Secrecy and Authenticity in Mobile Ad-Hoc Networks

Master's Thesis

Willard Þór Rafnsson

10th June 2008

Department of Computer Science
Aalborg University

Title:
Secrecy and Authenticity in Mobile Ad-Hoc Networks

Topic:
Process Algebra, Static Analysis, Logic Programming, Cryptographic Protocols, Mobile Ad-hoc Networks.

Group:
d603a

Group Members:
Willard Þór Rafnsson

Supervisor:
Hans Hüttel

Project Period:
DAT6 semester,
February 1st 2008 - June 10th 2008

Number of Copies: 5

Number of Pages: 125

Completed on:
10th June 2008

Abstract:
This thesis documents a framework for verifying secrecy and authenticity properties of Mobile Ad-hoc Networks. We present the Distributed Applied π Calculus with Broadcast, which is a simple, yet generic, expressive, and conceptually simple extension to the well-established Applied π calculus consisting of connectivity graphs, broadcast primitives, and explicit locations. The calculus can be instantiated by an arbitrary Term Rewrite System, and the semantics of the calculus is largely alike that of Applied π. We inherit key definitions of secrecy and authenticity from Applied π to our calculus, as well as those of frames and static equivalence.

We prove a powerful teorem expressing a syntactic relationship between Horn clauses generated from a process in the calculus of Abadi and Blanchet, and the source process. Several interesting corollaries follow from this theorem, including that Horn clause deduction overapproximates active syntactic secrecy, and that claims of message-passing deduced from the Horn clauses are sound with regards to the messages passed in the source process. During the process of this proof, we observe a frequently overlooked assumption in the Dolev-Yao threat model which causes frames to become inconsistent if particular internal reductions are performed. To address this, we give a (easily generalised) *revelation semantics* which ensures frames are consistent in the presense of such reductions.

At last, specifying how to extend the Horn clause generator to take into account connectivity graphs, We inherit a proof of soundness of deduction from Horn clauses for our calculus, thus clearing the way for the application of automated Horn clause constraint solvers.

Preface

This master's thesis documents the work we have made during the specialisation year in the Distributed Systems and Semantics unit, Department of Computer Science, Aalborg University, Denmark. The thesis is based on a project proposal by Hans Hüttel, who was my thesis adviser during my specialisation year.

While the thesis is otherwise self-contained, it is assumed that the reader has at least the Mathematical skills corresponding to a bachelor level in computer science. Furthermore, familiar with process calculi, in particular, with the π-calculus, would be advantagous. At last, it is assumed that the reader can grasp the basics of propositional- and predicate logic, abstract algebra, and is familiar with mathematical induction.

The work presented herein is largely written in the latter semester of the specialisation year. However, a few sections are derived on the work presented in the DAT5 project entitled *Knowledge in An Applied π Calculus With Locations*. While they have all seen change, they do deserve mentioning. These are parts of Chapter 1, Sections 2.2, 2.3, 4.5 and 1.1.3, and much of 3.1.

Acknowledgements

I would like to take this opportunity to thank my thesis supervisor for his insights and our stimulating discussions. While some might argue that he was merely doing his job, I am fairly certain that he spent significantly more hours on me than allotted.

While working on my thesis, I had many inspiring discussions with my fellow computer science students. For this I would like to thank Arild Martin Møller Haugstad, Robert Olesen Engdahl, Anders Franz Terkelsen, and Simon Kongshøj, with special thanks going to the last two for reading and nit-picking a draft of my thesis.

Finally, I would like to thank none other than my mother, for suggesting that we try living in Denmark, to which I responded back then, "of all places...". If not for her, I would most likely not have experienced the wonders of this wonderful country, and would not have met the good friends and colleagues I have today.

Willard Þór Rafnsson

Contents

List of Tables

CHAPTER 1

Introduction

Increasingly, our society relies more on the functioning of software systems, as these systems take on the responsibility of more safety-critical tasks. Today, we see examples of software systems being applied to monitor and control nuclear power plants, to assist in maneuvering otherwise unpilotable aircraft, to control medical equipment, and to control subway trains without a human operator. It is therefore of utmost importance that software behaves as expected, to avoid potential catastrophes. Examples of said catastrophes include the failed Ariane-5 launch in 1996, the loss of the NASA Mars Climate Orbiter in 1999, and the Therac-25 radiation therapy machine which caused 6 cancer patients to die from radiation overdose during a two-year period late in the 1980s. The cause of all of these grave mishaps has been traced to unforeseen software behaviour.

Security and correct behaviour is also of great importance in the world of finance, where software errors can result in the loss of millions, or worse, in bankruptcy. Between 1993 and 1994, an error in the baggage delivery system at Denver international airport caused a 9 month delay of the opening of the airport, at the cost of $1.1 million, per day.

For correct operation, software systems often rely heavily on properties guaranteed by various protocols. As such, it is crucial that the correctness of these protocols be verified. Of particular interest in this regard in both research and practice is the correctness of cryptographic protocols[1], which are widely applied to guarantee privacy of information exchange, for instance, during online shopping and when bank records are transferred over the Internet. Clearly, a weakness in such a protocol could have grave consequences. Verifying the correctness of cryptographic prototols has proven to be quite a challenge, however; for instance, a vulnerability in the Needham-Shroeder public-key protocol was published in 1995 [Low95], 17 years after its development.

In particular, surprisingly little research has been done on verifying protocols for se-

[1]Also referred to as security protocols.

cure routing in mobile ad-hoc networks [NH04, God06], even in the dawn of ubiquitous computing [HM05] and the ever-increasing use of communicating mobile devices.

For all these reasons, researching formal methods for modelling software systems to verify beyond doubt that they work as intended is paramount.

1.1 Background

Here we give a brief account on the key subjects which this thesis regards, and hint at the objective of this thesis as we go.

1.1.1 Secrecy and Authenticity

Cryptographic protocols have been around since ancient times, the most famous examples of ancient times being the Caesar's Shift and the Atbash substitution cyphers, applied by Romans and Hebrews, respectively. Substitution cyphers involve displacing the alphabet in a written message by a fixed constant. These "pen-and-paper" codes are, provided an unintended viewer is familiar with the encryption scheme, quite easy to break, particularly by use of modern-day computers.

Since the early 20th century, however, machines and computers have been applied as an aid in the encryption process, making code-breaking significantly more difficult. Today's *encryption primitives* are based on the inherent complexity of some computational problems which, without the proper information, commonly referred to as the *encryption key*, makes code-breaking immensely time-consuming to the point of futility[2].

However, while strong cryptographic primitives are indeed important, the integrity of the cryptographic protocol itself is of at least equal importance. During the second world war, the Allies were capable of decrypting a large number of messages encoded by the German Enigma cipher machines, largely due to the Allies capturing machines and codebooks.

Verification of cryptographic protocols is thus the act of *ensuring that an attacker on the protocol never learns enough information to violate the protocol guarantee.* What we mean by a cryptographic protocol is the agreed method of encrypted or authenticated information exchange. Typically, in protocol verification, *perfect encryption* is assumed, as the task of verifying the integrity of encryption primitives is the subject of another field[3]. The protocol guarantees typically come in two forms:

Secrecy properties express to which extent a given program maintains the secrecy of some information from potential attackers. There are two general types of secrecy properties. *Syntactic secrecy* expresses that a principal A does not directly expose a message M. *Strong secrecy* expresses that, even if A does not expose M, an observer should not be capable to seeing any change in the subsequent behaviour of A for different values of M [CRZ06].

[2]Typically, using today's computers to break modern codes would take longer than the lifespan of *earth*, many times over.

[3]Coding and Information Theory.

Authentication properties express whether a pair of participants A and B are capable of performing authenticated *sessions*. That is, any time B ends a session, then A must have begun the session prior [Bla02]. Typically, this property is obtained by A secretly notifying B of the initiated session (through encrypted message passing, or on a secure communication medium). If the secret applied is compromised, an attacker can notify B of an initiated session, thus violating the property.

So, on one hand, the interest is in analysing *authentication protocols*, which involve "setting up" for future authenticated or encrypted information exchange. This usually involves two participants obtaining a secret key to use for encryption. On the other hand, the interest is in verifying that crucial information is not leaked to the environment, even when using encryption of messages.

When verifying cryptographic protocols, we do so with regards to a *threat model* expressing our assumtions on the capabilities of attackers. A commonly used threat model in this regard is the Dolev-Yao threat model [DY81], proposed in 1981 by Danny Dolev and Andrew C. Yao. In the Dolev-Yao threat model, the communication medium is considered a sophisticated attacker which can intercept, redirect, alter and send new messages based on anything he could deduce from common knowledge and previously sent messages.

In fact, this is the underlying assumption in the *secrecy* definition in Sπ [AG97], and in *static equivalence* in Aπ [AF01]; common to both these concepts is the fact that a message M in principal A is no longer considered secret if there exists an evaluation *context*[4] $C[\cdot]$ such that $C[\cdot]$ learns message M in $C[A]$. These have tried-and-true assumptions, and any protocol which keeps secret in this scenario is indeed robust.

As an example, we model the Needham-Shroeder authenticatino protocol with shared keys in the Burrows-Abadi-Needham logic (BAN logic for short) [AT91, BAN89] — a monumental, albeit much debated [AT91, Syv91, WK96, BM97], framework for formalising and analysing informal protocol specifications, which has revealed subtleties and severe errors in several published protocols [BAN89]. A run of a protocol provides two principals, A and B, with a secure authentication key they can use to sign and share messages, provided they know of each other's existence, that they know a key server S, and trust S to create secure keys. In brief, A asks for and subsequently receives an encryption key from S to use for communicating with B, forwards it to B with an authenticity guarantee, and, lastly, A and B establish trust. This is illustrated in Figure 7.1. There, $A \rightarrow B$ denotes a message passed from A to B, containing what is written after the colon ":". A principal name in a message is merely that principal's GUID[5]. N_a is a *nonce*[6] created by principal A, K_{ab} is a key made for A and B, and $\{X\}_{K_{ab}}$ is message X encrypted under that key. The function of N_b, $a(N_b)$, makes no significant change to N_b; it is merely present so B can tell his N_b message apart from the N_b message sent back to B by A, as B stores sent and received messages in the same pool.

[4]Basically, an arbitrary attacker in the presense of A
[5]Globally Unique IDentifier
[6]A random fresh token used as a freshness guarantee

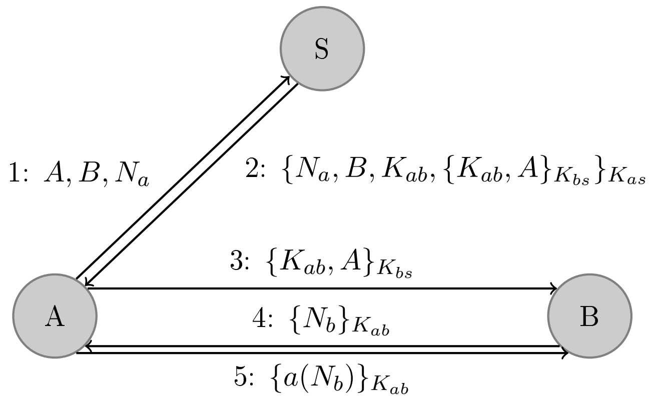
S
A
B
1: A, B, N_a
2: $\{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
3: $\{K_{ab}, A\}_{K_{bs}}$
4: $\{N_b\}_{K_{ab}}$
5: $\{a(N_b)\}_{K_{ab}}$

Figure 1.1: The Needham-Shroeder protocol, illustrated.

When modelled (idealised) in the BAN-logic, a protocol run becomes the following exchange of messages[7].

Message 2	$S \rightarrow A :$	$\{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
Message 3	$A \rightarrow B :$	$\{K_{ab}, A\}_{K_{bs}}$
Message 4	$B \rightarrow A :$	$\{N_b\}_{K_{ab}}$
Message 5	$A \rightarrow B :$	$\{a(N_b)\}_{K_{ab}}$

With this idealisation, a protocol designer/verifier can more easily see which assumptions the protocol makes to work, model these in the BAN-logic language, and subsequently analyse the knowledge inferred by participants. One assumption for the Needham-Shroeder protocol is "B believes fresh($A \overset{\mathrm{K}}{\rightarrow} B$)", meaning B believes that any shared key K_{ab} is fresh. This anomalous assumption is required for the protocol to work, and prior to idealising the protocol in the BAN-logic, the protocol designers were unaware they were making this assumption. It is also this assumption which is the source of the attack discovered by Gavin Lowe mentioned earlier, which involves the re-use of out-dated shared keys. The main reason why this was not discovered by Burrows, Abadi and Needham in 1989 is because of the underlying assumptions of their logic; they assume all participants are trustworthy, and thus do not model attackers on the protocol. While several formalisms have been proposed since [AG97, AF01, AB02], finding formalisms for the verification of cryptographic protocols remains an active field of research.

1.1.2 MANETs

A Mobile Ad-hoc Network, or MANET for short, is a decentralised, self-configuring network of mobile broadcasting nodes, connected in a wireless manner, having an arbitrary network topology. The last remark implies that there is no pre-deployed network infrastructure. Because of this, and the fact that nodes in the network are mobile, routing is a

[7]Message 1 is not present in the idealisation, as it is of no cryptographic relevance in our analysis.

central issue in MANETs, as messages will frequently need to be routed from one node to another through several "hop"s. Frequently, node A will not even know of a route to node B through the arbitrary, ever-changing network topology, and will need to establish one prior to exchanging information with B.

There are several examples of MANETs. One example is the wireless sensor network, which consist of tiny, battery-powered, radio-linked nodes which are distributed in the wilderness to collectively monitor forests for fires, or to search the landscape in search-and-resque operations. Other examples include battlefield surveilance, wireless mesh networks, wireless personal area networks for on-person digital devices, peer-to-peer networks[8], and notebook computers.

Due to the inherent nature of MANETs, these networks are very vulnerable to traffic redirecting attacks, such as denial-of-service- and tunneling-attacks, to spoofing attacks, and attacks using fabricated routing messages. Since most network services abstract from routing of messages, and thus rely on routing protocols for convenience, securing these protocols becomes an utmost importance.

To address this, several *secure routing protocols* have been proposed. These include Ariadne, SAODV, ARAN, and several others which are either standardised, or in the process of being standardised. However, despite their apparent claims of security, the ARAN protocol [God06] and the SAODV protocol [NH06] have been proven faulty.

1.1.3 Process Calculi

The study of process calculi, also referred to as process algebra[9], is a well established field within theoretical computer science, which mainly concerns reasoning about the behaviour of concurrent systems. The field was established in 1982 by Robin Milner, in his book on the Calculus of Communicating Systems (CCS for short) [Mil82]. In CCS, software systems and protocols can be modelled and reasoned about abstractly as processes which communicate through point-to-point synchronisation across named channels.

Some years later, in 1989, Robin Milner, together with Joachim Parrow and David Walker, developed CCS into the π-calculus [MPW92, Mil99]. The π-calculus extends the CCS calculus by adding name-passing and scope-restrictions on names. The effect, and the purpose, of this extension, was to be capable of *expressing mobility*, thus greatly increasing the expressiveness of their calculus. For an example on how the π-calculus expresses mobility, consider the three parallel components P, Q and R, in Figure 1.2, where P and Q share between them the communication channel a, and P and R share the channel b. In π-calculus notation, this can be expressed as follows.

$$P \overset{\text{def}}{=} \overline{b}\langle a\rangle.a.b.\mathbf{0} \quad Q \overset{\text{def}}{=} \overline{a}.a.\mathbf{0} \quad R \overset{\text{def}}{=} b(y).\overline{y}.\overline{b}.\mathbf{0}$$

The mobility of channel a in Figure 1.2 can then be expressed by the following two

[8]Although not wireless, the principles of peer-to-peer networks are the same as that of a MANET.
[9]We shall use these terms interchangeably.

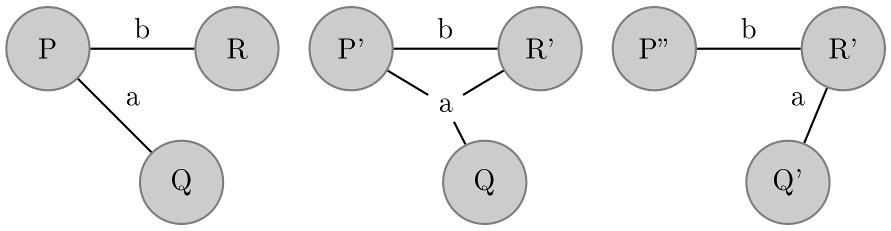

Figure 1.2: Mobility in the π-calculus.

synchronisation steps.

$$\begin{aligned} P \mid Q \mid R = {} & \overline{b}\langle a\rangle.a.b.\mathbf{0} \mid \overline{a}.a.\mathbf{0} \mid b(y).\overline{y}.\overline{b}.\mathbf{0} \\ \rightarrow {} & a.b.\mathbf{0} \mid \overline{a}.a.\mathbf{0} \mid (\overline{y}.\overline{b}.\mathbf{0})\{^{a}/_{y}\} \\ = {} & a.b.\mathbf{0} \mid \overline{a}.a.\mathbf{0} \mid \overline{a}.\overline{b}.\mathbf{0} \;=\; P' \mid Q \mid R' \\ \rightarrow {} & b.\mathbf{0} \mid a.\mathbf{0} \mid \overline{a}.\overline{b}.\mathbf{0} \;=\; P'' \mid Q' \mid R'' \end{aligned}$$

The channel a first becomes a common resource to the three nodes in Figure 1.2, whereafter it becomes accessible only to Q and (the reduced) R. In effect, during these two reduction steps, the nodes have *moved* from P to Q being capable of delivering messages on communication medium a, to Q and R being capable of delivering messages on a.

Although the π-calculus is still used extensively today, the fact that the π-calculus is easily modified has resulted in a vast amount — a whole family — of subcalculi and calculi extensions to the π-calculus, to achieve a framework better suited for a particular scenario [SW01]. Of particular interest in this regard is the Polyadic π-calculus [Mil93] which allows the sending of tuples of names across channels, The locality-extension of the π-calculus by Chothia and Stark [CS01], the Distributed π calculus[10] [HR98] which extends the Polyadic π-calculus with value-passing and explicit locations, or nodes, and the Asynchronous π-calculus [HT91], which is easier to implement and wherein various behavioural equivalences coincide.

A well-established calculus in the context of cryptographic protocols is the Sπ calculus by Martín Abadi and Andrew D. Gordon [AG97]. The Sπ calculus[11] extends the π-calculus by adding cryptographic primitives, thus allowing the formal description and analysis of cryptographic protocols in a process calculus. In the Sπ calculus, security guarantees are expressed as equivalences between Sπ calculus processes. For instance, *secrecy* of some information M can be expressed in terms of *indistinguishability*; protocol P containing information M is said to keep M secret if P is equivalent in the eyes of an arbitrary environment to P with M replaced by any other M'. Interestingly, these added abstractions have been proven to be expressible in the pure π-calculus [BPV05]. The Applied π calculus[12] by Martín Abadi and Cédric Fournet [AF01] generalises the

[10]Sometimes referred to here as Distributed π, or merely Dπ,

[11]Sometimes referred to here as Sπ.

[12]Sometimes referred to here as Applied π, or merely Aπ.

Sπ calculus by giving a uniform extension to the π-calculus in the form of value-passing, functions, equations and conditionals. The result is a calculus where only concurrency-specific actions are modelled using traditional π-calculus abstractions. This shifts the focus of modelling a scenario away from encoding everything to π-calculus abstractions, to providing your own abstractions with which the scenario can be expressed more intuitively. Examples of abstractions easily expressed in Aπ are "let" expressions, pairs (and thereby lists, trees, and any other datastructure), nonces, hashing functions, encoding and decoding primitives, and so on.

