is effectively equal to the message that it has supposed to be $H(n_1)$ in the first round; this check should occur before B sends its own public commitment to A .

7.4.2 Compilation of the ASW protocol

We now study this protocol in our setting. We first define some shortcuts:

$$\begin{array}{l} M_1 := \text{Enc}_{\text{priv}(k_A)}^a((\text{pub}(k_A) \cdot \text{pub}(k_B)) \cdot (m \cdot H(n_1))) \\ M_2 := \text{Enc}_{\text{priv}(k_B)}^a(\text{Enc}_{\text{priv}(k_A)}^a((\text{pub}(k_A) \cdot \text{pub}(k_B)) \cdot (m \cdot H(n_1)))) \cdot H(n_2)) \end{array}$$

Computing the initial knowledge The initial knowledge set of participants A and B is, by definition,

$$\begin{array}{l} \kappa_0^A := \{(A, A), (B, B)\} \\ \kappa_0^B := \{(A, A), (B, B)\} \end{array}$$

Compilation of the declarations When compiling the declarations, the compilation process checks that n_1 and n_2 are distinct names not used in

the other pieces of information declared to be known at the beginning of the protocol. After computation, we obtain the following knowledge sets

$$k_1^A := \{(A, A), (B, B), (m, m), (n_1, n_1), (k_A, k_A), (\text{pub}(k_B), \text{pub}(k_B))\}$$

$$k_1^B := \{(A, A), (B, B), (m, m), (n_2, n_2), (k_B, k_B), (\text{pub}(k_A), \text{pub}(k_A))\}$$

The line 7.1 of Figure 7.3 corresponds to the compilation of the declarations.

$$\nu n_1 ; \nu n_2 ; \tag{7.1}$$

$$A:B!E_1 ; B:?x_1 ; B:\phi_1 ; \tag{7.2}$$

$$B:A!E_2 ; A:?x_2 ; A:\phi_2 ; \tag{7.3}$$

$$A:B!E_3 ; B:?x_3 ; B:\phi_3 ; \tag{7.4}$$

$$B:A!E_4 ; A:?x_4 ; A:\phi_4 \tag{7.5}$$

Figure 7.3: Executable narration compiled from ASW protocol

First message When compiling the first message exchange, the compilation process

1. checks that A can synthesise message M_1 by looking for an expression E_1 such that $(M_1, E_1) \in \mathcal{S}(k_1^A)$. Here, the unique candidate is expression $E_1 := \text{Enc}_{\text{priv}(k_A)}^a((\text{pub}(k_A) \cdot \text{pub}(k_B)) \cdot (m \cdot H(n_1)))$.
2. takes a new variable x_1 to be bound to the message that participant B will receive; according to the statically defined information contained in the narration, this message is expected to be M_1 .
3. computes the consistency formula ϕ_1 of the analysis of the knowledge set resulting from the addition of (M_1, x_1) to k_1^B , and computes a reduced form of $k_1^B \cup \{(M_1, x_1)\}$.

Here we have

$$\begin{aligned}
& \mathcal{A}(k_1^B \cup \{(M_1, x_1)\}) \\
& = k_1^B \\
& \cup \{(M_1, x_1)\} \\
& \cup \left\{ \left((\text{pub}(k_A) \cdot \text{pub}(k_B)) \cdot (m \cdot H(n_1)), \text{Dec}_{\text{pub}(k_A)}^a x_1 \right) \right\} \\
& \cup \left\{ \left((\text{pub}(k_A) \cdot \text{pub}(k_B)), \pi_1 \left(\text{Dec}_{\text{pub}(k_A)}^a x_1 \right) \right) \right\} \\
& \cup \left\{ \left((m \cdot H(n_1)), \pi_2 \left(\text{Dec}_{\text{pub}(k_A)}^a x_1 \right) \right) \right\} \\
& \cup \left\{ \left(\text{pub}(k_A), \pi_1 \left(\pi_1 \left(\text{Dec}_{\text{pub}(k_A)}^a x_1 \right) \right) \right) \right\} \\
& \cup \left\{ \left(\text{pub}(k_B), \pi_2 \left(\pi_1 \left(\text{Dec}_{\text{pub}(k_A)}^a x_1 \right) \right) \right) \right\} \\
& \cup \left\{ \left(m, \pi_1 \left(\pi_2 \left(\text{Dec}_{\text{pub}(k_A)}^a x_1 \right) \right) \right) \right\} \\
& \cup \left\{ \left(H(n_1), \pi_2 \left(\pi_2 \left(\text{Dec}_{\text{pub}(k_A)}^a x_1 \right) \right) \right) \right\}
\end{aligned}$$

After some simplifications (see 7.7.1), the consistency formula appears to be equivalent to

$$\begin{aligned}
\phi_1 & := [\text{pub}(k_B) = \pi_2 \left(\pi_1 \left(\text{Dec}_{\text{pub}(k_A)}^a x_1 \right) \right)] \\
& \wedge [\text{pub}(k_A) = \pi_1 \left(\pi_1 \left(\text{Dec}_{\text{pub}(k_A)}^a x_1 \right) \right)] \\
& \wedge [m = \pi_1 \left(\pi_2 \left(\text{Dec}_{\text{pub}(k_A)}^a x_1 \right) \right)]
\end{aligned}$$

And a possible candidate for $\text{rep}(\mathcal{I}(k_1^B \cup (M_1, x_1)))$ is

$$\begin{aligned}
k_2^B & := k_1^B \\
& \cup \{(M_1, x_1)\} \\
& \cup \left\{ \left(H(n_1), \pi_2 \left(\pi_2 \left(\text{Dec}_{\text{pub}(k_A)}^a x_1 \right) \right) \right) \right\}
\end{aligned}$$

Note that (M_1, x_1) is not removed because B has no way to digitally sign a message with the *private* key of A .

4. generates line 7.2 of Figure 7.3.

Second message The compilation of the second message exchange is similar to the first message but with role of A and B swapped. So the compilation process

1. checks that B can synthesise the message M_2 by looking for an expression E_2 such that $(M_2, E_2) \in \mathcal{S}(k_B^B)$.

The candidate is expression $E_2 := \text{Enc}_{\text{priv}(k_B)}^a(x_1 \cdot H(n_2))$.

2. takes a new variable x_2 to be bound to the message that participant A will receive; according to the statically defined information contained in the narration, this message is expected to be M_2 .
3. computes the consistency formula ϕ_2 of the analysis of the knowledge set resulting of the addition of (M_2, x_2) to k_1^A and computes a reduced form of $k_1^A \cup \{(M_2, x_2)\}$.

Here we have

$$\begin{aligned}
& \mathcal{A}(k_1^A \cup \{(M_2, x_2)\}) \\
&= k_1^A \\
& \cup \{(M_2, x_2)\} \\
& \cup \{((M_1 \cdot H(n_2)), \text{Dec}_{\text{pub}(k_B)}^a x_2)\} \\
& \cup \{(M_1, \pi_1(\text{Dec}_{\text{pub}(k_B)}^a x_2))\} \\
& \cup \{(H(n_2), \pi_2(\text{Dec}_{\text{pub}(k_B)}^a x_2))\} \\
& \cup \left\{ \left((\text{pub}(k_A) \cdot \text{pub}(k_B)) \cdot (m \cdot H(n_1)), \right. \right. \\
& \quad \left. \left. \text{Dec}_{\text{pub}(k_A)}^a \pi_1(\text{Dec}_{\text{pub}(k_B)}^a x_2) \right) \right\} \\
& \cup \left\{ ((\text{pub}(k_A) \cdot \text{pub}(k_B)), \pi_1(\text{Dec}_{\text{pub}(k_A)}^a \pi_1(\text{Dec}_{\text{pub}(k_B)}^a x_2))) \right\} \\
& \cup \left\{ (m \cdot H(n_1), \pi_2(\text{Dec}_{\text{pub}(k_A)}^a \pi_1(\text{Dec}_{\text{pub}(k_B)}^a x_2))) \right\} \\
& \cup \left\{ (\text{pub}(k_A), \pi_1(\pi_1(\text{Dec}_{\text{pub}(k_A)}^a \pi_1(\text{Dec}_{\text{pub}(k_B)}^a x_2)))) \right\} \\
& \cup \left\{ (\text{pub}(k_B), \pi_2(\pi_1(\text{Dec}_{\text{pub}(k_A)}^a \pi_1(\text{Dec}_{\text{pub}(k_B)}^a x_2)))) \right\} \\
& \cup \left\{ (m, \pi_1(\pi_2(\text{Dec}_{\text{pub}(k_A)}^a \pi_1(\text{Dec}_{\text{pub}(k_B)}^a x_2)))) \right\} \\
& \cup \left\{ (H(n_1), \pi_2(\pi_2(\text{Dec}_{\text{pub}(k_A)}^a \pi_1(\text{Dec}_{\text{pub}(k_B)}^a x_2)))) \right\}
\end{aligned}$$

After some simplifications, the consistency formula appears to be equivalent to

$$\phi_2 := [\pi_1(\text{Dec}_{\text{pub}(k_B)}^a x_2) = M_1]$$

And a possible candidate for $\text{rep}(\mathcal{I}(k_1^A \cup (M_2, x_2)))$ is

$$\begin{aligned} k_2^A &:= k_1^A \\ &\cup \{(M_2, x_2)\} \\ &\cup \left\{ (H(n_2), \pi_2 \left(\text{Dec}_{\text{pub}(k_B)}^a x_2 \right)) \right\} \end{aligned}$$

4. generates line 7.3 of Figure 7.3.