Of particular interest to us are calculi related to the Calculus of Broadcasting Systems (CBS for short), which was developed in 1995 by K. V. S. Prasad [Pra95]. CBS differs significantly from CCS in that sending of messages occurs as an asynchronous *broadcast* over a shared medium, such that when one process sends, any number of listening processes can receive. CBS has inspired both CBS$^\sharp$ and CMAN, both of which are scoped broadcast calculi with explicit locations [NH06, God07]. These broadcast calculi are of particular interest to us since pure broadcast primitives cannot be modelled in the π-calculus [EM99].

1.1.4 Static Analysis

Static analysis is the act of analysing a program specification to obtain approximations of the behaviour of the program, without executing the program in question[13]. Traditionally, static analysis has been applied in the construction of efficient implementations of programming languages. More recently, static analysis has been applied for *program verification*, and, in the process calculi setting, security protocol verification. Approaches to static analysis include *type checking*, which involves annotating a language with types, and developing a type system for verifying *type correctness* of a program. The type correctness criteriae vary from setting to setting, ranging from termination guarantee to correct operator use to, in our setting, secrecy and authenticity guarantees [AB02, GJ01, Aba99]. Another approach is *control flow analysis*, which involves static prediction of safe and computable approximations to sets of values which may arise during program execution in functional programming languages [Shi88]. Control flow analysis has been applied to various other programming paradigms, including calculi for concurrency and security [BDNN01].

Recent work has seen an increasing interest in generating *Horn clause constraints* for representing the analysis of cryptographic protocols [BDNN01, Nan06, AB02, BAF08, Bla08]. Generating Horn clause constraints has certain appeal, as Horn clauses are simply formulae in predicate logic, and are the subject of logic programming languages such as Prolog [SS94]. However, the satisfiability problem of Horn clauses in predicate logic is undecidable in general, while it is decidable for Horn clauses in propositional logic. Because of this, much work has been invested in developing automatic solvers which, by imposing some limit to the predicate Horn clauses, can guarantee termination in some

[13]The counterpart of static analysis, *dynamic analysis*, involves analysing simulations of program executions.

select cases, or at least find a solution quickly in most success cases. Various such solvers exist; examples include ProVerif [Bla01], and the Succinct Solver Suite [NNS$^+$04].

1.2 Objective

Our goal is to address the apparent need for formalisms and proof techniques for Mobile Ad-hoc Networks. Specifically, to:

> Develop a framework for verifying secrecy and authenticity properties of (secure) Mobile Ad-hoc Network Protocols.

To achieve this, we set out to:

- Develop a process calculus within which to model security protocols for Mobile Ad-hoc Networks, and
- Provide an automatic verification technique for models in our language. We shall do this by providing a soundness result for deduction from Horn clauses generated from process specification expressed in our new calculus.

Together, these results pave the way for, with minimal effort, to apply automated Horn clause constraint solvers such as ProVerif and the Succinct Solver Suite to reason soundly about the validity of the modelled secure Mobile Ad-hoc Network protocols.

1.3 Related Work

A vastness of results and automated verification techniques for secrecy and authenticity have been published for the Sπ-calculus and the Aπ-like calculi [AG97, Aba99, GJ01, AF01, AB02, AC04, AC05, CRZ06, BAF08, Bla08]. While the inherent lack of a broadcast primitive makes these calculi inappropriate for verification of secure routing protocols for MANETs, then none the less, our work has been greatly inspired by these results. Nanz presents the language CBS$^\sharp$ in [NH06] for modelling secure routing protocols. However, the pattern-matching mechanism for terms therein is inappropriate for accurately expressing cryptographic primitives. In [God07], Godskesen provides CMAN for modelling secure routing protocols for MANETs, wherein verification is performed manually by use of behavioural equivalences.

As for automated verification using Horn clause constraints, Bodei et al apply Horn clause constraints in their analysis in [BDNN01]. In [Nan06], Nanz, using an analysis technique similar to [BDNN01], generates Horn clause constraints for automatic verification of security properties in the Succinct Solver Suite [NNS$^+$04]. Finally, in [AB02, Bla02, BAF08, Bla08], Abadi, Blanchet, and later Fournet, generate Horn clause constraints for automatic verification of secrecy and authenticity in ProVerif [Bla01].

1.4 Outline

In Chapter 2 we present the basic elements of theory on which our work is based, including term rewrite system and unification. Next we provide an overview in Chapter 3 of well-established theory in the subfield of security protocol verification, as well as frameworks for verifying security properties of protocols for MANETs. Notable is the summary in the end of the chapter wherein we compare the presented formalisms, to see which features are desirable in our calculus, and which are not. We then come to our calculus, DAπ_β, in Chapter 4, giving an operational semantics for it, and inhereting several results and definitions from previous work. In Chapters 5 and 6, we prove the key result of soundness of deduction from Horn clauses, with regards to the process the clauses were deduced from (which is a calculus different from DAπ). At last, in Chapters 7, we give the soundness of the Horn clauses generated from a network. In it we omit a few trivialities, for readability, and due to space constraints. Thereafter, we conclude our work, and provide some insight into which directions this project can proceed to.

CHAPTER 2

Preliminaries

We will start off by reviewing the foundation on which our work is based. First, in Section 2.1, we clarify the nonstandard notation applied in this thesis. Sections 2.2 and 2.3 briefly summarise Universal Algebra and Term Rewrite Systems, respectively, both of which are crucial elements in the process calculi reviewed in Chapter 3. In Section 2.4, we explain the concept of syntactic unification, which will be of key importance in our proofs later in the thesis. Next we give the key definitions and results regarding Horn clauses in Section 2.5. Finally, in Sections 2.6 and 2.7, we give the basic definitions of graphs and transition systems, for easy reference.

2.1 Notation

Tuples as lists as sets: We sometimes consider lists as (multi) sets, so that we inherit set theoretic operators to apply on them. For instance, $a, b, b, c, d, e \cap d, e, e, f, g, h = d, e$. When convenient, we also sometimes represent the list a, b, b, c, d, e as the tuple (a, b, b, c, d, e), on which we also apply set-theoretic operators, in particular set-membership (to denote tuple- and list-membership).

Enumerated list $\tilde{x}$: We define the tilde-operator as the enumerated list $\tilde{x} = x_1, \ldots, x_{|\tilde{x}|}$ of finite, context-specific length.

Identifier reservations: When we state that a finite list of syntactic elements, for instance, a, b, c, range over some possibly infinite set, we are "type-declaring" the syntactic elements a, b, and c, *and any annotations thereof*, as being members of that set (within the context of the definition). For instance, when a, b ranges over the set $\mathcal{S}$, then $a, b, a', b', a'', b'', \ldots, a_1, b_1, \ldots a''_3, \ldots b^h, \ldots, a^\gamma_{\text{end}} \cdots \in \mathcal{S}$. In cases where some annotation has a specific meaning, this will be made clear in the text.

Objects as singleton sets: When it improves readability, we consider any object x as a singleton set $\{x\}$, and vice versa.

Relations as functions: Any infix relation operator $\mathcal{R}$ mapping X to X' can be considered a prefix operator $\mathcal{R} : X \longrightarrow X'$.

Functions as sets: When appropriate, we consider a function $\mathsf{f} : X \longrightarrow X'$ as a set $\mathsf{f} \subseteq X \times X'$.

Omitted list concatenation: Sometimes we write $\tilde{x}x$ for $\tilde{x} \cdot x$ (where, here, $\cdot$ denotes list concatenation). Note also that unless defined otherwise, $\mathcal{Y}y$ is the concatenation of y to the list-interpretation of the set $\mathcal{Y}$, that is, resulting in $\mathcal{Y} \cup \{y\}$ (or $\mathcal{Y} \cup y$, when y itself is not a set).

Relation composition: For two relations, $\mathcal{R}$, $\mathcal{R}'$, we define $\mathcal{R} \cdot \mathcal{R}'$ as the relation $\mathcal{R}''$, where

$$\mathcal{R}''(x) = \mathcal{R}'(\mathcal{R}(x)).$$

2.2 Universal Algebra

Universal algebra is the study of commonalities in all algebraic structures. It abstracts the features common to mathematical structures, like groups, rings, vector space and metric spaces, such that they can all be defined as an *algebraic theory*, consisting of a *structure*, and axioms on these in the form of *equational laws*. We will follow the treatment of universal algebra provided in [Joh96]. We start off with (algebraic) structures.

Definition 2.1 (Structure)
A *structure* is a pair (A, O), where A is a set and O is a set of operations on A. $\square$

In literature, a structure is also referred to as an *algebraic structure*, or simply an *algebra*.

Definition 2.2 (Signature)
A *signature* is a pair, (Σ, α), where Σ is a set of operational symbols, and $\alpha : \Sigma \longrightarrow \mathbb{N}$, the arity function, is a total function assigning to each operational symbol in Σ its arity. $\square$

In litterature, a signature is also referred to as an *operational type* (and then often denoted by Ω). Note that we consider 0 to be a natural number. We often omit the arity function α and just write that Σ is a signature.

Definition 2.3 (Σ-structure)
Given a set A and a signature (Σ, α), let $\Sigma_A = \{\omega_A : A^{\alpha(\omega)} \longrightarrow A \mid \omega \in \Sigma\}$ be the *set of interpretations* of all $\omega \in \Sigma$ on A. The pair (A, Σ_A) is then a Σ-*structure*. $\square$

In litterature, a Σ-structure is sometimes refered to as a structure of type (Σ, α), or Σ-algebra. We also refer to Σ_A as an Σ-*structure on the set* A. We sometimes omit Σ_A and just write that A is a Σ-structure.

Definition 2.4 (Σ-terms in X)
Let Σ be a signature, and let X, where $\Sigma \cap X = \emptyset$, be the set of variables. Then the set $F_\Sigma(X)$ of Σ-*terms in* X is the smallest set of finite strings over $\Sigma \cup X$ satisfying the following inductive definition.

a) If $x \in X$, then $x \in F_\Sigma(X)$,

b) If $\omega \in \Sigma$, $\alpha(\omega) = n$ and $t_1, \dots, t_n \in F_\Sigma(X)$, then $\omega t_1 \cdots t_n \in F_\Sigma(X)$. □

Lemma 2.1
For any X,

$$F_\Sigma(X) = \bigcup \{F_\Sigma(X') \mid X' \subseteq X, X' \textit{ is finite}\}.$$

Proof
Trivial, as the union iterates all possible (finite) subsets of X. ■

Definition 2.5 (Derived Operation)
For a given finite set $X_n = \{x_1, \dots, x_n\}$ of variables, a set of Σ-terms $F_\Sigma(X_n)$ and a Σ-structure A, we define for each term $t \in F_\Sigma(X_n)$ its corresponding *derived operation* $t_A : A^n \longrightarrow A$ inductively as follows.

i) if $t = x_i$, where $1 \leq i \leq n$, then t_A is a projection onto the ith factor. That is,

$$t_A(t_1, \dots, t_n) = t_i$$

ii) if $t = \omega t_1 \cdots t_m$, where $\alpha(\omega) = m$, then t_A is the composite

$$A^n \xrightarrow{(t_{1_A}, \dots, t_{m_A})} A^m \xrightarrow{\omega_A} A.$$

That is,

$$t_A(t'_1, \dots . t'_n) = \omega_A(t_{1_A}(t'_1, \dots, t'_n), \dots, t_{m_A}(t'_1, \dots, t'_n)).$$ □

Note that if $t = \omega x_1 \cdots x_n$, where $\alpha(\omega) = n$, then $t_A = \omega_A$. We also call t_A the interpretation in A of the n-ary derived operation corresponding to the term t.

We now turn our attention to equations.

Definition 2.6 (n-ary Equation in Signature Σ)
An n-ary equation in a signature Σ is an expression $(s = t)$, where $s, t \in F_\Sigma(X_n)$. □

Definition 2.7 (Equation Satisfaction in a Structure)
An equation $(s = t)$ is satisfied in a structure A if $s_A = t_A$. □

Note that s_A, t_A are derived operators from the terms s, t, and that the equality $s_A = t_A$ is an equality on function definitions.

Definition 2.8 (Algebraic Theory)
An *algebraic theory* is a pair $T = (\Sigma, E)$, where Σ is a signature, and E the set of equations in Σ. □

Definition 2.9 (Model for T)
A *model* for an algebraic theory $T = (\Sigma, E)$ is a Σ-structure satisfying all the equations in E. □

In litterature, a model of T is also referred to as a T-*algebra*.

Example 2.2 (Groups)
Recall that a group is a set G equipped with a single binary operation $\circ : G \times G \longrightarrow G$ (composition) satisfying the axioms

$$\begin{array}{ll}
\text{(associativity)} & \forall s_1, s_2, s_3 \in G[s_1 \circ (s_2 \circ s_3) = (s_1 \circ s_2) \circ s_3)], \\
\text{(identity)} & \exists e \in G \forall s \in G[s \circ e = s \wedge e \circ s = s], \\
\text{(inverse)} & \forall s \in G \exists i \in G[s \circ i = e \wedge i \circ s = e],
\end{array}$$

where e is the identity of G (also called a neutral element) [Lau05, Definition 2.1.1].

In terms of universal algebra, the definition of a group can be expressed as the algebraic theory $T = ((\Sigma, \alpha), E)$, where $\Sigma = \{\mathsf{c}, \mathsf{i}, \mathsf{e}\}$, $\alpha(\mathsf{c}) = 2$, $\alpha(\mathsf{i}) = 1$, $\alpha(\mathsf{e}) = 0$, and

$$E = \left\{ \begin{array}{c} (\mathsf{c}x_1\mathsf{c}x_2x_3 = \mathsf{cc}x_1x_2x_3), \\ (\mathsf{ce}x_1 = x_1), (\mathsf{c}x_1\mathsf{e} = x_1), \\ (\mathsf{ci}x_1x_1 = \mathsf{e}), (\mathsf{c}x_1\mathsf{i}x_1 = \mathsf{e}) \end{array} \right\}.$$

Any T-algebra is then a group. One example of a T-algebra (and thus a group) is the set $\mathbb{Q}$ with the interpretation of c as $+$. Two other examples are the sets $\mathbb{R}\backslash\{0\}$ and $\mathbb{Q}\backslash\{0\}$ with the interpretation of c as $\cdot$. □

Example 2.3 (Symmetric Key Cryptography)
Expressing an algebraic theory $T = ((\Sigma, \alpha), E)$ for symmetric key cryptography proves to be surprisingly simple: Let $\Sigma = \{\mathsf{e}, \mathsf{d}\}$, where e and d denote encryption and decryption respectively, $\alpha(\mathsf{e}) = 2$, $\alpha(\mathsf{d}) = 2$, and $E = \{\mathsf{de}xyy = x\}$. A typical example of a T-algebra is the infinite set $\mathcal{N} \cup \mathcal{V}$ of channel names and variables in Aπ[1]. An interesting remark here is that $F_\Sigma(\mathcal{N} \cup \mathcal{V})$ is the set of all possible terms in an application of Aπ with Σ being the signature in the application, and E as an equational theory. □

Example 2.4 (Equation Satisfaction)
To see what role Definition 2.7 plays, consider our expression of groups in universal algebra from Example 2.2 before, and consider the $\mathbb{Q}\backslash\{0\}$-structure with c interpreted as $\cdot$. Now consider the term $x_1 \cdot x_2 \cdot x_3$, where $x_1, x_2, x_3 \in \mathbb{Q}\backslash\{0\}$. We know $\cdot$ is associative, so the terms $\tau_1 = x_1 \cdot (x_2 \cdot x_3)$ and $\tau_2 = (x_1 \cdot x_2) \cdot x_3$ are equal. To show that $\mathbb{Q}\backslash\{0\}$ satisfies the first equation in E, we check whether the derived operators $\tau_{1_{\mathbb{Q}\backslash\{0\}}}$ and

[1]Explained in more detail in Chapter 3.

$\tau_{2_{\mathbb{Q}\setminus\{0\}}}$ of τ_1 and τ_2, are equal. Let $X_3 = \{x_1, x_2, x_3\}$. Then $\tau_1, \tau_2 \in F_\Sigma(X_3)$. We get from Definition 2.5 that

$$\tau_{1_{\mathbb{Q}\setminus\{0\}}}(t_1, t_2, t_3) = c_{\mathbb{Q}\setminus\{0\}}\begin{pmatrix} t_1 \\ c_{\mathbb{Q}\setminus\{0\}}(t_2, t_3) \end{pmatrix} = t_1 \cdot (t_2 \cdot t_3)$$
$$\tau_{2_{\mathbb{Q}\setminus\{0\}}}(t_1, t_2, t_3) = c_{\mathbb{Q}\setminus\{0\}}\begin{pmatrix} c_{\mathbb{Q}\setminus\{0\}}(t_1, t_2) \\ t_3 \end{pmatrix} = (t_1 \cdot t_2) \cdot t_3.$$

As $\cdot$ is associative, we get that $\tau_{1_{\mathbb{Q}\setminus\{0\}}} = \tau_{2_{\mathbb{Q}\setminus\{0\}}}$, and thus that $\mathbb{Q}\setminus\{0\}$ satisfies the first equation in E. □

2.3 Term Rewrite Systems

A term rewrite system, also referred to as a rewrite system, or a rewriting system, consists of a set of terms, and a set of rules describing how to reduce said terms. Simplification of logical formulae, set expressions, and algebraic operations can all be performed by use of a term rewrite system. Here we will follow the treatment of term rewrite systems given in [BN98] and [AC04], the latter when we define convergent subterm theories.

Let X be a given (finite or infinite) set, and let $F_\Sigma(X)$ be the set of Σ-terms in X. A *rewrite rule* r is on the form $T_l > T_r$, where $T_l, T_r \in F_\Sigma(X)$ and $\mathsf{v}(T_r) \subseteq \mathsf{v}(T_l)$. We call T_l and T_r the left- and right-hand side of r, respectively, and we refer to the outermost operator symbol in T_l as the *destructor*.

A *term rewrite system* $\mathcal{R}$ is then a set of rewrite rules. A term T_1 *reduces primitively* to T_2 using a rule $r := T_l > T_r$, written $T_1 >_r T_2$, if $T_1 = \sigma T_l$ and $T_2 = \sigma T_2$ for some substitution σ. Furthermore, we say T_1 reduces to T_2, written $T_1 > T_2$ iff there is a rule $r := T_l > T_r$ and a substitution σ for which $T_l = \sigma T_1'$ where $C[T_1'] = T_1$, and $T_2 = C[\sigma T_r]$[2].

We adopt the following notation for $>$. [BN98]

(identity)	$>^0 := \{(x, x) \mid x \in F_\Sigma(X)\}$
(fold composition)	$>^{i+1} :=>^i \circ > \,;\, i \geq 0$
(transitive closure)	$>^+ := \bigcup_{i>0} >^i$
(reflexive transitive closure)	$>^* :=>^+ \cup >^0$
(reflexive closure)	$>^= :=> \cup >^0$
(inverse)	$>^{-1} := \{(y, x) \mid x > y\}$
(symmetric closure)	$<> :=< \cup >$
(transitive closure)	$<>^+ := (<>)^+$
(transitive symmetric closure)	$<>^* := (<>)^*$

[2]here, C is a term missing a substring, such that $C[T]$ is a well-formed term, for any well-formed term T

Definition 2.10 (Terminology)
For terms x, y we have that

i) x is *reducible* if $\exists y[x > y]$ holds.

ii) x is on *normal form*[3] iff it is not reducible.

iii) y is a *normal form of* x iff $x <>^* y$ and y is in normal form. If x has a uniquely determined normal form, we denote it by $x\downarrow$.

iv) y is a *direct successor* of x iff $x > y$.

v) y is a *successor* of x iff $x >^+ y$.

vi) x, y are *joinable* iff $\exists z[x >^* z <^* y]$ holds, in which case we write $x \downarrow y$. □

Definition 2.11 (Reduction Types)
A reduction $>$ is said to be

i) *confluent* iff $y_1 <^* x >^* y_2 \implies y_1 \downarrow y_2$.

ii) *terminating* iff there is no infinite descending chain $a_0 > a_1 > \ldots$.

iii) *normalising* iff every element has a normal form.

iv) *convergent* iff it is confluent and terminating. □

Any terminating relation is normalising, but note that the converse is not necessarily true. Note also that x can be reduced to its normal form ambiguously if there exist y_1, y_2 such that $x >^* y_1$, $x >^* y_2$ and $y_1 \downarrow y_2$. When $y_1 \downarrow y_2$ holds for any two reductions from x via. y_1 and y_2, then $>$ is *confluent*. Note here the usage of $*$, and not $+$.