Third message For the third message, the compilation process

1. checks that n_1 is synthesisable by A . The expression n_1 is a candidate to build the message n_1 (the pair (n_1, n_1) has been added to the knowledge set of A during the compilation of the declarations).
2. takes a new variable x_3 to be bound to the message that participant B will receive; according to the statically defined information contained in the narration, this message is expected to be n_1 .
3. computes the consistency formula ϕ_3 of the analysis of the knowledge set resulting of the addition of (n_1, x_3) to k_2^B and computes a reduced form of $k_2^B \cup \{(n_1, x_3)\}$

Here we have

$$\begin{aligned} \mathcal{A}(k_2^B \cup \{(n_1, x_3)\}) &= k_2^B \\ &\cup \{(n_1, x_3)\} \end{aligned}$$

After some simplifications (taking into account that at this point of the executable narration, the formula ϕ_1 should have been satisfied), the consistency formula appears to be equivalent to

$$\phi_3 := [H(x_3) = \pi_2 \left(\pi_2 \left(\text{Dec}_{\text{pub}(k_A)}^a x_1 \right) \right)]$$

And a possible candidate for $\text{rep}(\mathcal{I}(k_2^B \cup (n_1, x_3)))$ is

$$\begin{aligned} k_3^B &:= (k_2^B \setminus \left\{ (H(n_1), \pi_2 \left(\pi_2 \left(\text{Dec}_{\text{pub}(k_A)}^a x_1 \right) \right)) \right\}) \\ &\cup \{(n_1, x_3)\} \end{aligned}$$

Note that the pair $(H(n_1), \pi_2 \left(\pi_2 \left(\text{Dec}_{\text{pub}(k_A)}^a x_1 \right) \right))$ has been removed from the knowledge set of B because now B knows n_1 and thus can synthesise himself $H(n_1)$.

4. generates line 7.4 of Figure 7.3.

Fourth message Finally, for the fourth message, the compilation process

1. checks that n_2 is synthesisable by B . The expression n_2 is a candidate to build the message n_2 .
2. takes a new variable x_4 to be bound to the message that participant a will receive; according to the statically defined information contained in the narration, this message is expected to be n_2 .
3. computes the consistency formula ϕ_4 of the analysis of the knowledge set resulting of the addition of (n_2, x_4) to k_2^A and computes a reduced form of $k_2^A \cup \{(n_2, x_4)\}$

Here we have

$$\mathcal{A}(k_2^A \cup \{(n_2, x_4)\}) = k_2^A \cup \{(n_2, x_4)\}$$

After some simplifications (taking into account that at this point of the executable narration, the formula ϕ_2 should have been satisfied), the consistency formula appears to be equivalent to

$$\phi_4 := [H(x_4) = \pi_2 \left(\text{Dec}_{\text{pub}(k_B)}^a x_2 \right)]$$

And a possible candidate for $\text{rep}(\mathcal{I}(k_2^A \cup (n_2, x_4)))$ is

$$k_3^A := (k_2^A \setminus \{ (H(n_2), \pi_2 \left(\text{Dec}_{\text{pub}(k_B)}^a x_2 \right)) \}) \cup \{(n_2, x_4)\}$$

Note that the pair $(H(n_2), \pi_2 \left(\text{Dec}_{\text{pub}(k_B)}^a x_2 \right))$ has been removed from the knowledge set of A because now A knows n_2 and thus is able to synthesise himself $H(n_2)$.

4. generates line 7.5 of Figure 7.3.

7.4.3 The ASW protocol and the pattern-matching spi calculus

If we just look at participant B , the spi calculus term we can derive from the executable narration of Figure 7.3 is (see also Section 7.6)

$$\begin{aligned}
& (vn_2)B(x_1). \\
& \quad [\text{pub}(k_A) = \pi_1 \left(\pi_1 \left(\text{Dec}_{\text{pub}(k_A)}^a x_1 \right) \right)] \\
& \quad \wedge [\text{pub}(k_B) = \pi_2 \left(\pi_1 \left(\text{Dec}_{\text{pub}(k_A)}^a x_1 \right) \right)] \\
& \quad \wedge [m = \pi_1 \left(\pi_2 \left(\text{Dec}_{\text{pub}(k_A)}^a x_1 \right) \right)] \\
& \quad \overline{A} \langle \text{Enc}_{\text{priv}(k_B)}^a (x_1 \cdot H(n_2)) \rangle. \\
& \quad B(x_3). \\
& \quad [H(x_3) = \pi_2 \left(\pi_2 \left(\text{Dec}_{\text{pub}(k_A)}^a x_1 \right) \right)] \\
& \quad \overline{A} \langle n_2 \rangle. \mathbf{0}
\end{aligned}$$

But it is not clear to us how such a process can be expressed in the pattern-matching spi calculus in the spirit of what is defined in [78].

A possible term in the pattern-matching spi calculus would be

$$\begin{aligned}
& \text{new } n_2; \\
& \text{inp } B \left\{ x \bullet \text{Enc}_{\text{priv}(k_A)}^a ((\text{pub}(k_A) \cdot \text{pub}(k_B)) \cdot (m \cdot H(x))) \right\}; \\
& \text{out } A \text{Enc}_{\text{priv}(k_B)}^a (\text{Enc}_{\text{priv}(k_A)}^a ((\text{pub}(k_A) \cdot \text{pub}(k_B)) \cdot (m \cdot H(x_1))) \cdot H(n_2)); \\
& \text{inp } B \left\{ \bullet x \right\}; \\
& \text{out } A n_2; \mathbf{0}
\end{aligned}$$

But unfortunately, the first pattern is not *implementable* in the sense of [78].

Indeed, being able to write $\{x \bullet \dots H(x) \dots\}$ in a pattern position would intuitively mean that it is possible to inverse the supposed one-way function $H(\cdot) : M_\diamond \rightarrow M_\diamond$ and thus get a value x from its hashing $H(x)$.

7.5 Executing protocol narrations

In this section, we propose an operational semantics for narrations. It proceeds in a traditional syntax-directed manner by analysing the current top-level construct in order to see what to execute next. Since narrations contain some implicit concurrency among principals, we introduce a structural reordering relation to shuffle concurrently enabled actions to the top level. The actual execution of steps further needs to take care of the evaluation of messages to be sent, and also to prevent from name clashes that are possible due to the presence of binders.

7.5.1 Binders and α -conversion.

Our language of executable narrations contains two sort of binders: one for names and one for variables.

The first binder is introduced by the construction vn . If $X = vn ; X'$, then n is bound in X (i.e. the free occurrences of n in X' refers to this binder). As the identity of n is not important, we identify X with $vn' ; X'\{n'/n\}$ where n' is a name that is not free in X and $X'\{n'/n\}$ is X' where all the free occurrences of n has been replaced with n' . X and $vn' ; X'\{n'/n\}$ are called α -equivalent. In the following, we identify α -equivalent executable narrations. Now, for an executable narration X , we can define the usual *bound names* $\text{bn}(X)$, *free names* $\text{fn}(X)$ of X and, moreover, if $n, n' \in N_\circ$, $X\{n'/n\}$, the substitution of n' for n in X .

The second binder is the one introduced by the construction $A:?x$. If $X = A:?x ; X'$, then x is bound in the actions of X' concerning A : indeed, if further in the executable narration, B refers to x , the x is not the same as the one used by A . Since variables will typically be substituted with messages, we do not need α -conversion on variables but we need to define a new kind of *local substitution*: if X is an executable narration, $x \in V$, $M \in M_\circ$ with $\text{n}(M) \cap \text{bn}(X) = \emptyset$ (which can be assured by choosing a suitable α -equivalent version of X), and $A \in A$, we define in Table 7.9 the substitution $X\{M/x\}_{@A}$ of M for x in X on A .

$$\begin{aligned}
 \epsilon\{M/x\}_{@A} &:= \epsilon \\
 (A':B!E ; X)\{M/x\}_{@A} &:= \begin{cases} A':B!E ; X\{M/x\}_{@A} & \text{if } A' \neq A \\ A':B!E\{M/x\} ; X\{M/x\}_{@A} & \text{otherwise} \end{cases} \\
 (A':?y ; X)\{M/x\}_{@A} &:= \begin{cases} A':?y ; X\{M/x\}_{@A} & \text{if } A' \neq A \\ A':?y ; X\{M/x\}_{@A} & \text{if } A = A' \text{ and } y \neq x \\ A':?x ; X & \text{otherwise} \end{cases} \\
 (A':\phi ; X)\{M/x\}_{@A} &:= \begin{cases} A':\phi ; X\{M/x\}_{@A} & \text{if } A' \neq A \\ A':\phi\{M/x\} ; X\{M/x\}_{@A} & \text{otherwise} \end{cases} \\
 (vn ; X)\{M/x\}_{@A} &:= vn ; X\{M/x\}_{@A}
 \end{aligned}$$

Table 7.9: Substitution

7.5.2 Reordering.

Protocol narrations are sequences of actions. However, the sequential character is not always causally motivated. Instead, the order of two consecutive actions carried out by *different* principals can always be swapped, because—after our split of message exchanges in the compilation process of Section 7.3—they are *independent*. The same holds for the consecutive occurrence of an action and a scope, unless the scope’s name occurs in the action. Formally, we manifest the swapping of independent actions in a structural congruence relation.

Definition 101.

The *reordering* $\cong \subseteq X \times X$ is the least equivalence relation satisfying the rules given in Table 7.10, and closed under contexts of the form $X; [\cdot]; X'$.