Example 2.5
The symmetric key cryptographic protocol modelled as an algebraic theory in Example 2.3 easily becomes a term rewrite system by adopting the single equation directly as a rewrite rule: $E_r = \{\mathsf{dec}(\mathsf{enc}(x, y), y) >_{r_1} x\}$. An example equality valid in $=_{E_r}$ is

$$\mathsf{dec}(\mathsf{enc}(\mathsf{enc}(x, y), z), z) =_{E_r} \mathsf{enc}(x, y)$$ □

Lastly, the class of equational theories subject in [AC04], for which deduction and static equivalence is computationally solvable.

Definition 2.12 (Convergent Subterm Theory [AC04])
An equational theory is a convergent subterm theory if it is generated from a convergent term rewrite system $\mathcal{R} = \bigcup_{i=1}^{n}\{t_{l_i} >_{r_i} t_{r_i}\}$, where each t_{r_i} is a proper subterm of t_{l_i}. □

[3] also said to be *irreducible*

2.4 Syntactic Unification

Syntactic unification is central in our work, as it is the heart in tools for deducing truths from a set of Horn clauses, like Prolog interpreters, ProVerif, and the Succinct Solver. Here we will follow the treatment of syntactic unification in [BN98], unless otherwise noted. Here, $=$ will denote the assertion of *syntactic identity*, while $s \overset{?}{=} t$ will be an equation of potential unifiability.

A substitution σ is a mapping from variables in some set $\mathcal{V}$, to terms in $F_\Sigma(\mathcal{V})$. We write $T\sigma$ for the term T with each variable occuring in T synactically replaced by its value in σ, if defined.

Definition 2.13
A substitution σ is *more general than* σ' if there is a substitution σ'' such that $\sigma\sigma'' = \sigma'$. Here, we say σ' is an *instance of* σ, and denote this by $\sigma \lesssim \sigma'$. $\square$

Lemma 2.6
$\lesssim$ *is a* quasi-order *on substitutions.*

Proof
We get that $\lesssim$ is reflexive by letting $\sigma = \mathsf{I}$ (the identity map). To prove transitivity, suppose that $\sigma_2 = \sigma_1'\sigma_1$ and $\sigma_3 = \sigma_2'\sigma_2$. Then $\sigma_3 = \sigma_2'\sigma_2 = \sigma_2'(\sigma_1'\sigma_1) = (\sigma_2'\sigma_1')\sigma_1$ (composition of substitutions is associative). $\blacksquare$

We write $\sigma \sim \sigma'$ if $\sigma \lesssim \sigma'$ or $\sigma' \lesssim \sigma$. We say the substitution ρ is a *renaming* if ρ is a bijection on $\mathcal{V}$ (and thus also on $F_\Sigma(\mathcal{V})$)

Lemma 2.7
$\sigma \sim \sigma'$ *iff there exists a renaming* ρ *st.* $\sigma = \rho\sigma'$. $\square$

Definition 2.14 (Unification)
A *unification problem* is a finite set of equations $S = \{t_1 \overset{?}{=} t_1', \ldots, t_n \overset{?}{=} t_n'\}$. A *unifier*, or *solution*, of S is a substitution σ such that $t_i = t_i'; 1 \leq i \leq n$. $\mathcal{U}(S)$ is the set of all unifiers of S. S is *unifiable*, or *solvable*, if $\mathcal{U}(S) \neq \emptyset$. $\square$

Example 2.8
Consider the simple unification problem

$$S \overset{\text{def}}{=} \{\mathsf{enc}(\mathsf{pair}(x,y),z) \overset{?}{=} \mathsf{enc}(\mathsf{pair}(a,\mathsf{h}(a)),b)\},$$

with unification variables $\{x, y, z\}$. A solution to this problem is the substitution map $\sigma = \{x \mapsto a, y \mapsto \mathsf{h}(a), z \mapsto b\}$, as $(\mathsf{enc}(\mathsf{pair}(x,y),z))\sigma = \mathsf{enc}(\mathsf{pair}(a,\mathsf{h}(a)),b)$. $\square$

Definition 2.15
A substitution σ is *idempotent* if $\sigma = \sigma\sigma$. $\square$

We wish to avoid nonidempotent substitutions, as these are cumbersome in practice. To identify nonidempotent substitutions, we use the following lemma.

Lemma 2.9
σ is idempotent iff $\mathrm{dom}(\sigma) \cap \bigcup_{T \in \mathrm{im}(\sigma)} \mathsf{v}(T) = \emptyset$. $\square$

Now this theorem states that never need to consider nonidempotent substitutions, as our unifiers are idempotent.

Theorem 2.10
If a unification problem S has a solution σ, then σ is an idempotent most general unifier. $\square$

That is, most general unifiers are *unique* only up to renaming. We typically identify α-equivalent most general unifiers. This justifies their name, as any other less-general unification must be an instance of the most general unifier.

We are now ready to present the unification algorithm.

Definition 2.16
A unification problem $S = \{x_1 \overset{?}{=} t_1, \ldots, x_2 \overset{?}{=} t_n\}$ is in *solved form* if the x_i are pairwise distinct variables, none of which occur in any t_j. In this case we define $\overline{S} = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$. $\square$

Lemma 2.11
If S is in solved form then $\sigma = \sigma\overline{S}$, for all $\sigma \in \mathcal{U}(S)$.

Proof
Let $S = \{x_1 \overset{?}{=} t_1, \ldots, x_2 \overset{?}{=} t_n\}$. We show by case distinction that $\forall x \in \mathcal{V}. \sigma x = \sigma\overline{S}x$.

i) $x = x_k; 1 \leq k \leq n$: Then $\sigma x = \sigma t_k = \sigma\overline{S}t_k$, as $\sigma \in \mathcal{U}(S)$.

ii) $x \notin \tilde{x}$: Since $\overline{S}x = x$, we have $\sigma x = \sigma\overline{S}x$. ∎

Lemma 2.12
If S is in solved form, then $\overline{S}$ is an idempotent most general unifier of S.

Proof
Idempotence follows directly from Lemma 2.9, as, since S is on closed form, no x_i occurs in a t_j. For the same reason we get $\overline{S}x_i = t_i$, that is, $\overline{S} \in \mathcal{U}(S)$. Finally, $\overline{S}$ is a most general unifier because by Lemma 2.11, $\overline{S} \lesssim \sigma$ for all $\sigma \in \mathcal{U}(S)$. ∎

Now we can extract idempotent most general unifiers from an S on solved form. We conclude with Algorithm 2.1, which transforms any S into solved form.

2.5 Horn Clauses

Here we shall follow the treatment of Horn clauses in [Bur98], until we reach SLD-Resolution, which we have from [SS94, Fer04]. First, we look at Horn clauses in the propositional logic setting.

Algorithm 2.1: Unification algorithm, $\mathsf{U}(S)$.

```
1 Input: A unification problem, $S$
2 Result: $S$ on solved form, $\overline{S}$, if $S$ is unifiable. Failure otherwise.
3 Recursively apply the following transformation rules in a pattern-matching manner
  until none can be applied. If a failure is reached, the algorithm terminates with an
  appropriate error.
```

$$
\begin{aligned}
\{\mathsf{f}(s_1,\dots,s_n) \overset{?}{=} \mathsf{f}(t_1,\dots,t_n)\} \cup E &\to \{s_1 \overset{?}{=} t_1,\dots,s_n \overset{?}{=} t_n\} \cup E \\
\{\mathsf{f}(s_1,\dots,s_n) \overset{?}{=} \mathsf{g}(t_1,\dots,t_m)\} \cup E &\to \textbf{failure} \\
\{x \overset{?}{=} x\} \cup E &\to E \\
\{T \overset{?}{=} x\} \cup E &\to \{x \overset{?}{=} T\} \cup E \\
\{x \overset{?}{=} T\} \cup E &\to \{x \overset{?}{=} T\} \cup E,\ \text{if } x \notin \mathsf{v}(T) \wedge x \in \mathsf{v}(E) \\
\{x \overset{?}{=} T\} \cup E &\to \textbf{failure} \text{ if } x \in \mathsf{v}(T) \wedge x \neq T
\end{aligned}
$$

```
  This yields $\overline{S}$.
4 return $\overline{S}$.
```

2.5.1 Propositional Logic Setting

Definition 2.17 (Clause)
We let *literals* be ranged over by l, defined by the grammar

$$l ::= p \mid \neg p,$$

where p is a *propositional variable.* Finite sets of literals, $\{l_1,\dots,l_n\}$, are called *clauses.* The *literals of a clause* are the members of the clause. □

Traditionally, the clauses are on the form $\bigvee_{1\leq i\leq n} l_i$ for the clause $\{l_i\}_{1\leq i\leq n}$, but by representing them as a set, we ignore repetition and orderings amongst literals.

Definition 2.18 (Satisfiability)
A clause $\{l_i\}_{1\leq i\leq n}$ is satisfiable by a truth evaluation e iff the formula $\bigvee_{1\leq i\leq n} l_i$ is satisfiable by e. A set $\mathcal{S}$ of clauses is satisfiable iff there is a truth evaluation e that satisfies each clause in $\mathcal{S}$. By definition, $\emptyset$ is not satisfiable. □

We let the literal complement function $\bar{\cdot}$ be defined as

$$\bar{l} = \begin{cases} p, & \text{if } l = \neg p \\ \neg p, & \text{if } l = \end{cases} .$$

Note that these identities are syntactic.

Definition 2.19 (Resolution)
Resolution is defined by the inference rule

$$\text{(resolution)} \qquad \frac{cl_1 \cup \{l\} \qquad cl_2 \cup \{\bar{l}\}}{cl_1 \cup cl_2},$$

where cl_1, cl_2 are clauses. We say we are *resolving* cl_1, cl_2 *over* l, and that $cl_1 \cup cl_2$ is the *resolvent.* □

Example 2.13
A resolution derivation of the empty clause $\emptyset$ from

$$\mathcal{S} = \{\{\neg p, q\}, \{\neg q, \neg r, s\}, \{p\}, \{r\}, \{\neg s\}\}$$

is

1.	$\{\neg s\}$	given
2.	$\{\neg q, \neg r, s\}$	given
3.	$\{\neg q, \neg r\}$	resolvent of resolution (1, 2) over s
4.	$\{r\}$	given
5.	$\{\neg q\}$	resolvent of resolution (3, 4) over r
6.	$\{\neg p, q\}$	given
7.	$\{\neg p\}$	resolvent of resolution (5, 6) over q
8.	$\{p\}$	given
9.	$\emptyset$	resolvent of resolution (7, 8) over p

□

Lemma 2.14 (Resolution preserves Satisfiability)
If $\mathcal{S}$ is satisfiable by truth evaluation e, *then any resolvent of any pair of clauses in $\mathcal{S}$ is satisfiable by* e. □

Theorem 2.15 (Soundness and Completeness of Resolution)
A nonempty set $\mathcal{S}$ of clauses is unsatisfiable *iff there is a derivation of the empty clause $\emptyset$ using* (resolution). □

Definition 2.20 (Horn Clause)
A *Horn clause* is a clause with at most one positive literal. □

For instance, $\{\neg p, \neg q, \neg r, \neg s\}$ and $\{p, \neg q, \neg r, \neg s\}$ are Horn clauses, while $\{p, \neg q, r, \neg s\}$ is not.

Traditionally, Horn clauses which have exactly one positive literal, $\{\neg l_i\}_{1 \le i \le n} \cup \{l\}$, which as explained before can be written as $l \vee \bigvee_{1 \le i \le n} \neg l_i$, are written on the form $\bigwedge_{1 \le i \le n} l_i \implies l$; these formulae are logically equivalent.

Lemma 2.16
A resolvent of two Horn clauses is always a Horn clause. □

Definition 2.21 (Unit Resolution)
A *unit clause* is a clause $\{l\}$ with a single literal. *Unit resolution* refers to resolution derivations in which at least one of the clauses being resolved is a unit clause. □

Theorem 2.17
Unit resolution is sound and complete for Horn clauses. □

This theorem follows from the important fact that if we *cannot* derive the empty clause $\emptyset$ by unit resolution from some set $\mathcal{S}$ of Horn clauses, then $\mathcal{S}$ is satisfiable, that is, there exists a truth evaluation e satisfying $\mathcal{S}$.

Clearly, the advantage of unit resolution is that the *resolvents* never grow in size. Unit resolution on Horn clauses thus always terminates, and is in fact polynomial-time computable.

Theorem 2.18
Resolution of Horn clauses in propositional logic is polynomial-time decidable. □

We now move on to the *predicate logic* setting.

2.5.2 Predicate Logic Setting

The basic definitions, like that of a clause and a Horn clause remain unchanged in this setting, with the exception of a *literal.* Recall that *literals* in predicate logic are not propositional variables, but *atomic formulae*, which are strings on the form $rt_1 \cdots t_n$, where t_i are terms in the algebra of the predicate logic (specified, for instance, as a boolean equational logic in universal algebra), and r is a relational symbol of arity n.

We say a Horn clause is a *definite clause* if it has exactly one positive literal. We call it a *goal clause* if it has no positive literal.

Definition 2.22 (SLD Resolution)
SLD resolution (Selective Linear resolution for Definite clauses) is defined by the inference rule

$$\text{(SLD)} \qquad \frac{\{\neg l_i\}_{1\leq i\leq n} \cup \{\neg l\} \qquad \{\neg l'_j\}_{1\leq j\leq m} \cup \{l'\}}{(\{\neg l_i\}_{1\leq i\leq n} \cup \{\neg l'_j\}_{1\leq j\leq m})\sigma} \text{ where } \quad \sigma \text{ unifies } \{l \overset{?}{=} l'\}$$ □

Notice that the first premise in (SLD) is a goal clause, while the second is a definite clause.

Given a set of definite clauses $\mathcal{S}$, and a goal clause $cl_{\text{goal}} = \{\neg l_i\}_{1\leq i\leq n}$, determining whether cl_{goal} is satisfiable in $\mathcal{S}$ follows the following SLD resolution procedure: Pick a definite clause $\{\neg l'_j\}_{1\leq j\leq m} \cup \{l'\} \in \mathcal{S}$ which (only) positive literal l' *unifies* with some select literal l_i, where $\neg l_i \in cl_{\text{goal}}$. Call this unifier σ. This operation results in a new goal clause $cl'_{\text{goal}} = (\{\neg l_1, \ldots, \neg l_{i-1}, \neg l_{i+1}, \ldots, \neg l_n\} \cup \{\neg l'_j\}_{1\leq j\leq m})\sigma$. Repeat this procedure until you reach the unsatisfiable goal clause $\emptyset$. Then compose the computed unifiers into one big unifier σ_{solution}, which becomes the solution to cl_{goal} in $\mathcal{S}$.

Note, however, that this process can reach a goal clause cl_{fail} wherein there are no definite clauses in $\mathcal{S}$ which positive literal unifies with a literal in the goal clause. In this case, the procedure “backtracks” until it reaches a point where it can select a *different* definite clause.

Again, note that this process is *not guaranteed to terminate!*[4] Consider, for instance,

$$\begin{aligned} cl_{\text{goal}} &\overset{\text{def}}{=} p \\ \mathbb{S} &\overset{\text{def}}{=} \{\{\neg p, p\}\}, \end{aligned}$$

where p is a relational constant (arity 0). With the empty substitution, the SLD Resolution algorithm will, by selecting the (only possible) definite clause $\{\neg p, p\}$, reach the “new” goal clause $cl'_{\text{goal}} = p$. The analogous program specification in Prolog is

```
p :- p.
p.
```

This example can function as a counterexample for the proof of the following.

Theorem 2.19
Resolution of Horn clauses in predicate logic is undecidable in general. □

On a last note, we define derivation of a predicate from a set of Horn clauses as follows.

Definition 2.23 (Derivation from Horn clauses)
Given a set of Horn clauses $\mathbb{S}$ and a predicate p, We say that p is *derivable from* $\mathbb{S}$ if the goal clause p is satisfiable in $\mathbb{S}$. □

2.6 Graph Theory

We present here some of the basic definitions and notation of graph theory, which will be of use later in this thesis. We follow the treatment in [Die06].

Definition 2.24 (Graph)
A graph is a pair (V, E), where V is a set, called the set of vertices, and E, the set of edges, is defined by

- $E \subseteq V^2$, in which case G is a *directed graph*, or
- $E \subseteq \{v \subseteq V \mid v \neq \emptyset \wedge |v| \leq 2\}$, in which case G is an *undirected graph*.

For a graph $G = (V_G, E_G)$, we define $\mathsf{V}(G) = V_G$ and $\mathsf{E}(G) = E_G$. □

Definition 2.25 (Subgraph, Supergraph)
Let $G = (V, E)$, $G' = (V', E')$ be graphs. G' is a subgraph of G (and G a supergraph of G' if $V' \subseteq V$ and $E' \subseteq E$. □

[4] In fact, if resolutions always terminated, then Prolog would not be Turing-complete.

Definition 2.26 (Path)
A path is a nonempty graph $P = (V, E)$ on the form $V = \{x_0, \ldots, x_k\}$ and $E = \{x_i x_{i+1} \mid 0 \leq i \leq k-1\}$. □

Definition 2.27 (Complete graph)
A graph G is *complete* when all vertices in G are connected. That is, when G is undirected,

$$\forall l, n \in \mathsf{V}(G) \,.\, \exists \{l, n\} \in \mathsf{E}(G) \wedge \{l\}, \{n\} \in \mathsf{E}(G)$$

holds, and when G is directed,

$$\forall l, n \in \mathsf{V}(G) \,.\, \exists (l, n) \in \mathsf{E}(G)$$

holds. □

2.7 Labelled Transition System

Transition systems are a clear and concise way of specifying operational semantics of process calculi. As this will be done in the next two chapters, the definitions of labelled and unlabelled transition systems merit a recap.

The basic definition of a transition system comes from [Hü05].

Definition 2.28 (Transition System)
A *transition system* is a pair $(\mathscr{P}, \rightarrow)$, where $\mathscr{P}$ is a set of *states*, ranged over by s, and $\rightarrow \subseteq \mathscr{P} \times \mathscr{P}$ is a transition relation. □

Here, we call s a *final state*, if there exists no s' such that $s \rightarrow s'$.

The remaining definitions are as defined in [AILS07].

Definition 2.29 (Labelled Transition System)
A *labelled transition system* is a triple $(\mathscr{P}, \mathbf{A}, \{\xrightarrow{a} \mid a \in \mathbf{A}\})$, where $\mathscr{P}$ is a set of *states*, ranged over by s, $\mathbf{A}$ is a set of *actions* ranged over by a, and $\xrightarrow{a} \subseteq \mathscr{P} \times \mathscr{P}$ is a transition relation, for every $a \in \mathbf{A}$. □

When $\xrightarrow{\alpha} \subseteq (\mathscr{P} \times \mathbf{A}) \times \mathbf{A}$, we will simply write that $(\mathscr{P}, \mathbf{A}, \xrightarrow{\alpha})$ is a labelled transition system. We write $s \xrightarrow{a} s'$ when $(s, s') \in \xrightarrow{a}$ (and when $((s, a), s') \in \xrightarrow{\alpha}$). We write $s \stackrel{a}{\not\rightarrow} s'$ when $(s, s') \notin \xrightarrow{a}$ (and when $((s, a), s') \notin \xrightarrow{\alpha}$), and in that case, say s *refuses* s'.

Definition 2.30 (Strong Bisimilarity)
A binary relation $\mathcal{R}$ over the set of states of a labelled transition system is a *simulation* iff whenever $s_1 \mathcal{R} s_2$ and a is an action, when it holds that

$$s_1 \xrightarrow{a} s_1' \implies \exists s_2' \,.\, s_2 \xrightarrow{a} s_2' \wedge s_1' \mathcal{R} s_2'.$$

$\mathcal{R}$ is a *bisimulation* if $\mathcal{R}^{-1}$ is a simulation as well. s and s' are *bisimilar*, written $s \sim s'$, if there is a bisimulation that relates them. Thus, $\sim$ is the *largest bisimulation*. □

We denote the unobservable action by τ. We define

$$(\overset{a}{\Longrightarrow}) \overset{\text{def}}{=} (\xrightarrow{\tau}^{*} \cdot \xrightarrow{a} \cdot \xrightarrow{\tau}^{*})$$

Definition 2.31 (Weak Bisimilarity)
A binary relation $\mathcal{R}$ over the set of states of a labelled transition system is a *weak simulation* iff whenever $s_1 \mathcal{R} s_2$ and a is an action, when it holds that

$$s_1 \xrightarrow{a} s_1' \implies \exists s_2' \,.\, s_2 \overset{a}{\Longrightarrow} s_2' \wedge s_1' \mathcal{R} s_2'.$$

$\mathcal{R}$ is a *weak bisimulation* if $\mathcal{R}^{-1}$ is a weak simulation as well. s and s' are *weakly bisimilar* (*observationally equivalent*), written $s \approx s'$, if there is a bisimulation that relates them. Thus, $\approx$ is the *largest weak bisimulation.* □

CHAPTER 3

Related Protocol Analysis Frameworks

In this chapter we present the key results to which the work of this thesis relates, and subsequently reflect upon it, considering how adequate they are for the purpose of modelling secure routing protocols for MANETs. We begin in Section 3.1 with the Applied π calculus of Abadi and Fournet, by which our framework is greatly inspired. Here we will be very thorough in explaining various concepts, as we shall subsequently assume these to be known when we reach the other calculi. We then move on to the closely-related calculus in Section 3.2 by Abadi and Blanchet, which we call ABπ in this thesis. Next we switch over towards the two broadcast calculi, CBS$^\sharp$ and CMAN, in Sections 3.3 and 3.4, respectively, noting carefully the semantics of these two different framework proposals.

To elevate readability, we take the liberty of altering the syntax in the presented calculi such that conceptually similar syntactic element resemble each other syntactically across the various calculi presented in this chapter.

3.1 The Applied π Calculus

We now present the highlights of the Applied π calculus. We will start off by giving the syntax of Aπ and explaining the various terminology. Next we will see the semantics of Aπ in the form of congruence rules and reduction relations. Following this, some proof techniques to reason about processes expressed in Aπ are given. Finally we present the definitions of secrecy we typically see in work on the Applied π calculus.

The semantics definitions in terms of transition systems are ours. Everything else, unless otherwise noted, comes from [AF01].

3.1.1 Syntax and Semantics

Syntax

Let $\mathcal{N}$ be the set of names ranged over by $a, \ldots, c, m, \ldots, t$, and $\mathcal{V}$ be the set of variables ranged over by $x, \ldots, z$ (both sets being infinite). Let $\mathcal{N} \cup \mathcal{V}$ be the set of identifiers ranged over by $u, \ldots, w$. The syntax of Aπ is then given by the grammar specification in Table 3.1[1].

$$
\begin{array}{rcl@{\qquad\qquad}rcl}
P & ::= & \mathbf{0} & A & ::= & P \\
 & | & u(x).P & & | & A \mid A \\
 & | & \overline{u}\langle T\rangle.P & & | & (\nu u)A \\
 & | & P \mid P & & | & \{T/x\} \\
 & | & !P & & & \\
 & | & (\nu a)P & T & ::= & u \\
 & | & \mathbf{if}\ T = T\ \mathbf{then}\ P\ \mathbf{else}\ P & & | & \mathsf{f}(\tilde{T})
\end{array}
$$

Table 3.1: Syntax of Aπ.

The syntax of Aπ is broken down into three syntactic categories, and we now consider them all separately.

Terms: Recall from Section 2.2 that a signature is a finite set of function symbols, or operators, each assigned an arity. The set of terms $\mathcal{T}$, which elements are ranged over by $T, \ldots, V$, is then defined by the syntactic category T, where, for any term on the form $\mathsf{f}(T_1, \ldots, T_k)$, we have $\alpha(\mathsf{f}) = k$. In universal algebra terms, the set of terms becomes the set $F_\Sigma(\mathcal{N} \cup \mathcal{V})$ of Σ-terms in $\mathcal{N} \cup \mathcal{V}$, being the least set of strings over $\Sigma \cup \mathcal{N} \cup \mathcal{V}$ satisfying the constraints in Definition 2.4.

As Aπ has implemented value-passing which performs exactly like name-passing, one could imagine problems arising, for instance, when a process receives a term which is the number 5, instead of an expected channel name, and attempts to synchronise over the received term. For this reason, the Applied π calculus relies on a Milner-like *sort system* [Mil99]. It consists of the pair (T, Γ). Here, $\mathbf{T}$ is a basic set of *types*, consisting of for instance the type `Integer`, `Key`, `Data` for any data, and `Channel`$\langle\tau\rangle$, where τ is any basic type. Γ, however, is a mapping $\Gamma : \mathcal{N} \longrightarrow \mathbf{T}$ associating each name with its sort. For instance, $\Gamma(a) =$ `Channel`$\langle$`Data`$\rangle$ means that it is only possible to pass other channels of any type across a (for instance, channels of type `Integer`. You can even pass a across itself.). We deem it sufficient to know of the presence of the sort system, and shall therefore not consider it in further detail for the remainder of this thesis.

Each application of Aπ is equipped with an *algebraic theory*[2], (Σ, E), consisting of

[1]The observant reader will notice that the syntax presented here deviates slightly from the syntax originally provided in [AF01]; restrictions are represented in the traditional paranthesised manner, as opposed to being terminated by a full stop ".".

[2]Sometimes called *equational theory*

operators, and equations on these. These operators need to be defined on all possible terms in $\mathcal{T}$, and need to satisfy the equations in E. In Universal Algebra terms, we observe that we can accomplish this as follows: Given an algebraic theory, (Σ, E), *interpret* Σ on the set of Σ-terms in $\mathcal{N} \cup \mathcal{V}$, $F_\Sigma(\mathcal{N} \cup \mathcal{V})$, in such a manner that the obtained Σ-*structure* $(F_\Sigma(\mathcal{N}\cup\mathcal{V}), \Sigma_{F_\Sigma(\mathcal{N}\cup\mathcal{V})})$ satisfies the equations in E. The result is a $(F_\Sigma(\mathcal{N}\cup\mathcal{V}), \Sigma_{F_\Sigma(\mathcal{N}\cup\mathcal{V})})$-*algebra on terms*. That is, the operators in Σ, defined on terms, satisfying the equations in E.

Primitive Processes: Primitive processes are represented by the syntactic category P, and ranged over by $P, \ldots, R \in \mathscr{P}$. The syntax specification of a primitive process is identical to that of an ordinary process in the π-calculus, with the exception of primitive processes being capable of performing a *match*. A short description of the syntax follows.

1) $\mathbf{0}$: the inactive null process.
2) $u(x).P$: receive a term T over u, and binds it to the new, private variable x, and then proceed as P with T bound to x.
3) $\overline{u}\langle T\rangle.P$: outputs the term T on channel u. Then behave like P.
4) $P \mid Q$: a parallel composition (concurrent execution) of two primitive processes.
5) $!P$: an infinite number of copies of P running in parallel.
6) $(\nu n)P$: a process which introduces a new, private name n, and then behaves as P.
7) **if** $T = U$ **then** P **else** Q: a conditional construct, which will behave as P if the equality $T = U$ holds, and as Q otherwise.

Extended Processes: Last and most important are extended processes, represented by the syntactic category A and ranged over by $A, B \in \mathscr{A}$. Extended processes are primitive processes extended with the notion an *active substitution*, which plays a crucial role in Aπ. An active substitution $\{^{T}/_{x}\}$ functions as a "let"[3], enabling access to the value T of the variable x, to *everything* within the scope of the variable x. We often write a parallel composition of active substitutions $\prod_{i=1}^{k}\{^{T_i}/_{x_i}\}$ into one active substitution $\{^{T_1}/_{x_1}, \ldots, ^{T_k}/_{x_k}\}$, or $\{^{\tilde{T}}/_{\tilde{x}}\}$ for short, and often denote such a parallel composition as σ. For a more formal treatment, we can consider an active substitution as a total function $\sigma : \mathcal{V} \longrightarrow F_\Sigma(\mathcal{N} \cup \mathcal{V})$ mapping variables to terms in the following manner

$$\sigma(x) = \begin{cases} T_i & \text{if } \exists x_i \in \text{dom}(\sigma)[x_i = x] \\ x & \text{otherwise,} \end{cases}$$

where dom(σ) and im(σ)[4] denote the domain and the image of σ, respectively. To aid in the reasoning of Aπ processes, it is assumed that active substitutions are cycle-free,

[3]See Example 3.1 for how an actual "let" is expressed in Aπ.
[4]These will be defined more formally in Section 3.1.2

that there is at most one active substitution for each variable, and that there is exactly one active substitution within the scope of a restricted variable. We write $\mathsf{fv}(A)$, $\mathsf{bv}(A)$, $\mathsf{fn}(A)$, $\mathsf{bn}(A)$, $\mathsf{fu}(A)$, and $\mathsf{bu}(A)$ for the sets of free and bound variables, free and bound names, and free and bound identifiers in A, respectively We define $\mathsf{fn}(A)$ inductively by

$$\begin{aligned}
\mathsf{fn}(\mathbf{0}) &= \emptyset\\
\mathsf{fn}(A \mid B) &= \mathsf{fn}(A) \cup \mathsf{fn}(B)\\
\mathsf{fn}(!P) &= \mathsf{fn}(P)\\
\mathsf{fn}((\nu a)P) &= \mathsf{fn}(P)\backslash\{a\}\\
\mathsf{fn}(\textbf{if } T = U \textbf{ then } P \textbf{ else } Q) &= \mathsf{n}(T) \cup \mathsf{n}(U) \cup \mathsf{fn}(P) \cup \mathsf{fn}(Q)\\
\mathsf{fn}(a(x).)P &= \{a\} \cup \mathsf{fn}(P)\\
\mathsf{fn}(\overline{a}\langle T\rangle.P) &= \{a\} \cup \mathsf{fn}(T) \cup \mathsf{fn}(P)\\
\mathsf{fn}(\{{}^{T}/_{x}\}) &= \mathsf{fn}(T),
\end{aligned}$$

and $\mathsf{bn}(A)$ by,

$$\begin{aligned}
\mathsf{bn}(\mathbf{0}) &= \emptyset\\
\mathsf{bn}(A \mid B) &= \mathsf{bn}(A) \cup \mathsf{bn}(B)\\
\mathsf{bn}(!P) &= \mathsf{bn}(P)\\
\mathsf{bn}((\nu a)P) &= \{a\} \cup \mathsf{bn}(P)\\
\mathsf{bn}(\textbf{if } T = U \textbf{ then } P \textbf{ else } Q) &= \mathsf{bn}(P) \cup \mathsf{bn}(Q)\\
\mathsf{bn}(a(x).P) &= \mathsf{bn}(P)\\
\mathsf{bn}(\overline{a}\langle T\rangle.P) &= \mathsf{bn}(P)\\
\mathsf{bn}(\{{}^{T}/_{x}\}) &= \emptyset.
\end{aligned}$$

$\mathsf{fv}(A)$ and $\mathsf{bv}(A)$ are then defined in a similar manner, and $\mathsf{fu}(A)$, $\mathsf{bu}(A)$ are defined in the obvious manner. We define the functions

$$\begin{aligned}
\mathsf{n}(A) &= \mathsf{fn}(A) \cup \mathsf{bn}(A)\\
\mathsf{v}(A) &= \mathsf{fv}(A) \cup \mathsf{bv}(A)\\
\mathsf{u}(A) &= \mathsf{fu}(A) \cup \mathsf{bu}(A),
\end{aligned}$$

which represent all the names, variables, and identifiers occurring in a process, respectively.

We say that an extended process is *closed* when every variable occurring in it is either bound (in which case it is either yet to be instantiated, or defined by an active substitution) or defined by an active substitution. This is another way of saying that any variable name in use must have a value which is apparent in the syntax specification of the process. We consider only these kind of extended processes from now on.

Context: At last, we shall frequently make use of the concept of *contexts*. Essentially, a context is a process with a "hole" in it, that is, a process with a parameterised subtree.

Non-Evaluation Contexts:

$$\begin{array}{rcl} C[R] & ::= & C[R] \mid A \\ & \mid & A \mid C[R] \\ & \mid & (\nu u)C[R] \\ & \mid & C'[R] \\ C'[R] & ::= & R \\ & \mid & C'[R] \mid P \\ & \mid & P \mid C'[R] \\ & \mid & (\nu a)C'[R] \\ & \mid & !C'[R] \\ & \mid & \textbf{if } T = T \textbf{ then } C'[R] \textbf{ else } P \\ & \mid & \textbf{if } T = T \textbf{ then } P \textbf{ else } C'[R] \\ & \mid & u(x).C'[R] \\ & \mid & \overline{u}\langle T\rangle.C'[R] \end{array}$$

Evaluation Contexts:

$$\begin{array}{rcl} C[B] & ::= & B \\ & \mid & C[B] \mid A \\ & \mid & A \mid C[B] \\ & \mid & (\nu u)C[B] \end{array}$$

Table 3.2: Contexts of Aπ.

Table 3.2 contains our grammar specification of a context. Contexts come in two variants: *evaluation context*, and *non-evaluation contexts*. In evaluation contexts, the "hole" may not be prefixed by actions of any form (input, output, replication, and conditionals). Since only primitive processes are allowed to be prefixed with these actions, then any context designed to be parameterised with extended processes is an evaluation context. Any such a context can also be parameterised with primitive processes. It isn't until the intent is to prefix the hole with actions that the non-evaluation contexts syntactic category, which enables prefixing the hole with actions, becomes relevant.

A context $C[\cdot]$ is said to *close* A when $C[A]$ is closed.

Semantics

Next we give an overview of the semantics of Aπ. It is defined in terms of an *operational semantics*, expressed in terms of a *structural equivalence* relation, and an *internal reduction* relation. A *labelled operational semantics* has also been presented in [AF01], to reason about processes interacting to their surroundings. We shall summarise all these in the subsequent paragraphs.

Structural Equivalence: The structural equivalence relation for Aπ is defined as follows.

Definition 3.1 (Structural Equivalence)
Structural Equivalence, $\equiv$, is the smallest equivalence relation on $\mathscr{A}$ that is closed by α-conversion on both names and variables, by application of evaluation contexts, and satisfying the equalities in Table 3.3. $\square$

(par-$\mathbf{0}$)	$A \equiv A \mid \mathbf{0}$
(par-A)	$A \mid (B \mid C) \equiv (A \mid B) \mid C$
(par-C)	$A \mid B \equiv B \mid A$
(repl)	$!P \equiv P \mid !P$
(new-$\mathbf{0}$)	$(\nu n)\mathbf{0} \equiv \mathbf{0}$
(new-C)	$(\nu u)(\nu v)A \equiv (\nu v)(\nu u)A$
(new-par)	$A \mid (\nu u)B \equiv (\nu u)(A \mid B)$, if $u \notin \mathsf{fv}(A) \cup \mathsf{fn}(A)$
(alias)	$(\nu x)\{^T/_x\} \equiv \mathbf{0}$
(subst)	$\{^T/_x\} \mid A \equiv \{^T/_x\} \mid A\{^T/_x\}$
(rewrite)	$\{^T/_x\} \equiv \{^U/_x\}$, if $\Sigma \vdash T =_E U$

Table 3.3: Structural equivalence in Aπ.

The (par)-rules express that $\equiv$ satisfies the commutative monoid laws w.r.t. $(|, \mathbf{0})$. (repl) expresses precicely the definition of replication in the syntax specification. The rules for restriction are fairly self-explanatory. (alias) expresses that an active substitution which scope encompasses only itself is equivalent with the null process, meaning that arbitrary active substitutions can be added to a process (and expired ones removed). (subst) expresses how an active substitution affects everything in contact with it[5]. The rules (alias) and (subst) play a key role in the expression of “let”, as is shown in the following example.

Example 3.1 (“Let” expressions in Aπ [AF01])
The following equivalence shows that a restricted active substitution functions like a “let”.

$$\begin{aligned}
(\nu x)(\{^T/_x\} \mid A) &\equiv (\nu x)(\{^T/_x\} \mid A\{^T/_x\}) && \text{by (subst)}\\
&\equiv (\nu x)(\{^T/_x\}) \mid A\{^T/_x\} && \text{by (new-par)}\\
&\equiv \mathbf{0} \mid A\{^T/_x\} && \text{by (alias)}\\
&\equiv A\{^T/_x\} \mid \mathbf{0} && \text{by (par-C)}\\
&\equiv A\{^T/_x\} && \text{by (par-}\mathbf{0}\text{).}
\end{aligned}$$
□

A last note, (rewrite) deals with equational rewriting, which can be used to remove redundant, duplicate active substitutions.

Proposition 3.2 (Normal Form of Extended Processes)
Let A be a closed extended process. Then there exists an extended process nf(A) *on the form*

$$\text{nf}(A) \overset{\text{def}}{=} (\nu\tilde{a})(\sigma \mid P),$$

where $\sigma = \{^{\tilde{T}}/_{\tilde{x}}\}$, $\mathsf{fv}(P) = \emptyset$, $\mathsf{fv}(\tilde{T}) = \emptyset$, and $\tilde{a} \subseteq \mathsf{fn}(\tilde{T})$. □

[5]Through an active substitution.

Informally, $\mathsf{fv}(\tilde{T}) = \emptyset$ is achieved by substituting the occurrence of variables in the terms with the terms bound to those variables. $\mathsf{fv}(P) = \emptyset$ is obtained from the "let" equivalence in Example 3.1. That is, by replacing the occurrence of bound variables with the term bound to those variables, whereafter the variable and the associated restriction and active substitution disappear. Lastly, $\tilde{a} \subseteq \mathsf{fn}(\tilde{T})$ makes sense since any other name occurring in A can be factored into P.

We sometimes also refer to $\mathrm{nf}(A)$ as the *inner normal form* of A, as it involves substituting all values of all active substitutions "into" the process.

Internal Reduction: Internal reduction expresses the actual computation and behaviour of a process, and is defined as follows.

Definition 3.2 (Internal Reduction [AF01])
Internal reduction, $\rightarrow$, is the smallest relation on $\mathscr{A}$ closed by $\equiv$ and application of evaluation contexts, satisfying the relations in Table 3.4. □

(comm)	$\overline{a}\langle x\rangle.P \mid a(x).Q \rightarrow P \mid Q$
(then)	$\textbf{if } T = U \textbf{ then } P \textbf{ else } Q \rightarrow P$
(else)	$\textbf{if } T = U \textbf{ then } P \textbf{ else } Q \rightarrow Q, \text{ if} \Sigma \nvdash T = U$

Table 3.4: Internal Reduction in Aπ.

Notice how quaint the (comm) reduction rule is; it may at first glance seem as if the semantics does not define message-passing at all. However, by clever application of (alias) and (subst), the following reduction can be performed

$$\begin{aligned}\overline{a}\langle T\rangle.P \mid a(x).Q &\equiv (\nu x)\{{}^{T}/_{x}\} \mid \overline{a}\langle x\rangle.P \mid a(x).Q\\ &\rightarrow (\nu x)\{{}^{T}/_{x}\} \mid P \mid Q\\ &\equiv P \mid Q\{{}^{M}/_{x}\},\end{aligned}$$

provided $x \notin \mathsf{fv}(M) \cup \mathsf{fv}(P)$, which is a problem easily avoided by use of α-conversion.