We define \cong_α to be the union of \cong and α -equivalence.

$$\begin{array}{c}
 \cong\text{-S-S} \frac{A \neq C}{A:B!E ; C:D!F \cong C:D!F ; A:B!E} \\
 \\
 \cong\text{-S-C} \frac{A \neq C}{A:B!E ; C:\phi \cong C:\phi ; A:B!E} \qquad \cong\text{-S-R} \frac{A \neq C}{A:B!E ; C:?x \cong C:?x ; A:B!E} \\
 \\
 \cong\text{-R-C} \frac{A \neq C}{A:?x ; C:\phi \cong C:\phi ; A:?x} \qquad \cong\text{-R-R} \frac{A \neq C}{A:?x ; C:?y \cong C:?y ; A:?x} \\
 \\
 \cong\text{-C-C} \frac{A \neq C}{A:\phi ; C:\psi \cong C:\psi ; A:\phi} \qquad \cong\text{-S-N} \frac{n \notin \mathfrak{n}(E)}{A:B!E ; vn \cong vn ; A:B!E} \\
 \\
 \cong\text{-C-N} \frac{n \notin \mathfrak{n}(\phi)}{A:\phi ; vn \cong vn ; A:\phi} \qquad \cong\text{-R-N} \frac{}{A:?x ; vn \cong vn ; A:?x} \\
 \\
 \cong\text{-N-N} \frac{}{vn ; vm \cong vm ; vn}
 \end{array}$$

Table 7.10: Reordering

Given a particular message exchange $A \rightsquigarrow B : M$, it may possibly seem surprising at first that the reordering relation allows the respective reception action $B:?x$ to occur *before* its associated emission action $A:B!M$. Clearly, the received message cannot be the intended one. Such a behaviour must

be dealt with carefully, e.g., by rejecting unintended messages, but its existence cannot be avoided; it is a matter of fact that concurrent systems exchange messages asynchronously.

7.5.3 Labelled transitions.

We define a straightforward labelled semantics of executable narrations, in style influenced by spi calculus, in Table 7.11.

$$\begin{array}{c}
\text{SEND} \frac{\llbracket E \rrbracket = M \in \mathbf{M}_\diamond}{A:B!E ; X \xrightarrow{A:B!M} X} \quad \text{RECEIVE} \frac{}{A:?x ; X \xrightarrow{A:?M} X\{M/x\}_{@A}} M \in \mathbf{M}_\diamond \\
\\
\text{CHECK} \frac{X \xrightarrow{A:\beta} X'}{A:\phi ; X \xrightarrow{A:\beta} X'} \llbracket \phi \rrbracket = \mathbf{true} \\
\\
\text{OPEN} \frac{X \xrightarrow{A:(v\tilde{n})B!M} X'}{vz ; X \xrightarrow{A:(vz\tilde{n})B!M} X'} z \in \mathbf{n}(M) \setminus \{\tilde{n}\} \\
\\
\text{RES} \frac{X \xrightarrow{A:\beta} X'}{vz ; X \xrightarrow{A:\beta} vz ; X'} z \notin \mathbf{fn}(\beta) \quad \text{REARRANGE} \frac{X \cong_\alpha X' \quad X' \xrightarrow{A:\beta} X''}{X \xrightarrow{A:\beta} X''}
\end{array}$$

Table 7.11: Labelled semantics of executable narrations

Our semantics relates two executable narrations with a transition $\xrightarrow{A:\beta}$ where $A \in \mathbf{A}$ and β is either an input action $?M$ where $M \in \mathbf{M}_\diamond$ or a bound output action $(v\tilde{n})B!M$ where \tilde{n} is a (possibly empty) list of pairwise distinct names $n_1 \cdots n_k$ (that are bound in the remainder), $B \in \mathbf{A}$ and $M \in \mathbf{M}_\diamond$. If $k = 0$ (i.e. \tilde{n} is empty), we will simply write $B!M$. Note that there is no internal action in our formal semantics of narrations. We might also have introduced a rule

$$\text{COM} \frac{X \xrightarrow{A:(v\tilde{n})B!M} X' \quad X' \xrightarrow{B:?M} X''}{X \xrightarrow{\tau} v\tilde{n} ; X''}$$

but we tend to insist on the fact that every communication necessarily passes through the network, while such a rule COM would allow to avoid this.

7.6 Rewriting protocol narrations ... into spi calculus

As the reader might have noticed, the executable narrations as of Section 7.3 and the spi calculus of Chapter 3 are similar. Thus, we may now provide a straightforward translation of executable narrations into the spi calculus and easily show that the semantics is preserved. The main idea is that the implicit concurrency structure of narrations as encoded with explicit agent names is projected out ($X \upharpoonright_A$ of Definition 102) and explicitly represented using the parallel composition operator of the spi calculus. Any intended sequential occurrence of actions, namely those actions that are associated to the same agent, is preserved by using the prefix operator of the spi calculus. The private names are then simply put as a top-level restriction around the parallel composition.

We consider as target spi calculus the one we defined in Chapter 3. However, we assume here that the set of spi calculus names N is in bijection with $N_\diamond \cup A \cup V$. In other words, we assume that $N_\diamond \cup A \cup V$ is a partition of N and will thus identify expressions of this chapter with spi calculus expressions. We will also identify M_\diamond with a subset of M : the set of closed messages (with no variables). We also define a way to guard a process P by a formula ϕ :

$$\begin{aligned} \text{guard}(tt, P) &:= P \\ \text{guard}([E=F], P) &:= [E=F]P \\ \text{guard}([E:M], P) &:= [E=E]P \\ \text{guard}(\text{inverse}(E, F), P) &:= \text{guard}([\text{Dec}_F^a \text{Enc}_E^a(E.F):M], P) \\ \text{guard}(\phi \wedge \psi, P) &:= \text{guard}(\phi, \text{guard}(\psi, P)) \end{aligned}$$

Definition 102 (Translation).

Let $X \in \mathcal{X}$ be an executable narration.

1. $\mathcal{A}(X)$ (Table 7.12) defines the set of agents acting in X .
2. $\mathcal{R}(X)$ (Table 7.12) defines the set of fresh restricted names of X .
3. $X \upharpoonright_A$ (Table 7.13) defines the spi projection of X on $A \in \mathcal{A}$.
4. The translation $\mathcal{T}[X]$ of X into spi calculus is defined by:

$$\mathcal{T}[X] := (\nu n)_{n \in \mathcal{R}(X)} \prod_{A \in \mathcal{A}(X)} X \upharpoonright_A$$

where $(\nu n)_{n \in I}$ and $\prod_{n \in I}$ denote n -ary restriction and composition.

$$\begin{aligned}
\mathcal{A}(\epsilon) &:= \emptyset \\
\mathcal{A}(A:B!E; X) &:= \{A\} \cup \mathcal{A}(X) \\
\mathcal{A}(A:?x; X) &:= \{A\} \cup \mathcal{A}(X) \\
\mathcal{A}(A:\phi; X) &:= \{A\} \cup \mathcal{A}(X) \\
\mathcal{A}(vn; X) &:= \mathcal{A}(X) \\
R(\epsilon) &:= \emptyset \\
R(A:B!E; X) &:= R(X) \\
R(A:?x; X) &:= R(X) \\
R(A:\phi; X) &:= R(X) \\
R(vn; X) &:= \{n\} \cup R(X)
\end{aligned}$$

Table 7.12: Definition of $\mathcal{A}(\cdot)$, $R(\cdot)$

$$\begin{aligned}
\epsilon \uparrow_A &:= \mathbf{0} \\
(A':B!E; X) \uparrow_A &:= \begin{cases} \overline{B}\langle E \rangle.X \uparrow_A & \text{if } A' = A \\ X \uparrow_A & \text{otherwise} \end{cases} \\
(A':?x; X) \uparrow_A &:= \begin{cases} A(x).X \uparrow_A & \text{if } A' = A \\ X \uparrow_A & \text{otherwise} \end{cases} \\
(A':\phi; X) \uparrow_A &:= \begin{cases} \text{guard}(\phi, X \uparrow_A) & \text{if } A' = A \\ X \uparrow_A & \text{otherwise} \end{cases} \\
(vn; X) \uparrow_A &:= X \uparrow_A
\end{aligned}$$

Table 7.13: Definition of $\cdot \uparrow$.

We now conclude by showing that the operational semantics of executable narrations and their spi calculus translations coincide up to \equiv . Since the evaluation of expressions fails on open expressions (expressions E such that $v(E) \neq \emptyset$) and since the semantics uses closed messages for inputs (messages of M_\circ), this result can only hold on closed executable narrations. According to the translation function, these narrations X are characterised by $\text{fn}(\mathcal{T}[\![X]\!]) \cap V = \emptyset$.

The following theorem states the correspondence.

Theorem 18:

Let $X \in X$ be a closed executable narration.

1. if $X \xrightarrow{A: ?M} X'$ then there is P' such that $\mathcal{T}[\![X]\!] \xrightarrow{A} (x)P'$ and such that $P'\{M/x\} \equiv \mathcal{T}[\![X']\!]$.
2. if $X \xrightarrow{A:(v\tilde{n})B!M} X'$ then there is P' such that $\mathcal{T}[\![X]\!] \xrightarrow{\bar{B}} (v\tilde{n})\langle M \rangle P'$ and such that $P' \equiv \mathcal{T}[\![X']\!]$.
3. if $\mathcal{T}[\![X]\!] \xrightarrow{\alpha} (x)P'$ and $M \in M_\circ$ then there exist $A \in A$ and X' such that $\alpha = A, X \xrightarrow{A: ?M} X'$ and $P'\{M/x\} \equiv \mathcal{T}[\![X']\!]$.
4. if $\mathcal{T}[\![X]\!] \xrightarrow{\bar{\alpha}} (v\tilde{n})\langle M \rangle P'$ then there exist $A, B \in A$ and X' such that $\alpha = B, X \xrightarrow{A:(v\tilde{n})B!M} X'$ and $P' \equiv \mathcal{T}[\![X']\!]$.

PROOF

The proof relies on several facts:

- It is obvious that if $X \cong_\alpha X'$ then $\mathcal{T}[\![X]\!] \equiv \mathcal{T}[\![X']\!]$.
- Every executable narration X can be written

$$X \cong_\alpha v n_1 ; \dots ; v n_k ; X_{A_1} ; \dots ; X_{A_l}$$

with $R(X) = \{n_1, \dots, n_k\}$, $A(X) = \{A_1, \dots, A_l\}$, $R(X_{A_i}) = \emptyset$ and $A(X_{A_i}) = \{A_i\}$ for $1 \leq i \leq l$.

$$\mathcal{T}[\![X]\!] \equiv (v n_i)_{1 \leq i \leq k} \prod_{1 \leq j \leq l} X_{A_j} \upharpoonright_{A_j}$$

- If $E \in E$ is closed (i.e. $n(E) \cap V = \emptyset$) then $\mathbf{e}_c(E) = \llbracket E \rrbracket$.