We are now ready to define the operational semantics of Aπ.

Definition 3.3 (Operational Semantics of Aπ)
The *operational semantics* of Aπ is defined by the transition system $(\mathscr{A}, \rightarrow)$, where $\rightarrow \subseteq \mathscr{A} \times \mathscr{A}$. □

Labelled Operational Semantics: The labelled operational semantics extends the operational semantics detailed in Definition 3.3, with a labelled reduction relation such that synchronisations between a process and its environment can be observed. It does so with a new relation $A \xrightarrow{\alpha} A'$, where α ranges over the set of *labels* $\mathbf{A}$, generated by the grammar

$$\alpha ::= a(T) \mid \overline{a}\langle u\rangle \mid (\nu u)\overline{a}\langle u\rangle \mid \tau.$$

Here, $a(T)$ is an input action, denoting message receival from the environment, $\overline{a}\langle u\rangle$ and $(\nu u)\overline{a}\langle u\rangle$ are actions denoting output to the environment, where the latter expresses that the previously unexposed variable u now becomes exposed. τ denotes internal reduction.

Definition 3.4 (Labelled Reduction)
Labelled reduction, $\xrightarrow{\alpha}$, extends the rules of $\rightarrow$ with the rules in Table 3.5 $\square$

(in)	$a(x).P \xrightarrow{a(M)} P\{^{M}/_{x}\}$
(out-atom)	$\overline{a}\langle u\rangle \xrightarrow{\overline{a}\langle u\rangle} P$
(open-atom)	$\dfrac{A \xrightarrow{\overline{a}\langle u\rangle} A', \quad u \neq a}{(\nu u)A \xrightarrow{(\nu u)\overline{a}\langle u\rangle} A'}$
(scope)	$\dfrac{A \xrightarrow{\alpha} A', \quad u \text{ does not occur in } \alpha}{(\nu u)A \xrightarrow{(\nu u)\overline{a}\langle u\rangle} A'}$
(par)	$\dfrac{A \xrightarrow{\alpha} A', \quad \mathsf{bv}(\alpha) \cap \mathsf{fv}(B) = \mathsf{bn}(\alpha) \cap \mathsf{fn}(B) = \emptyset}{A \mid B \xrightarrow{\alpha} A' \mid B}$
(struct)	$\dfrac{A \equiv B, \quad B \xrightarrow{\alpha} B', \quad B' \equiv A'}{A \xrightarrow{\alpha} A'}$

Table 3.5: Structural equivalence in Aπ.

Definition 3.5 (Labelled Operational Semantics of Aπ)
The *labelled operational semantics* of Aπ is a triple $(\mathscr{A}, \mathbf{A}, \xrightarrow{\alpha} \cup \rightarrow)$, where $\xrightarrow{\alpha} \cup \rightarrow \subseteq \mathscr{A} \times \mathscr{A}$. $\square$

3.1.2 Equivalences

Here we introduce three equivalence relations; two behavioural equivalence relations, namely Observational Equivalence and Labelled Bisimilarity, and one knowledge indistinguishability relation, called Static Equivalence. These definitions are central in the secrecy definitions in Aπ.

Observational Equivalence

Here we define the behavioural equivalence of interest in our setting, namely observational equivalence. It is more interesting than strong bisimilarity in our setting because what we are interested in is whether two processes have observably different behaviour.

Before we define observational equivalence, we define the observables, or *barbs*, of a process. Let τ, as tradition, represent an internal reduction in a process, and let $\Longrightarrow$

represent the reflexive and transitive closure of $\xrightarrow{\tau}$. Furthermore, denote $\Longrightarrow\xrightarrow{\mu}\Longrightarrow$ by $\overset{\mu}{\Longrightarrow}$.

Internal actions (or τ, to the environment) are considered unobservable by the environment. The next definition defines the observables of a process.

Definition 3.6 (Barbs of π-calculus processes [CM02])
We define the observatility predicates, called barbs, of π-calculus processes, as follows.

$$\begin{array}{ll} P\downarrow_a \overset{\text{def}}{=} \exists y, Q[P \xrightarrow{a(y)} Q] & P \Downarrow_a \overset{\text{def}}{=} \exists R[P \Longrightarrow R \wedge R\downarrow_a] \\ P\downarrow_{\overline{a}} \overset{\text{def}}{=} \exists y, Q[P \xrightarrow{\overline{a}\langle y\rangle} Q] & P \Downarrow_{\overline{a}} \overset{\text{def}}{=} \exists R[P \Longrightarrow R \wedge R\downarrow_{\overline{a}}] \\ P\downarrow \overset{\text{def}}{=} \exists a[P\downarrow_a \vee P\downarrow_{\overline{a}}] & P\Downarrow \overset{\text{def}}{=} \exists a[P\downarrow_a \vee P\downarrow_{\overline{a}}] \end{array}$$

□

We are now ready to define observational equivalence.

Definition 3.7 (Observational Equivalence [AF01])
Observational equivalence, denoted $\approx$, is the largest binary symmetric relation $\mathcal{R}$ over closed extended processes with the same domain s.t. $A \approx B$ implies that

1) $A \Downarrow_a \Longrightarrow B \Downarrow_a$,

2) $A \rightarrow^* A' \implies \exists B'[B \rightarrow^* B' \wedge A'\mathcal{R}B']$,

3) $\forall C[\cdot][C[\cdot]$ closing evaluation context $\implies C[A]\mathcal{R}C[B]]$

holds. □

An example of how observational equivalence could be used in practice is to ensure that when replacing an old implementation with a new, improved inplementation (performance wise), that this can be done without notice (and thus without consequence) to the environment. This would be done by comparing the models of both implementations A and A', to see if $A \approx A'$ holds.

Static Equivalence

In the Aπ calculus, a *frame* is an extended process built only from **0**, active substitutions, parallel composition and name restriction. This is expressed in the following grammar.

$$\begin{array}{rl} \varphi ::= & (\nu a)\varphi \\ | & \{^T/_x\} \mid \varphi \\ | & \mathbf{0} \end{array}$$

By substituting each active substitution in an extended process A into each primitive process in contact with the substitution, eliminating redundant restrictions and active substitutions by use of (new-) rules, (alias), (subst), and (rewrite), and deleting all primitive processes, A is effectively *mapped to its frame.* We denote this mapping by

$\varphi(A)$. For an extended process on normal form, this mapping becomes straightforward; say $\mathrm{nf}(A) = (\nu\tilde{a})(\sigma \mid P)$. Then

$$\varphi(\mathrm{nf}(A)) = (\nu\tilde{a})\sigma,$$

which is structurally equivalent with $\varphi(A)$. In fact, we will from now on write $\varphi(A)$ as a shorthand for $\varphi(\mathrm{nf}(A))$. The active substitutions of a frame, like active substitutions themselves, can be considered a function which maps variable names to the term associated with it: The *domain* of a frame, written $\mathrm{dom}(\varphi)$, is recursively defined as

$$\begin{aligned}\mathrm{dom}((\nu a)\varphi) &\overset{\mathrm{def}}{=} \mathrm{dom}(\varphi)\\ \mathrm{dom}(\{^{T}/_{x}\} \mid \varphi) &\overset{\mathrm{def}}{=} \{x\} \cup \mathrm{dom}(\varphi)\\ \mathrm{dom}(\mathbf{0}) &\overset{\mathrm{def}}{=} \emptyset\end{aligned}$$

and denotes the set of variables φ exports, while the *image* of a frame, written $\mathrm{im}(\varphi)$, is recursively defined as

$$\begin{aligned}\mathrm{im}((\nu a)\varphi) &\overset{\mathrm{def}}{=} \mathrm{im}(\varphi)\\ \mathrm{im}(\{^{T}/_{x}\} \mid \varphi) &\overset{\mathrm{def}}{=} \{T\} \cup \mathrm{im}(\varphi)\\ \mathrm{im}(\mathbf{0}) &\overset{\mathrm{def}}{=} \emptyset\end{aligned}$$

and denotes the set of terms written in the variables which φ exports.

A frame $\varphi(A)$ of an extended process A can be viewed as an account for the static knowledge exposed by A to its environment, and not the dynamic behaviour of A, or the knowledge bound to A.

Definition 3.8 (Term Equality in a Frame)
The terms T and U are equal in frame φ, written $(T =_E U)\varphi$, if and only if $\varphi \equiv (\nu\tilde{a})\sigma$, $T\sigma =_E U\sigma$, and $\tilde{a} \cap (\mathsf{fn}(T) \cup \mathsf{fn}(U)) = \emptyset$. □

Definition 3.9 (Static Equivalence)
Two closed frames φ and ψ are statically equivalent, written $\varphi \approx_s \psi$, if $\mathrm{dom}(\varphi) = \mathrm{dom}(\psi)$, and $\forall T, U[(T =_E U)\varphi \iff (T =_E U)\psi]$ holds. Two closed extended processes A and B are statically equivalent, written $A \approx_s B$, if $\varphi(A) \approx_s \varphi(B)$. □

Informally, two frames φ and ψ are statically equivalent if they expose the same information to their environment.

We end this section with the statement of a couple of very useful lemmas from [AF01].

Lemma 3.3 (Closure Properties of $\approx_s$)
Static equivalence is closed by structural equivalence, bu reduction, and by application of closing evaluation contexts. □

Lemma 3.4 (Static- and Observational Equivalence Relationship)
Observational equivalence and static equivalence coincide on frames. Observational equivalence is strictly finer than static equivalence on extended processes. □

Labelled Bisimilarity

At last, we define labelled bisimilarity in Aπ, and mention the key result there is in its regard.

Definition 3.10 (Labelled Bisimilarity)
Labelled bisimilarity, denoted $\approx_l$, is the largest symmetric relation $\mathcal{R}$ on closed extended processes s.t. $A\mathcal{R}B$ implies

1. $A \approx_s B$.
2. if $A \rightarrow A'$, then $B \rightarrow^* B'$ and $A'\mathcal{R}B'$ for some B'.
3. if $A \xrightarrow{\alpha} A'$ and $\mathsf{fv}(\alpha) \subseteq \mathrm{dom}(A)$ and $\mathsf{bn}(\alpha) \cap \mathsf{fn}(B) = \emptyset$, then $B \overset{\alpha}{\Longrightarrow} B'$ and $A'\mathcal{R}B'$ for some B'. □

Theorem 3.5
Observational equivalence is labelled bisimilarity. □

The advantage of using static equivalence to check for behavioural equivalence of processes is because the universal quantification present in the definition of observational equivalence makes it hard at best to prove that two processes are related with that relation.

3.1.3 Secrecy

We now present the key definitions of secrecy in Aπ, as presented in [CRZ06]. There, Cortier et al consider secrecy in two settings.

Passive Case

This setting involves analysing the information leaked from a system at a given point in time, typically immediately after a protocol execution.

Definition 3.11 (Deduction from a Frame)
For a given equational theory E, we say T may be deduced from φ, written $\varphi \vdash T$, when that fact is derivable using the axioms

(ax$_1$) $$\frac{}{(\nu\tilde{n})\sigma \vdash x\sigma} \quad \text{where} \quad x \in \mathsf{dom}(\sigma)$$

(ax$_2$) $$\frac{}{(\nu\tilde{n})\sigma \vdash s} \quad \text{where} \quad s \notin \tilde{n}$$

(ax$_3$) $$\frac{\varphi \vdash T_1 \cdots \varphi \vdash T_l}{\varphi \vdash \mathsf{f}(T_1, \ldots, T_l)} \quad \text{where} \quad \mathsf{f} \in \Sigma$$

(ax$_4$) $$\frac{\varphi \vdash T \qquad T =_E T'}{\varphi \vdash T'}$$ □

It is worth mentioning now that Abadi and Cortier have proven in [AC04, AC05] that deduction from a frame and static equivalence are not only decidable for convergent subterm theories, but up to associativity and commutativity of the equational theory.

Definition 3.12 (Syntactic Secrecy)
T is *syntactically secret in* φ if $\varphi \not\vdash T$. □

Definition 3.13 (Strong Secrecy)
For s free in $\varphi = (\nu\tilde{n})\sigma$, s is *strongly secret* in φ if

$$\forall T, T' \text{ closed w.r.t. } \varphi \left[\varphi\,(^{T}/_{s}) \approx_s \varphi\,(^{T'}/_{s})\right]$$ □

Here it is helpful to consider s as a "placeholder" for any given term. Abadi and Cortier take this approach because they are not interested in that the term which "replaces" s appear as an active substitution, but rather, that it is "written into" the process specification directly, in place of s.

Examples of where strong secrecy is a very sound definition of secrecy is if the process being verified A represents a vote counter between two candidates. While the observer does not see the value of the actual votes, he may observe different behaviour in A for different for the receival of the respective votes.

Active Case

This setting involves analysing security protocols for any number of executions. While these are indeed very robust definitions, they are significantly more difficult to guarantee.

Let P be a closed process without variables as channels, and $s \in \mathsf{fn}(P)$, but not a channel name. We say a frame φ is valid in A if there exists a process A' to which A reduces through 0 or more reduction steps, where φ is the frame of A'.

Definition 3.14 (Syntactic Secrecy)
s is *syntactically secret* in P if

$$\forall \varphi \text{ valid w.r.t. } P \left[\nu s.\varphi \not\vdash s\right]$$ □

Definition 3.15 (Strong Secrecy)
s is *strongly secret* in P if

$$\forall T, T' \text{ closed}$$
$$\left[\mathsf{bn}(P) \cap (\mathsf{fn}(T) \cup \mathsf{fn}(T')) = \emptyset \implies P\,(^{T}/_{s}) \approx_l P\,(^{T'}/_{s})\right]$$ □

The way the condition $\mathsf{bn}(P) \cap (\mathsf{fn}(T) \cup \mathsf{fn}(T')) = \emptyset$ should be understood is that T, T' are terms which the environment is capable of constructing.

The "valid" frames of a process can be considered as the *reachable* frames of said process. We shall use the terms "valid" and "reachable" frame interchangeably.

As a final remark, Cortier, Rusinowitch and Zălinescu have proven in [CRZ06] that in some select (and quite constrained) cases, *syntactic secrecy implies strong secrecy.* They provide an automated technique for verifying whether a protocol specification satisfies the constraints, though.

3.2 Abadi/Blanchet Calculus

Next up is the calculus of Abadi and Blanchet, which we refer to here as ABπ. We present the syntax, semantics, and the Horn clause generation technique detailed in [AB02], the latter enriched with sessions as in [Bla02]. Afterwards we summarise the authenticicty extensions Blanchet makes to ABπ and the Horn clause generator, and which effect this has [Bla02].

3.2.1 Syntax and Semantics

Syntax

We let $\mathcal{N}$ be the set of names ranged over by $a, \dots, c, m, \dots, t$, and $\mathcal{V}$ be the set of variables ranged over by $x, \dots, z$. Furthermore, let $\mathcal{U} = \mathcal{N} \cup \mathcal{V}$ be the set of identifiers, ranged over by $u, \dots, w$. Let $\mathcal{F}$ be the set of constructors, ranged over by f, and $\mathcal{G}$ be the set of destructors, ranged over by g. The syntax of ABπ is then given by the grammar specification in Table 3.6, where P is a process, which are ranged over by $P, \dots, R \in \mathscr{P}$,

$$\begin{aligned} P ::=\ & T(x).P \\ \mid\ & \overline{T}\langle T\rangle.P \\ \mid\ & !P \\ \mid\ & (\nu a)P \\ \mid\ & P \mid P \\ \mid\ & \textbf{let } x = \mathsf{g}(T_1, \dots, T_n) \textbf{ in } P \textbf{ else } P \\ \mid\ & \mathbf{0} \end{aligned} \qquad \begin{aligned} T ::=\ & u \\ \mid\ & \mathsf{f}(T, \dots, T) \end{aligned}$$

Table 3.6: Syntax of ABπ.

and T is a term, which are ranged over by $T, \dots, V \in \mathcal{T}$.

The semantics of destructors in ABπ is given in terms of a particular *term rewrite system*: For each destructor g of arity n, there is a set $\mathrm{def}(\mathsf{g})$ of reduction rules on the form $\mathsf{g}(T_1, \dots, T_n) > T$, where $T_1, \dots, T_n, T$ are closed terms. For each such a rule, if there exists a substitution σ such that $T_i\sigma = T_i'$ and $T\sigma = T'$ for some closed terms $T_1', \dots, T_n', T'$, then the reduction rule $\mathsf{g}(T_1', \dots, T_n') > T'$ is also contained in $\mathrm{def}(\mathsf{g})$.

The term rewrite system in ABπ has fairly common Sπ-like cryptographic primitives. As opposed to Aπ, the data language is not directly replaceable. We are fairly confident, however, that replacing the term rewrite system with another one would result in a functioning calculus.

The only datatype in ABπ, like in the π-calculus, is the channel name. Notice that terms cannot be built from destructor applications. Rather, destructors are operations which processes can explicitly apply on terms. This is expressed by the process **let** $x = \mathsf{g}(T_1, \dots, T_n)$ **in** P **else** Q: Here, if $\mathsf{g}(T_1, \dots, T_n)$ is the left-side of any reduction rule in def(g), then x is bound to the right-side of one of those rules (chosen nondeterministically), whereafter $P\{^T/_x\}$ is executed. If not, then Q is executed.

All other syntactic constructs should be familiar from the π-calculus. Moreover, we let **let** $x = T$ **in** P be syntactic sugar for $P\{^T/_x\}$, and **if** $T = U$ **then** P **else** Q be a shorthand for **let** $x = \mathsf{equal}(T, U)$ **in** P **else** Q, where x does not occur in P and Q, with $\mathsf{equal}(T, T) > T \in \mathrm{def}(\mathsf{equal})$.

The semantics of ABπ is defined in terms of structural equivalence and internal reduction, as usual.

Definition 3.16 (Structural Equivalence in ABπ)
Structural equivalence, denoted by $\equiv$, is the smallest equivalence relation over $\mathscr{P}$ that is closed by α-conversions, by application of evaluation contexts, and which satisfies the axioms in Table 3.7 □

(nil)	$P \equiv P \mid \mathbf{0}$
(par$_{com}$)	$P \mid Q \equiv Q \mid P$
(par$_{ass}$)	$P \mid Q \mid R \equiv P \mid Q \mid R$
(new$_{com}$)	$(\nu a_1)(\nu a_2)P \equiv (\nu a_2)(\nu a_1)P$
(new$_{exp}$)	$(\nu a)(P \mid Q) \equiv P \mid (\nu a)Q$, if$a \notin \mathsf{fn}(P)$

Table 3.7: Structural equivalence in ABπ.

Since, by rules (nil), (par$_{com}$) and (par$_{ass}$), $\equiv$ is associative, commutative, and has an identity element, $(\mathscr{P}, |)$ is an abelian monoid.

Definition 3.17
Internal reduction, denoted by $\rightarrow$, is the least precongruence on processes closed by structural equivalence and evaluation contexts, and satisfying the axioms in Table 3.8.□

(destr$_1$)	**let** $x = \mathsf{g}(T_1, \ldots, T_n)$ **in** P **else** $Q \rightarrow P\{^T/_x\}$, if $\mathsf{g}(T_1, \ldots, T_n) > T \in \mathrm{def}(\mathsf{g})$
(destr$_2$)	**let** $x = \mathsf{g}(T_1, \ldots, T_n)$ **in** P **else** $Q \rightarrow P\{^T/_x\}$, if $\nexists T[\mathsf{g}(T_1, \ldots, T_n) > T \in \mathrm{def}(\mathsf{g})]$
(sync)	$\overline{a}\langle T\rangle.Q \mid a(x).P \rightarrow Q \mid P\{^T/_x\}$
(repl)	$!P \rightarrow P \mid !P$

Table 3.8: Internal reduction relation in ABπ.

We can now define the *operational semantics* of ABπ as a transition system.

Definition 3.18 (Operational Semantics of ABπ)
The *operational semantics* of ABπ is given as the transition system $(\mathscr{P}, \rightarrow)$, where $\rightarrow \subseteq \mathscr{P} \times \mathscr{P}$. □

3.2.2 Secrecy

When performing secrecy analysis on processes in ABπ, the Dolev-Yao threat model is applied. In that model, the environment is considered an adversary, which implicitly receives any sent message, and can send any message it knows, that is, any term built from free names and what could be deduced from prior received messages. In ABπ, the environment is expressed as an arbitrary adversary.

Definition 3.19 (Adversary)
Let S be a finite set of names. The closed process Q is an S-adversary if and only if $\mathsf{fn}(Q) \subseteq S$. □

Definition 3.20 (Secrecy)
We say that the closed process P *outputs* T on c if and only if $P \longrightarrow^* C[\overline{c}\langle T\rangle.R]$ for some process R and some context C which does not prefix its hole with actions. P preserves the secrecy of T from S if and only if $P \mid Q$ does not output T on c, for any S-adversary Q and any $c \in S$. □

These definitions contrast to the secrecy definitions in Aπ in that in Aπ, *any* name that becomes free during reduction is considered "compromised", whereas here, we always consider a static set S.

3.2.3 Static Analysis

We now present the approach to static analysis of secrecy constraints in ABπ. An initial process, P_0, is a process on which no reductions have been performed. Names in P_0 are treated as function symbols in the Horn clauses generated by the Horn clause generation algorithm. This is done to ensure that names that differ semantically in P_0 differ in the Horn clauses. Terms thus instead become *patterns*, which are given by the grammar specification in Table 3.9. The notation $a[p_1, \ldots, p_n, i_1, \ldots, i_{n'}]$, as opposed

$$\begin{aligned} p ::=\ & x, i \\ \mid\ & a[p_1, \ldots, p_n, i_1, \ldots, i_{n'}] \\ \mid\ & \mathsf{f}(p_1, \ldots, p_n) \end{aligned}$$

Table 3.9: Patterns in Horn clauses

to $\mathsf{a}(p_1, \ldots, p_n, i_1, \ldots, i_{n'})$, is merely used to distinguish names from constructors. If a is a free name in P_0, then its corresponding function arity in the Horn clauses is 0, and it encodes simply to $a[]$. If a is bound, however, then the arity is the total number of input statements, destructor applications and replications above restriction on a. In $a[p_1, \dots, p_n, i_1, \dots, i_{n'}]$, $p_1, \dots, p_n$ correspond to input statements and destructor applications, while $i_1, \dots, i_{n'}$ are *session identifiers* — integers or variables taken from the set $\mathcal{V}_s$ disjoint from the set $\mathcal{V}_o$ or ordinary variables. There is one session identifier for each replication above the restriction on a in P_0.

The Horn clause generation algorithm uses the *predicates* given in Table 3.10. The

$$
\begin{aligned}
F ::= {} & \mathsf{att}(p) \\
\mid {} & \mathsf{msg}(p, p')
\end{aligned}
$$

Table 3.10: Predicates in Horn clauses

$\mathsf{msg}(p, p')$ predicate expresses that a message p' has been sent on channel p, while $\mathsf{att}(p)$ expresses that the attacker knows p.

The Horn clauses for a security analysis on P_0 are split into two categories: The learning capabilities of the attacker, and the control flow in P_0. The former, which we denote $\mathsf{H}_{\mathsf{att}}(P_0, S)$, a function of P_0 and the designated unsafe names of P_0, is defined by Algorithm 3.1. Note the use of variables in the Horn clauses; this essentially means that the Horn clause is valid for any value assignment of these variables.

Algorithm 3.1: Horn clauses for the attacker, $\mathsf{H}_{\mathsf{att}}(P_0, S)$.

```
1  Input: Set of names $S$, and a process $P_0$.
2  Result: Set of Horn clauses $H_{att}$, expressing learning capability of attacker.
3  $H_{att} := \emptyset$
4  foreach $a \in S$ do
5      $H_{att} := H_{att} \cup \{\mathsf{att}(a[])\}$
6  $H_{att} := H_{att} \cup \{\mathsf{att}(b[i])\}$ for some nondeterministically chosen $b \notin \mathsf{n}(P_0)$
7  foreach $\mathsf{f} \in \Sigma$ do
8      $H_{att} := H_{att} \cup \{\bigwedge_{1 \le i \le \alpha(\mathsf{f})} \mathsf{att}(x_i) \implies \mathsf{att}(\mathsf{f}(x_1, \dots, x_{\alpha(\mathsf{f})}))\}$
9  foreach $\mathsf{g} \in \Sigma$ do
10     foreach $\mathsf{g}(T_1, \dots, T_n) > T \in \mathrm{def}(\mathsf{g})$ do
11         $H_{att} := H_{att} \cup \{\bigwedge_{1 \le i \le n} \mathsf{att}(T_i) \implies \mathsf{att}(T)\}$
12 $H_{att} := H_{att} \cup \{\mathsf{msg}(x, y) \land \mathsf{att}(x) \implies \mathsf{att}(y)\}$
13 $H_{att} := H_{att} \cup \{\mathsf{att}(x) \land \mathsf{att}(y) \implies \mathsf{msg}(x, y)\}$
14 return $H_{att}$
```

The latter, which we denote $\mathsf{H}_{\mathsf{prot}}(P_0)$, a function of P_0, is defined in terms of the encoding in Table 3.11. There, ρ is a map which maps names and variables in P_0 during encoding, to corresponding patterns. Notice that ρ is a homomorphism: $\rho(\mathsf{f}(T_1, \dots, T_n)) = \mathsf{f}(\rho(T_1), \dots, \rho(T_n))$. Also, $\rho(\mathcal{V}_o)$ is a shorthand for $\rho(x_1), \dots, \rho(x_m)$, where x_i are pairwise different and $\{x_i\}_{1\leq i\leq m}\,\mathrm{dom}(\rho)\cap\mathcal{V}_o$. Now, h, the "history", is a conjunction of input predicates encountered during the traversal of a branch in the encoding.

$$
\begin{aligned}
[\![\mathbf{0}]\!]_{\rho h} &= \emptyset \\
[\![P \mid Q]\!]_{\rho h} &= [\![P]\!]_{\rho h} \cup [\![Q]\!]_{\rho h} \\
[\![!P]\!]_{\rho h} &= [\![P]\!]_{\rho[i\mapsto i]h}, \text{ where } i \text{ is a new session identifier} \\
[\![(\nu a)P]\!]_{\rho h} &= [\![P]\!]_{(\rho[a\mapsto a[\rho(V_o),\rho(V_s)]])h} \\
[\![T(x).P]\!]_{\rho h} &= [\![P]\!]_{(\rho[x\mapsto x])(h\wedge \mathsf{msg}(\rho(T),x))} \\
[\![\overline{T}\langle U\rangle.P]\!]_{\rho h} &= [\![P]\!]_{\rho h} \cup \{h \implies \mathsf{msg}(\rho(T),\rho(U))\} \\
\left[\!\!\left[\begin{array}{l}\textbf{let } x = \mathsf{g}(T_1,\ldots,T_n)\\ \textbf{in } P\\ \textbf{else } Q\end{array}\right]\!\!\right]_{\rho h} &= \bigcup\left\{ [\![P]\!]_{((\sigma\rho)[x\mapsto\sigma' p'])(\sigma h)} \;\middle|\; \begin{array}{l}\mathsf{g}(p'_1,\ldots,p'_n) > p' \in \mathrm{def}(\mathsf{g}),\\ \text{and } (\sigma,\sigma') \text{ is the most gene-}\\ \text{ral pair of substitutions}\\ \text{st. } \sigma\rho(T_i) = \sigma' p'_i; 1\leq i\leq n\end{array}\right\} \\
&\cup\ [\![Q]\!]_{\rho h}
\end{aligned}
$$

Table 3.11: Horn clauses for control flow.

Let $\rho_{\mathsf{init}} = \{a \mapsto a[] \mid a \in \mathsf{fn}(P_0)\}$. Then $\mathsf{H}_{\mathsf{prot}}(P_0) = [\![P_0]\!]_{\rho_{\mathsf{init}}\emptyset}$. For each replication above a certain point in the parse tree of P_0, the set $\rho(\mathcal{V}_s)$ will contain one unique session variable. The idea is then that replicating a process corresponds to assigning a different value to the corresponding session variable. This can be expressed by changing (repl) to $!^{i\geq n}P \to P\{^n/_i\} \mid !^{i\geq n+1}P$, where $!^{i\geq n}P$ represents copies of P for each $i \geq n$.

The full set of Horn clauses for a security analysis on P_0 with S as the designated unsafe names is defined as $\mathsf{H}(P_0,S) = \mathsf{H}_{\mathsf{att}}(P_0,S) \cup \mathsf{H}_{\mathsf{prot}}(P_0)$. When the actual P_0 and S are obvious from the context, we let H, $\mathsf{H}_{\mathsf{att}}$, and $\mathsf{H}_{\mathsf{prot}}$, abbreviate $\mathsf{H}(P_0,S)$, $\mathsf{H}_{\mathsf{att}}(P_0,S)$, and $\mathsf{H}_{\mathsf{prot}}(P_0)$, respectively.

Finally, we define the following.

Definition 3.21 (Deduction from a set of Horn clauses)
We say that predicate p is deducible from a set of Horn clauses H, written $H \vdash p$, if p can be derived from H. □

Recall that derivation of a predicate from a set of Horn clauses is defined in Definition 2.23.

3.2.4 Authenticity Extension

In [Bla02], Blanchet extends on the work they made in [AB02] by adding primitives to ABπ for expressing beginnig and ending of authenticated sessions, for the purpose of verifying authentication properties in security protocols.

The syntax of ABπ is extended with the following authentication primitives.

$$\begin{aligned} P ::= {} & \mathbf{begin}(T).P \\ \mid {} & \mathbf{end}(T).P \\ \mid {} & \mathbf{begin\_ex}(T) \\ \mid {} & \mathbf{end\_ex}(T) \end{aligned}$$

Here, $\mathbf{begin}(T).P$ and $\mathbf{end}(T).P$ denote the act of initiating and ending an authenticated session, respectively, and then behaving like P. After these actions are performed, a floating begin $\mathbf{begin\_ex}(T)$, and a floating end indicator, $\mathbf{end\_ex}(T)$, respectively, will appear as a parallel component. This is expressed in the following addition the primitive reduction relation.

$$\mathbf{begin}(T).P \to \mathbf{begin\_ex}(T) \mid P$$
$$\mathbf{end}(T).P \to \mathbf{end\_ex}(T) \mid P$$

Blanchet then proceeds to define noninjective agreement.

Definition 3.22 (Non-injective Agreement)
The closed starting process P_0 *satisfies non-injective agreemenet* with respect to S-adversaries iff for any S-adversary Q, for any P_0' such that $P_0 \mid Q(\equiv \cup \to)^* P_0'$, for any T, if $\mathbf{end\_ex}(T)$ occurs in P_0', then $\mathbf{begin\_ex}(T)$ also occurs in P_0' (Assumption: different names in P_0' all differ syntactically). □

Definition 3.23 (Injective Agreement)
The closed starting process P_0 *satisfies injective agreemenet* with respect to S-adversaries iff for any S-adversary Q, for any P_0' such that $P_0 \mid Q(\equiv \cup \to)^* P_0'$, for any T, number of occurrences of $\mathbf{end\_ex}(T)$ in $P_0'$is less than or equal the number of $\mathbf{begin\_ex}(T)$ in P_0' (Assumption: different names in P_0' all differ syntactically). □

To verify these properties, Blanchet then extends the predicates,

$$\begin{aligned} F ::= {} & \mathsf{begin}(p, \rho) \\ \mid {} & \mathsf{end}(p, s) \end{aligned}$$

$$\begin{aligned} [\![\mathbf{begin}(T).P]\!]_{\rho h} &= [\![P]\!]_{\rho h \wedge \mathsf{begin}(\rho(T), \rho)} \\ [\![\mathbf{end}(T).P]\!]_{\rho h} &= [\![P]\!]_{\rho h} \cup \{h \implies \mathsf{end}(\rho(T), \rho(\mathcal{V}_S))\}, \end{aligned}$$

Theorem 3.6
Assume that, for any set $\mathsf{B_b}$ *of permitted* begin *actions, if* $\mathsf{end}(T, s)$ *is derivable from* $\mathsf{H}(P_0, S) \cup \mathsf{B_b}$*, then* $\mathsf{begin}(T, \rho) \in \mathsf{B_b}$*. Then* P_0 *satisfies non-injective agreement with respect to* S*-adversaries.* □

Blanchet has a similar theorem for injective agreement in [Bla02].

On a final note, in [BP05], Blanchet and Podelski guarantee termination of the resolution algorithm for H in an attacker-free setting, through a technique they refer to as "tagging".

3.3 CBS$^\sharp$

CBS$^\sharp$ [NH06, Nan06] is a broadcast calculus developed by Sebastian Nanz as part of his PhD (supervised by Chris Hankin), developed for the purpose of modelling and verifying protocols for mobile wireless networks. CBS$^\sharp$ is a further development of their previous work [NH04] on extending CBS with locations and locality[6]. In CBS$^\sharp$, a protocol is expressed as a network of locations, each containing a sequential process, or a parallel composition thereof, similar to Dπ. Unlike Dπ, processes cannot move from location to location, and CBS$^\sharp$ employs an asynchronous send broadcast semantics. Broadcasting a message is a lossy, atomic transmission step. Each location has its own (infinite) memory for (a monotonely increasing) storage of terms, which a process within the location can read from and add terms to, at runtime. Connectivity of locations is expressed as a set of undirected graphs, each graph representing the connectivity of locations at a given point in time (when a process is broadcasting). The connectivity graphs dictate the scope of broadcasts.

3.3.1 Syntax and Semantics

Syntax

Let $\mathcal{N}$ be the set of names ranged over by $a, \ldots, c, o, \ldots, t$, with nodes ranged over by $m, \ldots, n \in \mathcal{N}_{\text{loc}} \subseteq \mathcal{N}$, and $\mathcal{V}$ be the set of variables ranged over by $x, \ldots, z$ (both sets being infinite). Let $\mathcal{N} \cup \mathcal{V} = \mathcal{U}$ be ranged over by $u, \ldots, w$. Let $\mathcal{F}$ be the set of function symbols, ranged over by f. The syntax of CBS$^\sharp$ is then given by the grammar specification in Table 3.12, where the syntactic category N represents *networks*, which are ranged over

$$
\begin{array}{rcl@{\qquad\qquad}rcl}
P & ::= & \mathbf{0} & T & ::= & u \\
 & | & (x).P & & | & \mathsf{f}(\tilde{T}) \\
 & | & \langle T \rangle.P & & & \\
 & | & !P & N & ::= & n[P, S] \\
 & | & P \mid P & & | & N \mid N \\
 & | & \mathbf{case}\ T\ \mathbf{of}\ \mathsf{f}(\tilde{T}; \tilde{x})\ P\ \mathbf{else}\ P & & & \\
 & | & \mathbf{read}\ x.P & & & \\
 & | & \mathbf{store}\ T.P & & &
\end{array}
$$

Table 3.12: Syntax of CBS$^\sharp$.

by $M, \ldots, N \in \mathscr{N}$, the syntactic category T represents *terms*, which are ranged over by $T, \ldots, V \in \mathcal{T}$, and the syntactic category P represents *processes*, which are ranged over by $P, \ldots, R \in \mathscr{P}$.

[6]However, while locality in their prior extension to CBS is probabilistic, then this is not the case for CBS$^\sharp$.

Connectivity Graphs

We let $\mathsf{V}(N)$ denote the locations in a the network N. Network connectivity (locality of locations) is expressed by connectivity graphs.

Definition 3.24 (Connectivity Graph)
A graph G is a *connectivity graph* if $\mathsf{V}(G)$ is a finite set, $\mathsf{V}(G) \subseteq \mathcal{N}_{\text{loc}}$ and $\forall n, m \in \mathsf{V}(G)[(n, m) \in \mathsf{E}(G) \implies n \neq m]$ holds. We say G is *admissible on a network* N if $\mathsf{V}(N) \subseteq \mathsf{V}(G)$. □

That is, a connectivity graph is a directed graph which vertices are a finite set of locations where no location has an edge to itself. For an admissible connectivity graph G, all the locations which appear in N appear amongst the locations in G. As such, the connectivity graph expresses the connectivity of locations in the model N, as well as other (potentially unsafe) locations which are not part of the specified network N.

Definition 3.25 (Network Topology)
A network topology τ is a set of connectivity graphs. We say τ is admissible to a network N when all the graphs in τ are. □

Typically, we use network topologies to specify invariants on the connectivity of locations in our networks, such as

$$\tau = \{G \mid (n, m) \in \mathsf{E}(G),\ (m, n) \in \mathsf{E}(G)\},$$

which is the set of all graphs wherein the vertices m and n are connected.

We define τ_{max} as the set containing all graphs on $\mathcal{N}_{\text{loc}}$. Any network topology is thus a subset of τ_{max}.

Semantics

We are now ready to give the *operational semantics* of CBS$^\sharp$. It is defined by three relations: A labelled transition relation, a reduction relation, and a structural equivalence. We start off with the labelled transition relation, which expresses the semantics of message-passing in CBS$^\sharp$.

Definition 3.26 (Labelled Transition Relation)
The *labelled transition relation*, denoted by $\xrightarrow{\alpha}$, is the least relation on $\mathcal{N}$ satisfying the axioms in Table 3.13 □

To define the interaction between input and output, the semantics of the labelled transition relation makes use of an algebra of modes.

Definition 3.27 (Mode Identifiers)
The set $\{!, ?, :\}$ is the set of *mode identifiers*, which elements are ranged over by the meta variable $\sharp$. Mode identifiers can be composed with the $\circ$ operator according to the algebra in Table 3.14, where $\perp$ denotes that ! cannot be combined with itself. □

(nil) $$\frac{}{n[\mathbf{0}, S] \xrightarrow{(T,m):}_G n[\mathbf{0}, S]}$$

(out$_1$) $$\frac{}{n[\langle T\rangle.P, S] \xrightarrow{(T,n)!}_G n[P, S]}$$

(out$_2$) $$\frac{}{n[\langle T\rangle.P, S] \xrightarrow{(U,m):}_G n[\langle T\rangle.P, S]}$$

(in$_1$) $$\frac{(m,n) \in \mathsf{E}(G)}{n[(x).P, S] \xrightarrow{(T,m)?}_G n[P\{T/x\}, S]}$$

(in$_2$) $$\frac{(m,n) \notin \mathsf{E}(G)}{n[(x).P, S] \xrightarrow{(T,m):}_G n[(x).P, S]}$$

(repl$_1$) $$\frac{}{n[!P, S] \xrightarrow{(T,m):}_G n[!P, S]}$$

(ppar) $$\frac{N_1 \xrightarrow{(T,m)\sharp_1}_G N_1' \qquad N_2 \xrightarrow{(T,m)\sharp_2}_G N_2'}{N_1 \mid N_2 \xrightarrow{(T,m)(\sharp_1 \circ \sharp_2)}_G N_1' \mid N_2'}$$

(struct) $$\frac{N \equiv M \qquad M \xrightarrow{(T,m)\sharp}_G M' \qquad M' \equiv N'}{N \xrightarrow{(T,m)\sharp}_G N'}$$

Table 3.13: Labeled Transition in CBS$^\sharp$.

We usually perform transitions in a network in accordance to a given network topology. To emphasise this fact, we define the following.