Thus, if $E, F \in E$ are closed, we have

- $\llbracket [E = F] \rrbracket = \mathbf{e}(\llbracket E = F \rrbracket)$
- $\llbracket \text{inverse}(E, F) \rrbracket = \mathbf{e}(\llbracket \text{Dec}_E^{\mathfrak{a}} \text{Enc}_E^{\mathfrak{a}}(E.F) : M \rrbracket)$

The result is then straightforward. ■

7.7 spyer

`spyer` is a tool, developed in `ocaml`, that implements the previous formal developments. A source distribution of `spyer` can be found online [44]; an early version was developed by Gensoul [74].

`spyer` takes as an input file an extended protocol narration (using also the syntactic sugar described at the end of Section 7.2) and outputs an executable protocol narration and/or a network of spi calculus processes. The latter can then be used as input for our bisimulation checker `sbc` that implements the symbolic bisimulation described in [39]. We have briefly summarised in Table 7.14 the correspondence between the abstract syntax of expressions and formulae used in this chapter and the expressions and formulae used by `spyer`.

this chapter	\rightsquigarrow	spyer
<i>Expressions</i>		
a		<code>a</code>
x		<code>x</code>
A		<code>A</code>
$\text{Enc}_F^S E$		<code>enc_s(E, F)</code>
$\text{Enc}_F^A E$		<code>enc_a(E, F)</code>
$\text{Dec}_F^S E$		<code>dec_s(E, F)</code>
$\text{Dec}_F^A E$		<code>dec_a(E, F)</code>
$(E . F)$		<code><E, F></code>
$\pi_1(E)$		<code>fst(E)</code>
$\pi_2(E)$		<code>snd(E)</code>
$\text{pub}(E)$		<code>pub(E)</code>
$\text{priv}(E)$		<code>priv(E)</code>
$H(E)$		<code>hash(E)</code>
<i>Formulae</i>		
$[E : M]$		<code>wff(E)</code>
$[E = F]$		<code>[E=F]</code>
$\text{inverse}(E, F)$		<code>inv(E, F)</code>
$F \wedge G$		<code>F /\ G</code>

Table 7.14: Correspondence of abstract and `spyer` syntax for expressions and formulae

Before commenting some examples adapted from [58], we explain how consistency formulae, which can quickly become huge, may be simplified.

7.7.1 Simplifying formula.

The various sub-formulae generated by the consistency formula of Definition 98 contain lots of redundant information.

Example 26

For example, if $K = \{(A, A), (B, B), ((A \cdot B), x), (A, \pi_1(x)), (B, \pi_2(x))\}$, then

$$\begin{aligned} \Phi(K) = & [A:\mathbf{M}] \wedge [B:\mathbf{M}] \wedge [x:\mathbf{M}] \\ & \wedge [\pi_1(x):\mathbf{M}] \wedge [\pi_2(x):\mathbf{M}] \\ & \wedge [A = \pi_1(x)] \wedge [B = \pi_2(x)] \\ & \wedge [x = (A \cdot B)] \wedge [x = (\pi_1(x) \cdot B)] \\ & \wedge [x = (\pi_1(x) \cdot \pi_2(x))] \wedge [x = (A \cdot \pi_2(x))] \end{aligned}$$

Actually, we should also add the symmetric tests since they are syntactically different and the Definition 98 ignores the symmetry of $[\cdot = \cdot]$ and $\text{inverse}(\cdot, \cdot)$. *

To avoid this combinatorial explosion, we devise some mostly straightforward rules to simplify formulae. Before stating them, we define formula equivalence.

Definition 103 (Formula Equivalence).

Two formulae ϕ and ψ are equivalent—written $\phi \approx \psi$ —if and only if for all (closing) substitutions $\sigma : V \rightarrow M_\circ$, we have $\llbracket \phi \sigma \rrbracket = \llbracket \psi \sigma \rrbracket$.

Since substitution correspond to message reception, two formulae are thus equivalent if they evaluate in the same way in every execution.

In the following enumeration of equivalence laws, with $\phi_1 \wedge \phi_2 \approx \phi_2 \wedge \phi_1$ and $(\phi_1 \wedge \phi_2) \wedge \phi_3 \approx \phi_1 \wedge (\phi_2 \wedge \phi_3)$, we consider formulae up to commutativity and associativity of the conjunction operator.

The first set of laws states the symmetry and transitivity of the equality test.

- $[E = F] \wedge \phi \approx [F = E] \wedge \phi$
- $[E = F] \wedge [F = G] \wedge \phi \approx [E = F] \wedge [E = G] \wedge \phi$

The second set of laws simplifies well-formedness tests or inversion tests.

- If $\phi = [E:\mathbf{M}] \wedge \phi'$ and E is an expression without deconstructors (i.e., that does not contain any occurrence of $\pi_1(\cdot)$, $\pi_2(\cdot)$ or $\text{Dec}(\cdot)$), then $\phi \approx \phi'$.

- If $\phi = [E : \mathbf{M}] \wedge \phi'$ and E appears as a subexpression of an expression appearing in ϕ' , then $\phi \approx \phi'$.
- If $\phi = [\pi_1(E) : \mathbf{M}] \wedge \phi'$ and $\pi_1(E)$ or $\pi_2(E)$ appear as a subexpression of an expression appearing in ϕ' , then $\phi \approx \phi'$.
- If $\phi = [\pi_2(E) : \mathbf{M}] \wedge \phi'$ and $\pi_1(E)$ or $\pi_2(E)$ appear as a subexpression of an expression appearing in ϕ' , then $\phi \approx \phi'$.
- $\text{inverse}(M, N) \wedge \phi \approx \phi$ if $\text{inv}(M) = N$

The third set of laws rewrites well-formedness tests or inversion tests in terms of equality tests.

- $\text{inverse}(E, F) \wedge \phi \approx [\text{Dec}_F^a \text{Enc}_E^a G : \mathbf{M}] \wedge \phi$ for all G that do not contain destructors or if it contains some, they are inside an exact occurrence of E or F . For example, $G = F$ or $G = E$ are valid choices for G .
- $[E : \mathbf{M}] \wedge \phi \approx [E = E] \wedge \phi$.

The following law states a substitutivity property of equality:

- $[E = F] \wedge \phi \approx [E = F] \wedge \phi'$, for all ϕ' which is ϕ where some occurrences of E has been replaced by F or conversely.

Finally, the last set of laws rewrites equality tests such that the resulting expressions contain fewer constructors.

- $[(E_1 \cdot E_2) = (F_1 \cdot F_2)] \wedge \phi \approx [E_1 = F_1] \wedge [E_2 = F_2] \wedge \phi$
- $[\text{Enc}_{E_2}^s E_1 = \text{Enc}_{F_2}^s F_1] \wedge \phi \approx [E_1 = F_1] \wedge [E_2 = F_2] \wedge \phi$
- $[\text{Enc}_{E_2}^a E_1 = \text{Enc}_{F_2}^a F_1] \wedge \phi \approx [E_1 = F_1] \wedge [E_2 = F_2] \wedge \phi$
- $[\text{H}(E) = \text{H}(F)] \wedge \phi \approx [E = F] \wedge \phi$
- $[\text{pub}(E) = \text{pub}(F)] \wedge \phi \approx [E = F] \wedge \phi$
- $[\text{priv}(E) = \text{priv}(F)] \wedge \phi \approx [E = F] \wedge \phi$
- $[(E_1 \cdot E_2) = F] \wedge \phi \approx [E_1 = \pi_1(F)] \wedge [E_2 = \pi_2(F)] \wedge \phi$

Example 27

With the above laws, the formula of Example 26 is provably equivalent to:

$$\psi = [A = \pi_1(x)] \wedge [B = \pi_2(x)]$$

These ideas are implemented in `spyer`. Moreover, it exploits the fact that the consistency formula is only used when adding a pair (M, x) to an already reduced knowledge set K . So, to avoid that the same checks are performed several times, it keeps in a formula only the atoms involving the variable x .

7.7.2 The Wide-Mouthed Frog Protocol

First, we give the input file corresponding to the Wide-Mouthed Frog protocol that we have studied earlier in this chapter.

```
(* Wide Mouthed Frog protocol *)
(* initial knowledge *)
A,B,S know A B S
A,S share kAS
B,S share kBS
A generates kAB
A know m

(* protocol narration *)
A -> S: <A,enc_s(<B,kAB>,kAS)>
S -> B: enc_s(<A,B,kAB>,kBS)
A -> B: enc_s(m,kAB)
```

We then invoke `spyer` with the above file to obtain the following executable narration:

```
new kAS
new kBS
A: new kAB
A: S!<A,enc_s(<B,kAB>,kAS)>
S: ?0
S: [B = fst(dec_s(snd(0),kAS))] [A = fst(0)]
S: B!enc_s(<A,<B,snd(dec_s(snd(0),kAS))>>,kBS)
B: ?1
B: [B = fst(snd(dec_s(1,kBS)))]
  [A = fst(dec_s(1,kBS))]
A: B!enc_s(m,kAB)
B: ?2
B: wff(dec_s(2,snd(snd(dec_s(1,kBS))))))
```