Definition 3.28 (τ-faithfulness)
A reduction $\xrightarrow{(T,m)\sharp}_G$ is said to be τ-faithful if and only if $G \in \tau$. We denote this fact by $\xrightarrow{(T,m)\sharp}_{G\in\tau}$. □

We define $\rightarrow_\tau^*$ as the reflexive, transitive closure of τ-faithful output transitions. For instance, the transition sequence

$$N \xrightarrow{(T_1,m_1)!}_{G_1\in\tau} N_1 \xrightarrow{(T_2,m_2)!}_{G_2\in\tau} N_2 \cdots \xrightarrow{(T_k,m_k)!}_{G_k\in\tau} N'$$

can be written as $N \rightarrow_\tau^* N'$ (if $G_1, \ldots, G_k \in \tau$).

Next we have the reduction relation, which expresses computation in processes.

Definition 3.29 (Reduction Relation)
The *reduction relation*, denoted $\rightarrow$, is the least relation on $\mathscr{N}$ satisfying the axioms in Table 3.15. □

At last, we have the traditional structural equivalence relation.

$\circ$	!	?	:
!	$\perp$	!	!
?	!	?	?
:	!	?	:

Table 3.14: Mode Identifiers in CBS$^\sharp$.

(repl$_2$)	$\dfrac{}{n[!P,S] \to n[P \mid !P,S]}$
(store)	$\dfrac{}{n[\textbf{store } T.P,S] \to n[P,S \cup T]}$
(read)	$\dfrac{T \in S}{n[\textbf{read } x.P,S] \to n[P\{T/x\},S]}$
(case$_1$)	$\dfrac{T = \mathsf{f}(\tilde{T};\tilde{U}) \quad \lvert\tilde{U}\rvert = \lvert\tilde{x}\rvert}{n[\textbf{case } T \textbf{ of } \mathsf{f}(\tilde{T};\tilde{x})\ P \textbf{ else } Q,S] \to n[P\{\tilde{T}/\tilde{x}\};S]}$
(case$_2$)	$\dfrac{\nexists \tilde{U}.T = \mathsf{f}(\tilde{T};\tilde{U}) \wedge \lvert\tilde{U}\rvert = \lvert\tilde{x}\rvert}{n[\textbf{case } T \textbf{ of } \mathsf{f}(\tilde{T};\tilde{x})\ P \textbf{ else } Q,S] \to n[Q;S]}$

Table 3.15: Reduction in CBS$^\sharp$.

Definition 3.30 (Structural Equivalence)
The *structural equivalence* relation, denoted $\equiv$, is the least relation on $\mathscr{N}$ satisfying the axioms in Table 3.16. □

We have by (refl), (sym) and (trans) that $\equiv$ is an equivalence relation. However, $\equiv$ is *not* an abelian monoid with respect to parallel composition, even though $\equiv$ is associative and commutative with respect to parallel composition as per (assoc) and (comm) (there is no identity element in $\mathscr{N}$). $(\mathscr{N}, \mid)$ is a semigroup, though.

3.3.2 Control Flow Analysis

Overview

In the analysis of CBS$^\sharp$, we make use of an abstract network topology $\mathcal{G}(\tau)$ of a topology τ as

$$\mathcal{G}(\tau) \overset{\text{def}}{=} \left(\bigcup_{G \in \tau} \mathsf{V}(G), \bigcup_{G \in \tau} \mathsf{E}(G) \right),$$

which is a static abstraction which contains all $G \in \tau$ as subgraphs. $\mathcal{G}(\tau)$ is a useful overapproximation when performing control flow analysis, as it avoids state-space explosion problems associated with analysing each possible message flow with regards to any possible network connectivity.

(comm)	$\dfrac{}{N_1 \mid N_2 \equiv N_2 \mid N_1}$
(assoc)	$\dfrac{}{N_1 \mid (N_2 \mid N_3) \equiv (N_1 \mid N_2) \mid N_3}$
(refl)	$\dfrac{}{N \equiv N}$
(sym)	$\dfrac{N \equiv N'}{N' \equiv N}$
(trans)	$\dfrac{N \equiv N'' \quad N'' \equiv N}{N \equiv N'}$
(comp)	$\dfrac{N_1 \equiv N_1'}{N_1 \mid N_2 \equiv N_1' \mid N_2}$
(red)	$\dfrac{N \to N'}{N \equiv N'}$
(par)	$\dfrac{}{n[P_1 \mid P_2, S_1 \cup S_2] \equiv n[P_1, S_1] \mid n[P_2, S_2]}$

Table 3.16: Structural Equivalence in CBS$^\sharp$.

The control flow analysis is specified as a *flow logic*; an approach to static analysis where the specification of the acceptability of an analysis estimate is separated from the computation of the analysis. There are two judgements in the flow logic:

$(\kappa, \sigma) \models_{\mathcal{G}_\tau} N$	Judgement for networks.
$(\kappa, \sigma) \models^{\theta}_{\mathcal{G}_\tau, n} P$	Judgement for processes.

where

$\kappa \subseteq \mathcal{T} \times \mathcal{N}_{\mathrm{loc}}$	Network cache.
$\sigma \subseteq \mathcal{T} \times \mathcal{N}_{\mathrm{loc}}$	Store cache.

The overapproximation occurs in the computation of the sets κ and σ, representing which terms may be output in N from where, and which terms may be stored in N and where, respectively.

The judgements should be read "(κ, σ) is a valid analysis estimate describing the behaviour of N under $\mathcal{G}(\tau)$" and "(κ, σ) is a valid analysis estimate describing the behaviour of P at n under $\mathcal{G}(\tau)$", respectively. The latter makes use of a substitution θ, which keeps track of possible instantiations of variables in P.

Excerpt

An excerpt from the control flow analysis of CBS$^\sharp$ is contained in Table 3.3.2. The judgement for networks states (in brief) that (κ, σ) is a valid analysis result describing

Judgement for Networks:

$$
\begin{array}{ll}
\text{(node)} & (\kappa,\sigma) \models_{\mathcal{G}(\tau)} n[P\theta, S] \\
& \quad \text{iff } (\kappa,\sigma) \models^{\theta}_{\mathcal{G}(\tau),n} P \wedge \forall T \in S.(T,n) \in \sigma \\
\text{(ppar)} & (\kappa,\sigma) \models_{\mathcal{G}(\tau)} N_1 \mid N_2 \\
& \quad \text{iff } (\kappa,\sigma) \models_{\mathcal{G}(\tau)} N_1 \wedge (\kappa,\sigma) \models_{\mathcal{G}(\tau)} N_2
\end{array}
$$

Judgement for Processes:

$$
\begin{array}{ll}
\text{(out)} & (\kappa,\sigma) \models^{\theta}_{\mathcal{G}(\tau),n} \langle T \rangle.P \\
& \quad \text{iff } (T\theta, n) \in \kappa \wedge (\kappa,\sigma) \models^{\theta}_{\mathcal{G}(\tau)} P \\
\text{(in)} & (\kappa,\sigma) \models^{\theta}_{\mathcal{G}(\tau),n} (x).P \\
& \quad \text{iff } \forall (T,m) \in \kappa\,.\,(m,n) \in \mathsf{E}(\mathcal{G}(\tau)) \implies (\kappa,\sigma) \models^{\theta\{T/x\}}_{\mathcal{G}(\tau)} P \\
\text{(store)} & (\kappa,\sigma) \models^{\theta}_{\mathcal{G}(\tau),n} \textbf{store } T.P \\
& \quad \text{iff } (T\theta, n) \in \sigma \wedge (\kappa,\sigma) \models^{\theta}_{\mathcal{G}(\tau)} P \\
\text{(read)} & (\kappa,\sigma) \models^{\theta}_{\mathcal{G}(\tau),n} \textbf{read } x.P \\
& \quad \text{iff } \forall (T\theta, n) \in \sigma\,.\,(\kappa,\sigma) \models^{\theta\{T/x\}}_{\mathcal{G}(\tau)} P \\
\text{(case)} & (\kappa,\sigma) \models^{\theta}_{\mathcal{G}(\tau),n} \textbf{case } T \textbf{ of } \mathsf{f}(T_1 \ldots T_j; x_{j+1} \ldots x_k)\ P_1 \textbf{ else } P_2 \\
& \quad \text{iff } \left(T\theta = \mathsf{f}(V_1 \ldots V_k) \wedge \bigwedge_{i=1}^{j} T_i\theta = V_i \implies (\kappa,\sigma) \models^{\theta\{V_i/x_i\}_{j+1 \le i \le k}}_{\mathcal{G}(\tau),n} P_1 \right) \\
& \qquad \wedge (\kappa,\sigma) \models^{\theta}_{\mathcal{G}(\tau),n} P_2
\end{array}
$$

Table 3.17: Control Flow Analysis in CBS$^\sharp$, excerpt.

the behaviour of a network if and only if the same holds for each process in each parallel network component.

In the judgement of processes, (out) states that $(\kappa,\sigma) \models^{\theta}_{\mathcal{G}(\tau),n} \langle T \rangle.P$ holds if and only if κ accounts for the output (T,n), and $(\kappa,\sigma) \models^{\theta}_{\mathcal{G}(\tau),n} P$ holds. (in) states that $(\kappa,\sigma) \models^{\theta}_{\mathcal{G}(\tau),n} (x).P$ holds if and only if $(\kappa,\sigma) \models^{\theta\{T/x\}}_{\mathcal{G}(\tau),n} P$ holds for any T that can be sent to n (and thus may be written to x). The judgements for (store) and (read) are analogous. Lastly, (case) states that $(\kappa,\sigma) \models^{\theta}_{\mathcal{G}(\tau),n}$ **case** T **of** $\mathsf{f}(T_1 \ldots T_j; x_{j+1} \ldots x_k)$ P_1 **else** P_2 holds if θ is a unifier for the system of equations

$$S = \{T \overset{?}{=} \mathsf{f}(V_1 \dots V_k)\} \cup \{T_i \overset{?}{=} V_i\}_{1 \le i \le j}$$

with variables $\mathsf{v}(T) \cup \bigcup_{1 \le i \le j} \mathsf{v}(T_i) = \mathrm{dom}(\theta)$, $(\kappa, \sigma) \models^{\theta\{V_i/x_i\}_{j+1 \le i \le k}}_{\mathcal{G}(\tau),n} P_1$ holds (with θ extended with the proper pattern-matching instantiation of $x_{j+1} \dots x_k$), and the failure case $(\kappa, \sigma) \models^{\theta}_{\mathcal{G}(\tau),n} P_2$ holds. There are also judgements for the process constructs $P_1 \mid P_2$, $!P$, and $\mathbf{0}$ (the remainder of the grammar of $\mathscr{P}$), and they are fairly straightforward (see [NH06]).

Soundness Results

The judgement of networks and processes, respectively, is then proven sound [NH06].

Theorem 3.7 (Soundness of (κ, σ) w.r.t. N)
If $N \rightarrow^*_\tau N' \xrightarrow{(T,n)!}_{G \in \tau} N''$ *and* $(\kappa, \sigma) \models_{\mathcal{G}(\tau)} N$, *then* $(T, n) \in \kappa$. □

Theorem 3.8 (Soundness of (κ, σ) w.r.t. P)
For any $T \neq \epsilon$, *we have that if* $N \Downarrow^\tau_n T$ *and* $(\kappa, \sigma) \models_{\mathcal{G}(\tau)} N$, *then* $(T, n) \in \sigma$. □

Here, $N \Downarrow^\tau_n T$ denotes that "in network N, T can at some point be stored in location n under network topology τ".

Implementation

The control flow analysis for CBS$^\sharp$ has been implemented using the *Succinct Solver* constraint solving tool [NNS$^+$04], version 1.0 [NH06, Nan06]. Constraints are specified in ALFP logic, a Horn-like fragment of first-order predicate logic.

Security Model for Routing Protocols

At last, a security model *for routing protocols* is given in [NH06]. It is based on the concept of *mediated equivalence*. There, a *mediator* is a function on terms (that is, $\mu : \mathcal{T} \longrightarrow \mathcal{T}$). *Typically*, μ functions as a "filter", which maps undesired terms to ϵ (ϵ can always be stored, everywhere), and any other term to itself (such mediators are called *simple*).

Definition 3.31 (Mediated Equivalence)
Given a network topology τ and a mediator μ, we write $N \sqsubseteq^\mu_\tau M$ if

$$\forall T \,.\, \forall n \in \mathsf{V}(N) \cap \mathsf{V}(M) \,.\, N \Downarrow^\tau_n T \implies M \Downarrow^\tau_n \mu(T)$$

holds. Two networks N and M are *mediated equivalent*, written $N \simeq^\mu_\tau M$, if and only if $N \sqsubseteq^\mu_\tau M$ and $M \sqsubseteq^\mu_\tau N$. □

The idea with $\simeq_{\tau}^{\mu}$ is that we are not concerned with whether "false routing information" is communicated from one node to another or not (in which case we would consider the more classic notion of bisimilarity), but rather whether that false routing information is *stored* in that which corresponds to a routing table in CBS$^{\sharp}$: The local storage S.

In the context of routing protocols, a *consistency mediator* $\mu_{\tau} : \mathcal{T} \longrightarrow \mathcal{T}$ maps $\mu_{\tau}(T)$ to ϵ when τ is "correctly represented", and to T otherwise. The "correctly represented" condition varies from protocol to protocol.

Example 3.9 (Consistency Mediator in the SAODV protocol)
In the Secure Ad-hoc On-demand Distance Vector (SAODV) routing protocol, $\mu_{\tau}^{\text{SAODV}}$ could be defined as

$$\mu_{\tau}^{\text{SAODV}}(U) = \begin{cases} \epsilon, & \text{if for } U = \mathsf{Route}(n_d, n_1, n_2) \text{ there exist locations } n_3, \ldots, n_k \text{ s.t.} \\ & n_k = n_d \text{ and } \forall 1 \leq i \leq k-1 \exists G \in \tau \,.\, (n_i, n_i + 1) \in G \\ U & \text{otherwise.} \end{cases}$$

Here, $\mathsf{Route}(n_d, n, n')$ is a term (a predicate) expressing that whenever n receives a message addressed to n_d, then n will forward that message to n'. Thus, $\mu_{\tau}^{\text{SAODV}}$ filters out unwanted routing information (false routes). □

Definition 3.32 (Topology Consistency)
Given a network N and a network topology τ, let μ_{τ} be the consistency mediator for N. N is said to be *topology consistent* under network M (an attacker) if $N \simeq_{\tau}^{\mu_{\tau}} N \mid M$. □

Due to a result in [NH06] stating that $N \sqsubseteq_{\tau}^{\mu} N \mid M$ for any two networks N, M and simple mediators μ (and since the converse is false in general), then we can note that in Definition 3.32, M can store anything, while if $N \mid M$ stores undesired data (defined by μ_{τ}) in the nodes of N, then $N \simeq_{\tau}^{\mu_{\tau}} N \mid M$ cannot be established.

At last, we have that

Theorem 3.10 (Soundness of (κ, σ) w.r.t. Topological Consistency)
For all networks N, M and simple mediators μ, if $(\kappa, \sigma) \models_{\mathcal{G}(\tau)} N \mid M$ and $\forall (T, n) \in \sigma \,.\, \mu(T) = \epsilon$, then $N \simeq_{\tau}^{\mu} N \mid M$. □

It is then proven in [NH06] by use of this result that the modelled SAODV protocol is vulnerable to spoofing attacks.

3.4 CMAN

The Calculus for Mobile Ad-hoc Networks (CMAN) [God07] is a broadcast calculus developed by Jens Chr. Godskesen, for the purpose of modelling cryptographic routing protocols. In CMAN, a protocol is modelled as a network of locations, each containing a single sequential process. While a process can never leave its location, a process *can* actively manipulate the network topology, and as such, obtain a form of migration. The network topology is two-way, and is expressed explicitly in the syntax of the model, with each location having a set of adjacent locations associated with it. CMAN employs an asynchronous send broadcast semantics, where a broadcast action is lossy and atomic. A broadcast of a message is scoped by the network topology in the sense that only locations adjacent to the broadcasting location receive the message.

3.4.1 Syntax and Semantics

Syntax

Let $\mathcal{N}$ be the set of names ranged over by $a, b, c, o, \ldots, t$, with nodes (also referred to as locations) ranged over by $k, l, n \in \mathcal{N}_{\mathrm{loc}}$, where $\mathcal{N} \cap \mathcal{N}_{\mathrm{loc}} = \emptyset$. Let names and nodes be ranged over by $m \in \mathcal{N} \cup \mathcal{N}_{\mathrm{loc}}$, and $\mathcal{V}$ be the set of variables ranged over by $x, \ldots, y$ (both sets being infinite). Let $\mathcal{N} \cup \mathcal{V} = \mathcal{U}$ be ranged over by $u, \ldots, w$. Let $\mathcal{Z}$ be the set of process variables, ranged over by z. Let $\mathcal{F}$ be the set of function symbols, ranged over by f, and $\mathcal{G}$ be the set of destructors, ranged over by g. The syntax of CMAN is then given by the grammar specification in Table 3.18, where the syntactic category N represents

$$
\begin{array}{rll@{\qquad\qquad}rll}
P & ::= & \mathbf{0} & T & ::= & u \\
 & \mid & (x).P & & \mid & \mathsf{f}(\tilde{T}) \\
 & \mid & \langle T \rangle.P & & & \\
 & \mid & (\nu a)P & N & ::= & \mathbf{0} \\
 & \mid & \mathbf{let}\ x = \mathsf{g}(\tilde{T})\ \mathbf{in}\ P\ \mathbf{else}\ P & & \mid & n[P]^{\sigma} \\
 & \mid & \mathbf{let}\ x = T\ \mathbf{in}\ P & & \mid & (\nu m)N \\
 & \mid & \mathbf{if}\ T = U\ \mathbf{then}\ P\ \mathbf{else}\ P & & \mid & N \mid N \\
 & \mid & \mathbf{rec}\ z.P & & & \\
 & \mid & z & & &
\end{array}
$$

Table 3.18: Syntax of CMAN.

networks, which are ranged over by $M, \ldots, N \in \mathscr{N}$, the syntactic category T represents *terms*, which are ranged over by $T, \ldots, V \in \mathcal{T}$, and the syntactic category P represents *processes*, which are ranged over by $P, \ldots, R \in \mathscr{P}$.

The recursion operator, which is the only new syntactic element compared to the calculi reviewed prior, is defined as recursion is normally defined. This also makes sense since a process variable z can appear nowhere else than in the end of a process specification (as z cannot be postfixed).

Worth noting also is that there is no parallel composition of processes; only of networks. The σ in the syntax denotes the (symmetric) connection of n to all the nodes with names in σ.

We define $\mathsf{fl}(N)$ and $\mathsf{bl}(N)$ to be the free and bound locations in N, respecfively, and define $\mathsf{l}(N) = \mathsf{fn}(N) \cup \mathsf{bn}(U)$.

We assume that all networks are *well-formed*, which is defined as follows.

Definition 3.33 (Well-formedness)
We define *well-formedness* of networks inductively as follows.

i) $n[P]^\sigma$ is well-formed if $l \notin \sigma$.

ii) $N \mid M$ is well-formed if N and M are, and if $\mathsf{fl}(N) \cap \mathsf{fl}(M) = \emptyset$.

iii) $(\nu m)N$ is well-formed if N is. □

As we are solely interested in well-formed, and variable-closed networks, we shall henceforth let $\mathscr{N}$ denote the set of these networks.

Rewrite System

As Aπ does with algebraic theories, CMAN generalises the data language to *term rewrite systems*, which can vary from application to application.