Finally, this gives the following spi-calculus system:

```

agent A(agent_A, agent_B, agent_S, kAS, m) =
  (^kAB)
  'agent_S<<agent_A, enc_s(<agent_B, kAB>, kAS)>>.
  'agent_B<enc_s(m, kAB)>.0

agent B(agent_A, agent_B, kBS) =
  agent_B(x_1).
  {[agent_B = fst(snd(dec_s(x_1, kBS)))]}
  /\ [agent_A = fst(dec_s(x_1, kBS))]}
  agent_B(x_2).
  {wff(dec_s(x_2, snd(snd(dec_s(x_1, kBS)))))}0

agent S(agent_A, agent_B, agent_S, kAS, kBS) =
  agent_S(x_0).
  {[agent_B = fst(dec_s(snd(x_0), kAS))]}
  /\ [agent_A = fst(x_0)]}
  'agent_B<enc_s(<agent_A,
                  <agent_B,
                  snd(dec_s(snd(x_0), kAS))>>,
                  kBS)>.0

agent System(agent_A, agent_B, agent_S, m) =
  (^kAS, kBS)
  (A(agent_A, agent_B, agent_S, kAS, m)
   | B(agent_A, agent_B, kBS)
   | S(agent_A, agent_B, agent_S, kAS, kBS))

```

7.7.3 The Otway-Rees protocol

In the Otway-Rees protocol, lots of redundant information is shared by participants. For example, the message m appears in every single message exchange. The extended narration corresponding to the Otway-Rees protocol is:

```

(* Otway Rees protocol *)
(* initial knowledge *)
A,B,S know A B S
A,S share kAS
B,S share kBS
A know m
A generates nA

```

B generates nB
S generates kAB

```
(* protocol narration *)
A -> B : <m,A,B,enc_s(<nA,m,A,B>,kAS)>
B -> S : <m,A,B,
        enc_s(<nA,m,A,B>,kAS),enc_s(<nB,m,A,B>,kBS)>
S -> B : <m,enc_s(<nA,kAB>,kAS),enc_s(<nB,kAB>,kBS)>
B -> A : <m,enc_s(<nA,kAB>,kAS)>
```

The executable narration computed by `spyer` is then:

```
new kAS
new kBS
A: new nA
B: new nB
S: new kAB
A: B!<m,<A,<B,enc_s(<nA,<m,<A,B>>>,kAS)>>>
B: ?0
B: [B = fst(snd(snd(0)))] [A = fst(snd(0))]
B: S!<fst(0),<A,<B,<snd(snd(snd(0))),
    enc_s(<nB,<fst(0),<A,B>>>,kBS)>>>>
S: ?1
S: [snd(snd(dec_s(snd(snd(snd(snd(1))))),kBS))]
    = <A,B>]
    [snd(snd(dec_s(fst(snd(snd(snd(1))))),kAS))]
    = <A,B>]
    [fst(1)
    = fst(snd(dec_s(snd(snd(snd(snd(1))))),kBS))]
    [fst(1)
    = fst(snd(dec_s(fst(snd(snd(snd(1))))),kAS))]
    [B = fst(snd(snd(1)))]
    [A = fst(snd(1))]
S: B!<fst(1),
    <enc_s(<fst(dec_s(fst(snd(snd(snd(1))))),kAS)),
    kAB>,kAS),
    enc_s(<fst(dec_s(snd(snd(snd(snd(1))))),kBS)),
    kAB>,kBS)>>
B: ?2
B: [fst(2) = fst(0)]
    [nB = fst(dec_s(snd(snd(2)),kBS))]
B: A!<fst(2),fst(snd(2))>
```

A: ?3

A: [nA = fst(dec_s(snd(3), kAS))] [m = fst(3)]

Actually, if the computation of the consistency formulae was following literally what we devised in Section 7.3, the generated formulae of S would contain more than 300,000 (!) equality tests.

7.8 Related work and future work

We roughly and somewhat artificially divide the list of competing approaches into two classes. One class tries to tightly associate some formal semantics with (variants of) narrations themselves. Another class provides less tight associations, usually involving a different target formalism: here, we may distinguish approaches that informally reformulate narrations within a different formalism from those that offer mostly automated translations from narrations into target formalisms, but do not clearly justify the underlying translation principles (if exposed at all). Let us, for the purpose of structuring this section, use the terms “tight” and “lax” semantics to separate the two classes.

Sumii et. al. [132] propose a formal semantics of narrations by translation into spi calculus. The paper is written in Japanese, so it remains unclear to us how “tight” the approach really is, how they treat the problem of checks-on-reception, and also whether there is any formal or informal justification of the translation principles. In any case, our own intention was to provide a formal semantics that does *not require* the use of an underlying (and possibly too) general process calculus, so our approach is still substantially different.

Tight semantics

The work of Caleiro, Viganó and Basin [54, 55] is quite similar in spirit and aim with our work. They defined a trace-based denotational semantics and gave a corresponding operational semantics with a variant of pattern-matching spi calculus as target language. Some underlying ideas are quite similar to ours but we find our formalism of knowledge sets is more lightweight (although equally powerful) than their theory of view/opacity. The *view* that a principal has of a message M corresponds to how far it understands the message M with its current knowledge. A message is said to be *opaque* for a participant if the latter is not able to analyse at all the form of the message (this corresponds to a view equal to a special symbol γ_M). To relate to these definitions, one might say that our approach

consists in considering that initially a received message M is opaque and is thus bound to a fresh variable x_M . Then, the analysis of the receiver's knowledge set resulting from the addition of (M, x_M) to its current knowledge set corresponds to computing the view that the receiver then has of this particular message. In addition, the receiver also updates the "view" that it has of other previously received message. Then, we can directly use the result of the analysis to say which checks have to occur after the reception of M . In contrast, Caleiro, Viganó and Basin had to introduce and refer to further concepts like the *facial pattern*, the *constructive form* and the *inner facial pattern* (and relate them with the concept of view) before being able to give an operational semantics. The main simplification in our setting arises from the joint treatment of messages and associated "views" as knowledge elements of the form (M, E) .

Another way to give semantics to protocol narrations could exploit the widely-developed machinery of *strand spaces* [134], proposed by Thayer Fábrega, Herzog and Guttman. This formalism has proved to be a successful framework for reasoning on and verifying security properties of cryptographic protocols [77]. Strand spaces are graphs that represent the intended protocol behaviour of narrations by an explicit use of arrow-notations: one type of arrow captures the sequential dependencies within individual participants, giving rise to *strands*; another type of arrow captures the flow of messages between strands. In contrast to mere narrations, strand spaces are not limited to represent just the intended behaviour, but also the behaviour of malicious attackers, represented as so-called *penetrator strands*. Since strand spaces come with a formal semantics, in terms of so-called *bundles*, this immediately also provides some semantics for narrations. A bundle can be understood as a causality-closed subgraph of a given strand space. As such, it represents the possible result of a valid executions of the strand space. However, there is no notion of *dynamic* execution that could be understood as a form of operational semantics. Thus, there is also no study of dynamic checks-on-reception, which is the main technical contribution of the current chapter.

On the other hand, strand spaces have also been studied by Crazzolaro and Winskel in comparison to other models of concurrency [62, 63], notably including event structures Petri nets and the algebraic process language SPL; the latter is a simplified (since channel-free) spi-calculus that is enhanced with some form of pattern-matching (cf. also [78] for pattern-matching in a more standard spi-calculus, and our comments below). Since these models of concurrency—in particular the language SPL—are equipped with forms of operational semantics, one might think that their relation to strand spaces could provide an operational semantics of

the latter “for free”. This, however, is not the case. In [62], for any given SPL-process P in some particularly restricted form called *!-par* process, Crazzolarà and Winskel show how to formally and closely relate the net behaviour $\text{Net}(P)$ to the strand space behaviour $\text{Tr}(P)$. In contrast, they do not offer any way to translate strand spaces back into SPL-terms, which would be required to inherit the desired operational semantics. In [63], the authors further refine the relation between SPL and strand spaces by extending the latter with a notion of conflict to allow for better composition properties. Still, they offer no way to translate arbitrary strand spaces to SPL processes.

Lax semantics

The work of Bodei et. al. [30, 31] is also similar to ours, although still quite different. Like us, they present a refinement of protocol narrations, but the respective checks-on-reception appear only informally. Like us, they split message exchanges into three parts, albeit different to ours. A formal semantics is then only provided after “rewriting”, again informally, refined narrations into terms of their channel-free process calculus LYSA. To our knowledge, the above papers (short and long version) provide the best available information about the system underlying their “systematic expansion”. Still, the expansion is not unambiguously explained, while our expansion is fully automatic, based on simple intuitive principles and generates a maximum number of checks according to these principles. Finally, their approach aims at static analysis techniques, while we ultimately target at dynamic analysis, e.g., as in the form of bisimulation checks [39] in the spi calculus.

In other related approaches, narrations are reformulated or translated using Casper [94], HLPSSL2IF [22], CAPSL [96], CASRUL [87], or (s)pi calculus [9, 28]. They have in common that they do not easily help to understand how the gap between the rather informal narrations and the target formalism is bridged. A compiler can itself be interpreted as giving semantics to narrations, but usually the translation process is not well explained or otherwise justified, in particular regarding the treatment of checks-on-reception. Moreover, our interest was to try to formalise the semantics at the level of narrations rather than by translation into some reasonably unrelated target formalism.

A subtle, but interesting difference between our work and Casper [94] is their modified message syntax using a construction $M \text{ } \% \text{ } v$, meaning that the recipient of M should *not* try to decrypt M . We think this construct was added because of Casper’s rather strict policy to *require*, unless the

% is used, to be able to fully decrypt all messages (and possibly provide a warning in case this fails). Our (arguably more flexible) policy is instead to require agents to always *just try* to decrypt messages as far as their current knowledge permits, so we implicitly let agents accept messages even if they cannot (yet) fully decrypt them.

As we previously observed (cf. Section 7.4.3), the pattern-matching spi calculus of [78] is not expressive enough to capture the checks-on-reception we require. To overcome its limitations (in the part of our work that deals with rewriting into spi calculus), we could have used a variant similar in spirit to the one of [55]. However, we found it more orthogonal and extensible to express those checks by means of dedicated formulae. Moreover, our use of this version of the spi calculus was driven by the wish to have our tool `spyer` generate spi calculus code that is compatible with our symbolic bisimulation checker based on [39]. Finally, note that the aim of [78] was to offer a type system to study safety property of processes, but not to enable an automated way to turn protocol narrations into spi calculus processes.

Future work

Here, we do not tackle the fourth task listed by Abadi [2] on how to get to a formalisation of concurrent sessions on the basis of protocol narrations. The main problem is that *principals* may play different *roles* in concurrent sessions such that the lookup of their respective keys needs to be dealt with dynamically. The usual convenient confusion of the two concepts of principal and role is no longer appropriate, so we propose to non-trivially extend the narration notation rather than providing a suboptimal semantics to an inappropriate notation. Note that this confusion also rules out the naïve modelling of concurrent sessions by the bare unbounded replication within spi calculus. Some inspiration from the work of Cremers and Mauw [64] and the work done in the context of mixed strand spaces [133, 76] may help us here.

Furthermore, it should be possible to develop reasoning techniques for protocol narrations via an *environment-sensitive* extension of our semantics that could be used to define and study meaningful behavioural equivalences.

Conclusion

In this thesis, we have tackled the lack of tool support of the spi calculus approach for studying cryptographic protocols.

To achieve this goal, some theoretical developments were needed. On the one hand we have established a rigorous link between security protocols and spi calculus by providing a formal semantics to protocol narrations. On the other hand, inspired by the situation in the pi calculus where open bisimulation of Sangiorgi has given rise to several implementations of bisimulation checkers, we have defined an open style definition of bisimulation for the spi calculus. We have proven that this new notion of bisimulation —namely open hedged bisimulation— is indeed an extension of open bisimulation of the pi calculus and that it is a sound proof technique to show late hedged bisimilarity. We have also given a symbolic characterisation of open hedged bisimulation that, we have argued, constitutes a promising first step towards its mechanisation. Interestingly enough, the projection down to the pi calculus of open hedged bisimulation has enabled us to better understand certain aspects of the pi calculus and to formulate more precise congruence properties of open bisimulation than the ones that were first stated by Sangiorgi.

On the tool support side, we have implemented into `spyer` the formal procedure to translate a protocol narration into spi calculus. Besides, we have formalised the spi calculus in the `coq` proof assistant. Even if the goals of this formalisation were firstly to validate our theoretical developments and secondly to extract eventually a bisimulation checker, this already constitutes a sufficiently complete framework to reason formally in `coq` about security protocols using the spi calculus approach.

As obvious future work, we intend to pursue our study of open hedged bisimilarity. Taking advantage of the experience we got while working on a prototype tool [38], we hope to soon be able to devise a decision procedure for checking open hedged bisimilarity on finite terms and prove its correctness in `coq`. We also intend to investigate open hedged bisimulation from a more theoretical point of view; we have good hope that the induced notion of bisimilarity enjoys good congruence properties.

Bibliography

- [1] Martín Abadi. Secrecy by typing in security protocols. *Journal of the ACM*, 46(5):749–786, September 1999. An abstract appeared in the *Proceedings of TACS '97*, volume 1281 of LNCS.
- [2] Martín Abadi. Security protocols and their properties. In F. Bauer and Ralf Steinbrueggen, editors, *Foundations of Secure Computation*, pages 39–60. NATO ASI, IOS Press, 2000.
- [3] Martín Abadi and Bruno Blanchet. Secrecy types for asymmetric communication. In Honsell and Miculan [85], pages 25–41.
- [4] Martín Abadi and Bruno Blanchet. Analyzing security protocols with secrecy types and logic programs. In *POPL*, pages 33–44, 2002.
- [5] Martín Abadi and Bruno Blanchet. Analyzing security protocols with secrecy types and logic programs. *J. ACM*, 52(1):102–146, 2005.
- [6] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Proceedings of POPL '01*, pages 104–115. ACM, January 2001.
- [7] Martín Abadi and Andrew D. Gordon. Reasoning about cryptographic protocols in the Spi calculus. In Antoni Mazurkiewicz and Józef Winkowski, editors, *Proceedings of CONCUR '97*, volume 1243 of LNCS, pages 59–73. Springer, 1997.
- [8] Martín Abadi and Andrew D. Gordon. A bisimulation method for cryptographic protocols. *Nordic Journal of Computing*, 5(4):267–303, Winter 1998. An extended abstract appeared in the *Proceedings of ESOP '98*, LNCS 1381, pages 12–26.
- [9] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The Spi calculus. *Journal of Information and Computation*, 148(1):1–70, 1999. An extended abstract appeared in the *Proceedings*

- of the Fourth ACM Conference on Computer and Communications Security (Zürich, April 1997)*. An extended version of this paper appears as Research Report 149, Digital Equipment Corporation Systems Research Center, January 1998, and, in preliminary form, as Technical Report 414, University of Cambridge Computer Laboratory, January 1997.
- [10] Martin Abadi and Roger Needham. Prudent engineering practice for cryptographic protocols. *Software Engineering*, 22(1):6–15, 1996.
 - [11] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *IFIP International Conference on Theoretical Computer Science (IFIP TCS2000)*, Sendai, Japan, 2000. Springer-Verlag, Berlin Germany.
 - [12] Roberto M. Amadio and Denis Lugiez. On the reachability problem in cryptographic protocols. In Catuscia Palamidessi, editor, *Proceedings of CONCUR 2000*, volume 1877 of LNCS, pages 380–394. Springer, August 2000.
 - [13] Roberto M. Amadio and Sanjiva Prasad. The game of the name in cryptographic tables. In P[azhamaneri] S. Thiagarajan and Roland Yap, editors, *Proceedings of ASIAN '99*, volume 1742 of LNCS, pages 15–26. Springer, December 1999.
 - [14] R. Anderson and R. Needham. Programming satan’s computer. In J. van Leeuwen, editor, *Computer Science Today — Recent Trends and Developments*, volume 1000 of LNCS, pages 426–440. Springer, 1996.
 - [15] N. Asokan, Victor Shoup, and Michael Waidner. Asynchronous protocols for optimistic fair exchange. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 86–99, 1998.
 - [16] Brian E. Aydemir, Aaron Bohannon, Matthew Fairbairn, J. Nathan Foster, Benjamin C. Pierce, Peter Sewell, Dimitrios Vytiniotis, Geoffrey Washburn, Stephanie Weirich, and Steve Zdancewic. Mechanized metatheory for the masses: The POPLMARK challenge. In Springer Verlag, editor, *Theorem Proving in Higher Order Logics*, volume 3603 of *Lecture Notes in Computer Science*, pages 50–65, 2005.
 - [17] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1999.

- [18] M. Backes, B. Pfitzmann, and M. Waidner. A universally composable cryptographic library, 2003.
- [19] Michael Baldamus, Joachim Parrow, and Björn Victor. Spi calculus translated to pi-calculus preserving may-tests. In *Logics in Computer Science*, pages 22–31. IEEE Computer Society Press, 2004.
- [20] Michael Baldamus, Joachim Parrow, and Björn Victor. A fully abstract encoding of the pi-calculus with data terms. In *ICALP*, volume 3580 of *Lecture Notes in Computer Science*. Springer-Verlag, 2005.
- [21] H.P. Barendregt. *The Lambda Calculus. Its Syntax and Semantics.*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 1984.
- [22] D. Basin, S. Mödersheim, and L. Viganò. An On-The-Fly Model-Checker for Security Protocol Analysis. In *Proceedings of ESORICS'03*, LNCS 2808, pages 253–270. Springer-Verlag, Heidelberg, 2003.
- [23] Giampaolo Bella and Lawrence C. Paulson. Using Isabelle to prove properties of the Kerberos authentication system. In Hilarie Orman and Catherine Meadows, editors, *Proceedings of the Workshop on Design and Formal Verification of Security Protocols*. DIMACS, 1997.
- [24] Jesper Bengtson and Joachim Parrow. Formalising the i -calculus using nominal logic. In Helmut Seidl, editor, *FoSSaCS*, volume 4423 of *Lecture Notes in Computer Science*, pages 63–77. Springer, 2007.
- [25] Gérard Berry and Gérard Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.
- [26] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development – Coq'Art: The Calculus of Inductive Constructions*. Springer Verlag, 2004.
- [27] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules, 2001.
- [28] Bruno Blanchet. Automatic verification of cryptographic protocols: A logic programming approach. In *Proceedings of Principles and Practice of Declarative Programming (PPDP'03)*. ACM, 2003.

- [29] Frédéric Blanqui, William Delobel, Solange Coupet-Grimal, Sébastien Hinderer, and Adam Koprowski. CoLoR, a Coq library on rewriting and termination. In *Eighth International Workshop on Termination (WST 06')*, 2006.
- [30] Chiara Bodei, Mikael Buchholtz, Pierpaolo Degano, Flemming Nielson, and Hanne Nielson. Automatic validation of protocol narration. In *Proceedings of 16th IEEE Computer Security Foundations Workshop (CSFW 16)*, pages 126–140, 2003.
- [31] Chiara Bodei, Mikael Buchholtz, Pierpaolo Degano, Flemming Nielson, and Hanne Nielson. Static validation of security protocols. *Journal of Computer Security*, 13:347–390, 2005.
- [32] Chiara Bodei, Pierpaolo Degano, Flemming Nielson, and Hanne Riis Nielson. Static analysis of processes for no read-up and no write-down. In Wolfgang Thomas, editor, *Proceedings of FoSSaCS '99*, volume 1578 of *LNCS*, pages 120–134. Springer, 1999.
- [33] Michele Boreale. Symbolic trace analysis of cryptographic protocols. In *Proceedings of ICALP 2001*, volume 2076 of *LNCS*, pages 667–681. Springer, July 2001.
- [34] Michele Boreale and Rocco De Nicola. Testing equivalences for mobile processes. *Journal of Information and Computation*, 120:279–303, 1995. Available as Report SI 92 RR 04, Università “La Sapienza” di Roma; an extended abstract appeared in *Proceedings of CONCUR '92*, LNCS 630.
- [35] Michele Boreale and Rocco De Nicola. A symbolic semantics for the π -calculus. *Journal of Information and Computation*, 126(1):34–52, 1996. Available as Report SI 94 RR 04, Università “La Sapienza” di Roma; an extended abstract appeared in *Proceedings of CONCUR '94*, pages 299–314, LNCS 836.
- [36] Michele Boreale, Rocco De Nicola, and Rosario Pugliese. Proof techniques for cryptographic processes. *SIAM Journal on Computing*, 31(3):947–986, 2002.
- [37] Michele Boreale and Daniele Gorla. On compositional reasoning in the spi-calculus. In Mogens Nielsen and Uffe H. Engberg, editors, *Proceedings of FoSSaCS 2002*, volume 2303 of *LNCS*, pages 67–81. Springer, April 2002.

- [38] Johannes Borgström and Sébastien Briaïs. Symbolic bisimulation checker, 2006.
- [39] Johannes Borgström, Sébastien Briaïs, and Uwe Nestmann. Symbolic bisimulation in the spi calculus. In Philippa Gardner and Nobuko Yoshida, editors, *CONCUR*, volume 3170 of *Lecture Notes in Computer Science*, pages 161–176. Springer, 2004.
- [40] Johannes Borgström and Uwe Nestmann. On bisimulations for the spi calculus. In Hélène Kirchner and Christophe Ringeissen, editors, *Proceedings of AMAST 2002*, volume 2422 of *LNCS*, pages 287–303. Springer, 2002.
- [41] Johannes Borgström and Uwe Nestmann. On bisimulations for the spi calculus. *Mathematical Structures in Comp. Sci.*, 15(3):487–552, 2005.
- [42] Sébastien Briaïs. *ABC Bisimulation Checker*. EPFL, 2003.
- [43] Sébastien Briaïs. Formal proofs about hedges using the Coq proof assistant, 2004. <http://lamp.epfl.ch/~sbriaïs/spi/hedges/hedge.html>.
- [44] Sébastien Briaïs. *spyer*. <http://lamp.epfl.ch/spyer>, 2006.
- [45] Sébastien Briaïs. A formalisation of the spi calculus in the Coq proof assistant, 2007. <http://lamp.epfl.ch/~sbriaïs/>.
- [46] Sébastien Briaïs. A Symbolic Characterisation of Open Bisimulation for the Spi Calculus. Technical report, 2007.
- [47] Sébastien Briaïs and Uwe Nestmann. A formal semantics for protocol narrations. In Rocco de Nicola and Davide Sangiorgi, editors, *Trustworthy Global Computing*, volume 3705 of *Lecture Notes in Computer Science*, pages 163–181. Springer, April 2005.
- [48] Sébastien Briaïs and Uwe Nestmann. Open bisimulation, revisited. In Jos Baeten and Iain Phillips, editors, *Proceedings of EXPRESS 2005: Expressiveness in Concurrency*, volume 154 of *Electronic Notes in Theoretical Computer Science*, pages 93–105. Elsevier B.V., 2005.
- [49] Sébastien Briaïs and Uwe Nestmann. A formal semantics for protocol narrations. *Theoretical Computer Science*, 2007. To Appear. doi:10.1016/j.tcs.2007.09.005.

- [50] Sébastien Briaïs and Uwe Nestmann. Open bisimulation, revisited. *Special Issue of Theoretical Computer Science*, 2007. To Appear. doi : 10.1016/j.tcs.2007.07.010.
- [51] Connor Mc Bride and James Mc Kinna. Functional pearl: I am not a number — i am a free variable. In ACM Press, editor, *Haskell'04: Proceedings of the 2004 ACM SIGPLAN workshop on Haskell*, pages 1–9, 2004.
- [52] John A. Bull and David J. Otway. The authentication protocol. Technical Report DRA/CIS3/PROJ/CORBA/SC/1/CSM/436-04/03, Defense Research Agency, 1997.
- [53] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.
- [54] Carlos Caleiro, Luca Viganò, and David Basin. Deconstructing alice and bob. In *Proceedings of the ICALP 2005 Workshop on Automated Reasoning for Security Protocol Analysis (ARSPA'05)*, volume 135.1 of *Electronic Notes of Theoretical Computer Science*, pages 3–22, 2005.
- [55] Carlos Caleiro, Luca Viganò, and David Basin. On the semantics of alice and bob specifications of security protocols. *Theoretical Computer Science*, 2006. To appear.
- [56] Iliano Cervesato, N. A. Durgin, Patrick Lincoln, John C. Mitchell, and Andre Scedrov. A meta-notation for protocol analysis. In *CSFW*, pages 55–69, 1999.
- [57] Yannick Chevalier. *Résolution de problèmes d'accessibilité pour la compilation et la validation de protocoles cryptographiques*. PhD thesis, Université Henri Poincaré – Nancy 1, 2003.
- [58] John A. Clark and Jeremy L. Jacob. A survey of authentication protocol literature. Technical Report 1.0, University of York, 1997.
- [59] François Pottier. *Caml*. Software and Documentation available on the Web, <http://pauillac.inria.fr/~fpottier/alphaCaml/>.
- [60] Hubert Comon-Lundh and Veronique Cortier. New decidability results for fragments of first-order logic and application to cryptographic protocols.

- [61] Véronique Cortier. *Vérification automatique des protocoles cryptographiques*. PhD thesis, École Normale Supérieure de Cachan, 2003.
- [62] Federico Crazzolarà and Glynn Winskel. Events in security protocols. In *ACM Conference on Computer and Communications Security*, pages 96–105, 2001.
- [63] Federico Crazzolarà and Glynn Winskel. Composing strand spaces. In *FST TCS '02*, volume 2556 of *LNCS*, pages 97–108. Springer, 2002.
- [64] Cas Cremers and Sjouke Mauw. Operational semantics of security protocols. In *Scenarios: Models, Algorithms and Tools (Dagstuhl 03371 Post-Seminar Proceedings)*, volume 3466 of *Lecture Notes in Computer Science*, 2005.
- [65] N.G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. In *Indagationes Mathematicae*, volume 34, pages 381–392, 1972.
- [66] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.
- [67] Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. RIPEMD-160: A strengthened version of RIPEMD. In *Fast Software Encryption*, pages 71–82, 1996.
- [68] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, (12):198–208, March 1983.
- [69] Luca Durante, Riccardo Sisto, and Adriano Valenzano. Automatic testing equivalence verification of spi-calculus specifications. *ACM Transactions on Software Engineering and Methodology*, 12(2):222–284, April 2003.
- [70] A. Elkjær, M. Höhle, H. Hüttel, and K. Nielsen. Towards automatic bisimilarity checking in the spi calculus, 1999.
- [71] Jean-Christophe Filliâtre and Pierre Letouzey. Functors for proofs and programs. In David A. Schmidt, editor, *ESOP*, volume 2986 of *Lecture Notes in Computer Science*, pages 370–384. Springer, 2004.

- [72] Yuxi Fu. On quasi-open bisimulation. *Theoretical Computer Science*, 338:96–126, 2005.
- [73] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.
- [74] Christophe Gensoul. *Spyer — un compilateur de protocoles cryptographiques*. Semester project report, EPFL, July 2003.
- [75] Georges Gonthier. A computer-checked proof of the four colour theorem. Technical report, Microsoft Research Cambridge, 2005.
- [76] Joshua D. Guttman and F. Javier Thayer Fábrega. Protocol independence through disjoint encryption. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW'00)*, pages 24–34, 2000.
- [77] Joshua D. Guttman and F. Javier Thayer Fábrega. Authentication tests and the structure of bundles. *Theoretical Computer Science*, 283(2):333–380, 2002.
- [78] Christian Haack and Alan S. A. Jeffrey. Pattern-matching spicalculus. *Information and Computation*, 204(8):1195–1263, 2006.
- [79] Matthew Hennessy and H. Lin. Symbolic bisimulations. *Theor. Comput. Sci.*, 138(2):353–389, 1995.
- [80] Matthew Hennessy and Julian Rathke. Typed behavioural equivalences for processes in the presence of subtyping. *Mathematical Structures in Computer Science*, 14(5):651–684, 2004.
- [81] Daniel Hirschhoff. A full formalization of pi-calculus theory in the Calculus of Constructions. In E. L. Gunter and A. Felty, editors, *Theorem Proving in Higher-Order Logic: 10th International Conference, TPHOLs'97*, volume 1275 of LNCS, pages 153–169, 1997.
- [82] Daniel Hirschhoff. *Mise en Œuvre de Preuves de Bisimulation*. PhD thesis, École Nationale des Ponts et Chaussées, 1999.
- [83] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, 1978.

- [84] Kohei Honda and Nobuko Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 152(2):437–486, 1995. An extract appeared in *Proceedings of FSTTCS '93*, LNCS 761.
- [85] Furio Honsell and Marino Miculan, editors. *Proceedings of the 4th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2001), Held as Part of the Joint European Conferences on Theory and Practice of Software (ETAPS 2001), (Genova, Italy, April 2001)*, volume 2030 of LNCS. Springer, 2001.
- [86] Hans Hüttel. Deciding framed bisimilarity. In Antonin Kucera and Richard Mayr, editors, *Proceedings of INFINITY 2002*, volume 68 of ENTCS. Elsevier Science Publishers, 2002.
- [87] F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and Verifying Security Protocols. In *Logic for Programming and Automated Reasoning*, volume 1955 of LNCS, pages 131–160. Springer-Verlag, November 2000.
- [88] David Kahn. *The Codebreakers*. McMillan, 1967.
- [89] S. Kramer. *Logical Concepts in Cryptography*. PhD thesis, École Polytechnique Fédérale de Lausanne, 2007.
- [90] Xavier Leroy. A locally nameless solution to the POPLMARK challenge. Technical Report 6098, INRIA, 2007.
- [91] P. Letouzey. *Programmation fonctionnelle certifiée – L'extraction de programmes dans l'assistant Coq*. PhD thesis, Université Paris-Sud, July 2004.
- [92] Gavin Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, November 1995.
- [93] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In T. Margaria and B. Steffen, editors, *Proceedings of TACAS '96*, volume 1055 of LNCS, pages 147–166. Springer, 1996.
- [94] Gavin Lowe. Casper: A compiler for the analysis of security protocols. *Journal of Computer Security*, 6:53–84, 1998. An extended abstract appeared in the *Proceedings of 10th IEEE Computer Security Foundations Workshop*, 1997.

- [95] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.
- [96] Jonathan K. Millen. CAPSL: Common authentication protocol specification language. <http://www.csl.sri.com/users/millen/capsl/>.
- [97] Dale Miller and Alwen Fernanto Tiu. A proof theory for generic judgments: An extended abstract. In *LICS*, pages 118–127. IEEE Computer Society, 2003.
- [98] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.
- [99] Robin Milner. The polyadic π -calculus: A tutorial. In Friedrich L. Bauer, Wilfried Brauer, and Helmut Schwichtenberg, editors, *Logic and Algebra of Specification*, volume 94 of *Series F*. NATO ASI, Springer, 1993. Available as Technical Report ECS-LFCS-91-180, University of Edinburgh, October 1991.
- [100] Robin Milner. *Communicating and Mobile Systems: the π -Calculus*. Cambridge University Press, May 1999.
- [101] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, parts I and II. Technical Report -86, 1989.
- [102] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, part I/II. *Journal of Information and Computation*, 100:1–77, September 1992.
- [103] Robin Milner and Davide Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *Proceedings of ICALP '92*, volume 623 of *LNCS*, pages 685–695. Springer, 1992.
- [104] Monniaux. Decision procedures for the analysis of cryptographic protocols by logics of belief. In *PCSFW: Proceedings of The 12th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1999.
- [105] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.
- [106] Rocco De Nicola and Matthew Hennessy. Testing equivalence for processes. In *Proceedings of the 10th Colloquium on Automata*,

- Languages and Programming*, pages 548–560, London, UK, 1983. Springer-Verlag.
- [107] U.S. Department of Commerce/National Bureau of Standards and Technology. Data encryption standard (DES). Federal Information Processing Standards Publication 46-3, 1999.
- [108] U.S. Department of Commerce/National Bureau of Standards and Technology. Advanced encryption standard (AES). Federal Information Processing Standards Publication 197, 2001.
- [109] U.S. Department of Commerce/National Bureau of Standards and Technology. Secure hash standard. Federal Information Processing Standards Publication 180-2, 2002.
- [110] Fredrik Orava and Joachim Parrow. An algebraic verification of a mobile network. *Journal of Formal Aspects of Computing*, 4:497–543, 1992. Missing figures available at <http://www.it.kth.se/~joachim/handfig.ps.gz>.
- [111] Joachim Parrow. An introduction to the pi-calculus. In Jan Bergstra, Alban Ponse, and Scott Smolka, editors, *Handbook of Process Algebra*, pages 479–543. Elsevier Science, 2001.
- [112] Paulson. Proving security protocols correct. In *Proceedings of LICS '99*. IEEE, Computer Society Press, July 1999.
- [113] Lawrence C. Paulson. Mechanized proofs for a recursive authentication protocol. In *10th Computer Security Foundations Workshop*, pages 84–95. IEEE Computer Society Press, 1997.
- [114] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.
- [115] Lawrence C. Paulson. Inductive analysis of the Internet protocol TLS. *Journal of Computer Security*, 2(3):332–351, 1999.
- [116] A. M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*, 186:165–193, 2003.
- [117] Alain Poli and Llorenç Huguet. *Codes correcteurs: Théorie et applications*. Masson, 1988.
- [118] R. Rivest. The MD5 message-digest algorithm. MIT Laboratory for Computer Science and RSA Data Security, Inc., 1992. RFC 1321.

- [119] R. L. Rivest, A. Shamir, and L. M. Adelman. A method for obtaining digital signatures and public-key cryptosystems. Technical Report MIT/LCS/TM-82, 1977.
- [120] Christine Röckl and Daniel Hirschhoff. A fully adequate shallow embedding of the [pi]-calculus in isabelle/hol with mechanized syntax analysis. *J. Funct. Program.*, 13(2):415–451, 2003.
- [121] Christine Röckl, Daniel Hirschhoff, and S. Berghofer. Higher-order abstract syntax with induction in Isabelle/HOL: Formalizing the pi-calculus and mechanizing the theory of contexts. In Honsell and Miculan [85], pages 364–378.
- [122] Anthony W. Roscoe. Model-checking CSP. In Anthony W. Roscoe, editor, *A Classical Mind: Essays in Honour of C.A.R. Hoare*. Prentice Hall, 1994.
- [123] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is np-complete, 2001.
- [124] Peter Ryan and Steve Schneider. *Modelling and Analysis of Security Protocols*. Addison Wesley, 2001.
- [125] Peter Y. A. Ryan and Steve A. Schneider. An attack on a recursive authentication protocol. a cautionary tale. *Inf. Process. Lett.*, 65(1):7–10, 1998.
- [126] D. Sangiorgi and D. Walker. On barbed equivalences in π -calculus. In *Proceedings of CONCUR '01: Concurrency Theory*, volume 2154 of *Lecture Notes in Computer Science*, pages 292–304. Springer Verlag, 2001.
- [127] Davide Sangiorgi. A theory of bisimulation for the π -calculus. *Acta Informatica*, 33:69–97, 1996. Earlier version published as Report ECS-LFCS-93-270, University of Edinburgh. An extended abstract appeared in the *Proceedings of CONCUR '93*, LNCS 715.
- [128] Davide Sangiorgi. On the bisimulation proof method. *Mathematical Structures in Computer Science*, 8(5):447–479, 1998. An extended abstract appeared in the *Proceedings of MFCS '95*, LNCS 969: 479–488.
- [129] Davide Sangiorgi and David Walker. *The π -calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

- [130] M.R. Shinwell and A.M. Pitts. *Fresh Objective Caml*. Software and Documentation available on the Web, <http://www.fresh-ocaml.org>.
- [131] Eijiro Sumii. Tutorial: Formal verification of cryptographic protocols in spi-calculus. Presented at JSIAM FAIS on December 22, 2006.
- [132] Eijiro Sumii, Hideaki Tatsuzawa, and Akinori Yonezawa. Translating security protocols from informal notation into spi calculus. *IPSJ Transactions on Programming*, 45(SIG 12):1–10, November 2004. Written in Japanese, abstract in English.
- [133] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Mixed strand spaces. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop (CSFW 1999)*, pages 72–82, 1999.
- [134] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(1):191–230, 1999.
- [135] The Coq Development Team. *The Coq Proof Assistant*. Software and Documentation available on the Web, <http://coq.inria.fr>.
- [136] The Isabelle Development Team. *Isabelle*. Software and Documentation available on the Web, <http://isabelle.in.tum.de/>.
- [137] The OCaml Development Team. *The OCaml Language*. Software and Documentation available on the Web, <http://caml.inria.fr>.
- [138] Alwen Tiu. A trace based bisimulation for the spi-calculus: an extended abstract. In *Proceedings of APLAS 2007*, 2007. To appear.
- [139] Alwen Tiu and Dale Miller. A proof search specification of the π -calculus. In *3rd Workshop on the Foundations of Global Ubiquitous Computing*, volume 138 of ENTCS, pages 79–101, September 2004.
- [140] Christian Urban and Christine Tasson. Nominal techniques in isabelle/hol. In Robert Nieuwenhuis, editor, *CADE*, volume 3632 of *Lecture Notes in Computer Science*, pages 38–53. Springer, 2005.
- [141] Björn Victor. *A Verification Tool for the Polyadic π -Calculus*. Licentiate thesis, Department of Computer Systems, Uppsala University, Sweden, May 1994. Available as report DoCS 94/50.

- [142] Björn Victor and Faron Moller. The Mobility Workbench — a tool for the π -calculus. In David Dill, editor, *Proceedings of CAV '94*, volume 818 of *LNCS*, pages 428–440. Springer, 1994.

Appendix A

Curriculum Vitæ

Personal information

Name	Sébastien Briais
Citizenship	French
Date of birth	January 28 th , 1978
Place of birth	Orléans, France

Education

2002–2007	Ph.D., Laboratoire des Méthodes de Programmation (LAMP), EPFL, Switzerland
1999–2002	Magistère Informatique et Modélisation, Mention Bien.
2001–2002	D.E.A. Programmation : Sémantique, Preuves et Langages, Mention Très Bien (3 ^{ème} sur 24).
2000–2001	Maîtrise d'Informatique, Mention Bien.
1999–2000	Licence d'Informatique, Mention Bien.
1999	Admission à l'École Normale Supérieure de Lyon (Concours Mathématiques).
1996–1999	Classes Préparatoires aux Grandes Écoles MPSI et MP* au Lycée Pothier, Orléans.
1996	Baccalauréat Scientifique, Option Mathématiques, Mention Bien (Académie d'Orléans-Tours).

Professional experience

- 2002–2007 Assistant-Doctorant, Laboratoire des Méthodes de Programmation (LAMP), EPFL
- 2002 Stage de D.E.A. (4 mois)
Towards Open Bisimulation in the spi-calculus
supervisé par Uwe Nestmann, LAMP, EPFL
- 2001 Stage de Maîtrise (10 semaines)
Banc d'essai de Funnel, Migration d'objets dans Øjeblik
supervisé par Michel Schinz et Uwe Nestmann, LAMP, EPFL
- 2000 Stage de Licence (6 semaines)
*Recherche de motifs approchés à l'aide du logiciel *grappe**
supervisé par Grégory Kucherov et Michaël Rusinowitch, LORIA, Nancy