Semantics

Definition 3.34 (Structural Congruence on $\mathscr{P}$)
The *structural congruence* relation on processes, denoted $\equiv_{\mathscr{P}}$, is the least congruence and equivalence relation on $\mathscr{P}$ that is closed under α-conversion and satisfies the axioms in Table 3.19. □

(rec)
$$\frac{}{\textbf{rec } z.P \equiv_{\mathscr{P}} P\{^{\textbf{rec } z.P}/_{z}\}}$$

(assign)
$$\frac{}{\textbf{let } x = T \textbf{ in } P \equiv_{\mathscr{P}} P\{^{T}/_{x}\}}$$

(if$_1$)
$$\frac{}{\textbf{if } T = T \textbf{ then } P \textbf{ else } Q \equiv_{\mathscr{P}} P}$$

(if$_2$)
$$\frac{T \neq U}{\textbf{if } T = U \textbf{ then } P \textbf{ else } Q \equiv_{\mathscr{P}} Q}$$

(let$_1$)
$$\frac{\mathsf{g}(\tilde{T}) > T}{\textbf{let } x = \mathsf{g}(\tilde{T}) \textbf{ in } P \textbf{ else } Q \equiv_{\mathscr{P}} P\{^{T}/_{x}\}}$$

(let$_2$)
$$\frac{\nexists T \,.\, \mathsf{g}(\tilde{T}) > T}{\textbf{let } x = \mathsf{g}(\tilde{T}) \textbf{ in } P \textbf{ else } Q \equiv_{\mathscr{P}} Q}$$

Table 3.19: Structural Congruence on $\mathscr{P}$ in CMAN.

Here, a relation $\mathcal{R}$ on $\mathscr{P}$ is said to be congruent if, for any evaluation context $C[\cdot]$ and any processes P, Q, we have $P\mathcal{R}Q \implies C[P]\mathcal{R}C[Q]$.

Definition 3.35 (Structural Congruence on $\mathscr{N}$)
The *structural congruence* relation on networks, denoted $\equiv$, is the least congruence and equivalence relation on $\mathscr{N}$ that is closed under α-conversion and satisfies the axioms in Table 3.20. □

(nil) $$\frac{}{N \mid \mathbf{0} \equiv N}$$

(comm) $$\frac{}{N \mid M \equiv M \mid N}$$

(assoc) $$\frac{}{(N_1 \mid N_2) \mid N_3 \equiv N_1 \mid (N_2 \mid N_3)}$$

(cong$_{\mathscr{P}}$) $$\frac{P \equiv_{\mathscr{P}} Q}{n[P]^{\sigma} \equiv n[Q]^{\sigma}}$$

(new-ext) $$\frac{}{n[(\nu a)P]^{\sigma} \equiv (\nu a)n[P]^{\sigma}}$$

(new-C) $$\frac{}{(\nu m)(\nu m')N \equiv (\nu m')(\nu m)N}$$

(new-par) $$\frac{m \notin \mathsf{fn}(Q) \cup \mathsf{fl}(Q)}{(\nu m)N \mid M \equiv (\nu m)(N \mid M)}$$

Table 3.20: Structural Congruence on $\mathscr{N}$ in CMAN.

Here, a relation $\mathfrak{R}$ on $\mathscr{N}$ is said to be congruent if $N\mathfrak{R}N'$ implies that $(\nu m)N\mathfrak{R}(\nu m)N'$ for all m, and $N \mid M\mathfrak{R}N' \mid M$, for all M with $\mathsf{fl}(M) \cap (\mathsf{fl}(N) \cup \mathsf{fl}(N')) = \emptyset$.

Next we have the reduction relation, which expresses computation in processes.

Definition 3.36 (Reduction Relation)
The *reduction relation*, denoted $\rightarrow$, is the least relation on $\mathscr{N}$ closed under $\equiv$ and $|$, and satisfying the axioms in Table 3.21. □

(conn) $$\frac{}{n[P]^{\sigma} \mid l[Q]^{\sigma'} \rightarrow n[P]^{\sigma l} \mid l[Q]^{\sigma' n}}$$

(disc) $$\frac{}{n[P]^{\sigma l} \mid l[Q]^{\sigma' n} \rightarrow n[P]^{\sigma} \mid l[Q]^{\sigma'}}$$

(bc) $$\frac{}{n[\langle T\rangle.P]^{\sigma\sigma'} \mid \prod_{l\in\sigma} l[(x).P_l]^{\sigma_l n} \rightarrow_n n[P]^{\sigma\sigma'} \mid \prod_{l\in\sigma} l[P_l\{T/x\}]^{\sigma_l n}}$$

(scope) $$\frac{N \rightarrow_n N'}{(\nu m)N \rightarrow_n (\nu m)N'} \quad \text{where} \quad m \neq n$$

(hide) $$\frac{N \rightarrow_n N'}{(\nu n)N \rightarrow (\nu n)N'}$$

Table 3.21: Reduction in CMAN.

Recall that σ is a set of location names. We denote the set $\sigma \cup \{n\}$, where $n \notin \sigma$, by σn.

Definition 3.37 (Labelled Transition Relation for $\mathscr{P}$ [NH06])
The *labelled transition relation for processes*, denoted by $\xrightarrow{\lambda}_{\mathscr{P}}$, is the least relation on $\mathscr{P}$ closed by $\equiv_{\mathscr{P}}$ and satisfying the axioms in Table 3.22 □

(out) $$\frac{}{\langle T\rangle.P \xrightarrow{\langle T\rangle}_{\mathscr{P}} P}$$

(in) $$\frac{}{(x).P \xrightarrow{(T)}_{\mathscr{P}} P\{T/x\}}$$

(scope) $$\frac{P \xrightarrow{\lambda}_{\mathscr{P}} P'}{(\nu a)P \xrightarrow{\lambda}_{\mathscr{P}} (\nu a)P'} \quad \text{where} \quad a \notin \mathsf{n}(\lambda)$$

(open) $$\frac{P \xrightarrow{(\nu\tilde{a})\langle T\rangle}_{\mathscr{P}} P'}{(\nu a)P \xrightarrow{(\nu a\tilde{a})\langle T\rangle}_{\mathscr{P}} P'} \quad \text{where} \quad a \in \mathsf{fn}(T) \setminus \tilde{a}$$

Table 3.22: Labeled Transition for $\mathscr{P}$ in CMAN.

Here, λ ranges over the set of *process actions*, $\mathbf{A}_{\mathscr{P}}$, defined by the grammar

$$\lambda ::= (T) \mid (\nu\tilde{a})\langle T\rangle,$$

where we abreviate $(\nu\emptyset)\langle T\rangle$ by $\langle T\rangle$. We then have that the operational semantics for processes is defined as the labelled transition system $(\mathscr{P}, \mathbf{A}_{\mathscr{P}}, \xrightarrow{\lambda}_{\mathscr{P}})$, where $\xrightarrow{\lambda}_{\mathscr{P}} \subseteq (\mathscr{P} \times \mathbf{A}_{\mathscr{P}}) \times \mathscr{P}$.

Definition 3.38 (Labelled Transition Relation for $\mathscr{N}$ [NH06])
The *labelled transition relation for networks*, denoted by $\xrightarrow{\alpha}_{\mathscr{N}}$, is the least relation on $\mathscr{N}$ satisfying the axioms in Tables 3.23 and 3.24. □

Here, α ranges over the set of *network actions*, $\mathbf{A}_{\mathscr{N}}$, defined by the grammar

$$\alpha ::= \beta \mid \gamma \quad \beta ::= \overline{n} \mid \overline{n}.\sigma(\nu\tilde{a})\langle T\rangle \mid \overline{n}.\sigma(T) \mid \tau \quad \gamma ::= n\triangleright \mid (\nu n)n\triangleright \mid n\triangleleft l \mid \tau$$

The syntactic category β represents broadcast actions. Here, when considering these labels, it is helpful to consider the labelled reduction as occuring in some specified network N, and that these labelled reduction symbolise the interaction of N with some environment, in the form of an unspecified evaluation context $C[\cdot]$ (that is, where any labelled reduction in N is an (internal) reduction in $C[N]$). In β, $\overline{n}$ means that the process in location n in N broadcasts a message, and that this message does not reach the environment. By only looking at the label $\overline{n}$, and without looking inside N[7], we cannot say from this point of view which location(s) received the message broadcast by n, if any at all. The action $\overline{n}.\sigma(\nu\tilde{a})\langle T\rangle$ is an output action, meaning that the process at location n has broadcast the message T, containing names $\tilde{a}$, which is *receivable* by locations in σ. σ is always a subset of the names associated with n in N as the locations which can receive from n. Lastly, $\overline{n}.\sigma(T)$ is an input action, meaning that the process at location n broadcasts the message T which can be received by locations in σ. Again, σ is a subset

[7]and without monitoring changes in behaviour of N, and without comparing N to N', where $N \xrightarrow{\overline{n}}_{\mathscr{N}} N'$

(bc) $$\frac{P \xrightarrow{(\nu\tilde{a})\langle T\rangle}_{\mathscr{P}} P'}{n[P]^{\sigma} \xrightarrow{\overline{n}.\sigma(\nu\tilde{a})\langle T\rangle}_{\mathscr{N}} n[P']^{\sigma}}$$

(lose) $$\frac{N \xrightarrow{\overline{n}.(\sigma\sigma')(\nu\tilde{a})\langle T\rangle}_{\mathscr{N}} N'}{N \xrightarrow{\overline{n}.\sigma(\nu\tilde{a})\langle T\rangle}_{\mathscr{N}} N'}$$

(rec$_1$) $$\frac{P \xrightarrow{(T)}_{\mathscr{P}} P'}{n[P]^{\sigma l} \xrightarrow{\overline{l}.n(T)}_{\mathscr{N}} n[P']^{\sigma l}}$$

(rec$_2$) $$\frac{N \xrightarrow{\overline{n}.\sigma(T)}_{\mathscr{N}} N' \quad M \xrightarrow{\overline{n}.\sigma'(T)}_{\mathscr{N}} M'}{N \mid M \xrightarrow{\overline{n}.\sigma\sigma'(T)}_{\mathscr{N}} N' \mid M'} \quad \text{where} \quad \sigma \cap \sigma' = \emptyset$$

(close) $$\frac{N \xrightarrow{\overline{n}\epsilon(\nu\tilde{a})\langle T\rangle}_{\mathscr{N}} N'}{N \xrightarrow{\overline{n}}_{\mathscr{N}} (\nu\tilde{a})N'} \quad \text{where} \quad T \text{ is some term.}$$

(hide) $$\frac{N \xrightarrow{\overline{n}}_{\mathscr{N}} N'}{(\nu n)N \xrightarrow{\tau}_{\mathscr{N}} (\nu n)N'}$$

(open$_1$) $$\frac{N \xrightarrow{\overline{n}.\sigma(\nu\tilde{a})\langle T\rangle}_{\mathscr{N}} N'}{(\nu a)N \xrightarrow{\overline{n}.\sigma(\nu\tilde{a}a)\langle T\rangle}_{\mathscr{N}} N'} \quad \text{where} \quad a \in \mathsf{fn}(T) \setminus \tilde{a}$$

(sync) $$\frac{N \xrightarrow{\overline{n}.\sigma\sigma'(\nu\tilde{a})\langle T\rangle}_{\mathscr{N}} N' \quad M \xrightarrow{\overline{n}.\sigma'(T)}_{\mathscr{N}} M'}{N \mid M \xrightarrow{\overline{n}.\sigma(\nu\tilde{a})\langle T\rangle}_{\mathscr{N}} N' \mid M'} \quad \text{where} \quad \tilde{n} \cap \mathsf{fn}(M) = \sigma \cap \mathsf{fl}(M) = \emptyset$$

(par$_1$) $$\frac{N \xrightarrow{\beta}_{\mathscr{N}} N'}{N \mid M \xrightarrow{\beta}_{\mathscr{N}} N' \mid M} \quad \text{where} \quad \mathsf{l}(Q) \cap \mathsf{l}(\beta) = \emptyset$$

(res$_1$) $$\frac{N \xrightarrow{\beta}_{\mathscr{N}} N'}{(\nu m)N \xrightarrow{\beta}_{\mathscr{N}} (\nu m)N'} \quad \text{where} \quad m \notin \mathsf{n}(\beta) \cup \mathsf{fl}(\beta)$$

Table 3.23: Labelled Transition for $\mathscr{N}$ for broadcast in CMAN

of the location names associated with n as the locations within broadcast range. Note that n need not be part of N; n could be a location in the environment $C[\cdot]$, to which some locations in N are connected.

Thus, the role of σ is to keep track of possible, and legitimate, receivers of a broadcast message. Since the network topology is explicit in the syntax, as opposed to being provided alongside the network specification, like the graphs in CBS$^{\sharp}$, we cannot perform a "look up" in the network topology on demand in the labelled reduction rules in Table 3.23. Instead, σ is "carried around" during derivation, travelling up from the leaf in the abstract syntax tree of N where the broadcast stems from, and parted up and

(con$_1$)
$$\frac{}{n[P]^{\sigma} \xrightarrow{n\triangleright}_{\mathscr{N}} n[P]^{\sigma}}$$

(dis$_1$)
$$\frac{}{n[P]^{\sigma l} \xrightarrow{n\triangleleft l}_{\mathscr{N}} n[P]^{\sigma}}$$

(con$_2$)
$$\frac{N \xrightarrow{(\nu\tilde{n})n\triangleright}_{\mathscr{N}} N' \quad N \xrightarrow{(\nu\tilde{l})l\triangleright}_{\mathscr{N}} N'}{N \mid M \xrightarrow{\tau}_{\mathscr{N}} (\nu\tilde{n})(\nu\tilde{l})(N' \mid M')_{n\oplus l}} \quad \text{where} \quad (*)$$

(*)
$$\tilde{n} \cap \tilde{l} = \tilde{n} \cap \mathsf{fl}(N) = \tilde{l} \cap \mathsf{fl}(M) = \emptyset, \text{ and } \tilde{n} \in \{\{n\}, \emptyset\}, \tilde{l} \in \{\{l\}, \emptyset\}$$

(dis$_2$)
$$\frac{N \xrightarrow{n\triangleleft l}_{\mathscr{N}} N' \quad M \xrightarrow{l\triangleleft n}_{\mathscr{N}} M'}{N \mid M \xrightarrow{\tau}_{\mathscr{N}} N' \mid N'} \quad \text{where} \quad n \in \mathsf{l}(N), l \in \mathsf{l}(M)$$

(open$_2$)
$$\frac{N \xrightarrow{n\triangleright}_{\mathscr{N}} N'}{(\nu n)N \xrightarrow{(\nu n)n\triangleright}_{\mathscr{N}} N'}$$

(res$_2$)
$$\frac{N \xrightarrow{\gamma}_{\mathscr{N}} N'}{(\nu m)N \xrightarrow{\gamma}_{\mathscr{N}} (\nu m)N'} \quad \text{where} \quad m \notin \mathsf{l}(\gamma)$$

(par$_2$)
$$\frac{N \xrightarrow{\gamma}_{\mathscr{N}} N'}{N \mid M \xrightarrow{\gamma}_{\mathscr{N}} N' \mid M} \quad \text{where} \quad \mathsf{bl}(\gamma) \cap \mathsf{fl}(M) = \emptyset$$

Table 3.24: Labelled Transition for $\mathscr{N}$ for mobility in CMAN

"sent down" the branches down to nodes which can hear n, along the way up the abstract syntax tree. This is expressed in the rules (bc), (lose), (rec$_1$), (rec$_2$), and (sync)[8]. The purpose of (close) and (open$_1$) is to move the scope of restricted names as necessary (sometimes beyond the scope of N, effectively "free"ing them). The remaining rules are structural.

The syntactic category γ represents migration. There, the actions $n\triangleright$ and $(\nu n)n\triangleright$ mean that n might migrate. The action $n\triangleleft l$ expresses that the nodes n and l disconnect from one another. An important rule to note in Table 3.24 is (con$_2$), which, while it does not seem so from its premises, connects two locations. The connection occurs by using the *connection* operator, $n \oplus l$, on $N \mid M$, which connects n and l in $N \mid M$. Formally, for any N, l, and n, we define $N_{n\oplus l}$ inductively in Table 3.25. There is also a *disconnection* operator, $n \ominus l$, which separates n and l in the affected network.

We then have that the operational semantics for networks is defined as the labelled transition system $(\mathscr{N}, \mathbf{A}_{\mathscr{N}}, \xrightarrow{\alpha}_{\mathscr{N}})$, where $\xrightarrow{\alpha}_{\mathscr{N}} \subseteq (\mathscr{N} \times \mathbf{A}_{\mathscr{N}}) \times \mathscr{N}$.

[8]If the choice of σs in the inference rules seems "precognitive", then try looking at the rules as described previously in this paragraph: Top-down for output rules like (bc) and (lose), until you match the premise of a rule which has an input premise, like (sync). Match these in a bottom-up manner, and see what happens to the σs.

$$\begin{aligned}
\mathbf{0}_{n\oplus l} &= \mathbf{0}\\
(n[P]^{\sigma})_{n\oplus l} &= n[P]^{\sigma l}\\
(l[P]^{\sigma})_{n\oplus l} &= l[P]^{\sigma n}\\
(k[P]^{\sigma})_{n\oplus l} &= k[P]^{\sigma},\ \text{if } k \notin \{n,l\}\\
(N \mid M)_{n\oplus l} &= N_{n\oplus l} \mid M_{n\oplus l}\\
((\nu m)N)_{n\oplus l} &= (\nu m)(N_{n\oplus l}),\ \text{if } m \notin \{n,l\}
\end{aligned}$$

Table 3.25: Connection operator, $\oplus$, in CMAN

3.4.2 Equivalence

At last we present the notion of strong bisimilarity for CMAN. First, let

$$(x).A_{\sigma\oplus l} = \prod_{k\in\sigma} k[(x).P_k]^{\sigma_k l}.$$

We then, for any $(x).A_{\sigma\oplus l}$, write

$$A_{\sigma\oplus l}\{T/x\} = \prod_{k\in\sigma} k[P_k\{T/x\}]^{\sigma_k l}.$$

Definition 3.39 (Strong Bisimilarity)
A binary relation $\mathcal{R}$ on $\mathscr{N}$ is a *strong simulation* if $N\mathcal{R}M$ implies $\mathsf{fl}(N) = \mathsf{fl}(M)$, and for all $P \in \mathscr{P}$,

i) if $N \xrightarrow{\tau} N'$, then $\exists M'$ such that

$$M \xrightarrow{\tau} M' \wedge N'\mathcal{R}M'$$

ii) if $N \xrightarrow{\bar{l}.\sigma(\nu\tilde{n})\langle T\rangle} N'$, then $\forall\sigma' \subseteq \sigma \,.\, \forall(x).A_{\sigma'\oplus l} \,.\, \tilde{n} \cap \mathsf{fn}((x).A_{\sigma'\oplus l}) = \emptyset$, we have

$$M \mid (x).A_{\sigma'\oplus l} \xrightarrow{\bar{l}} M' \wedge (\nu\tilde{n})(N' \mid A_{\sigma\oplus l}\{T/x\})\mathcal{R}M'$$

iii) if $N \xrightarrow{\bar{l}.\sigma(T)} N'$, then $\forall\sigma' \,.\, \sigma' \cap \sigma l = \emptyset \,.\, \exists M'$, such that

$$M \mid l[\langle T\rangle P]^{\sigma\sigma'} \xrightarrow{\bar{l}} M' \wedge N' \mid l[\langle T\rangle P]^{\sigma\sigma'}\mathcal{R}M'$$

iv) if $N \xrightarrow{(\nu m)l\triangleright} N'$, then $\forall k \,.\, k \notin \mathsf{fl}(N) \cup \tilde{m} \,.\, \forall\sigma \,.\, \sigma \cap \tilde{m}k = \emptyset \,.\, \exists M'$, such that

$$M \mid k[P]^{\sigma} \xrightarrow{\tau} M' \wedge (\nu\tilde{m})(N' \mid k[P]^{\sigma}_{l\oplus k})\mathcal{R}M'$$

v) if $N \xrightarrow{l \triangleleft k} N'$, then $\forall \sigma \,.\, k \notin \sigma \,.\, \exists M'$, such that

$$M \mid k[P]^{\sigma l} \xrightarrow{\tau} M' \wedge N' \mid k[P]^{\sigma} \mathcal{R} M'$$

$\mathcal{R}$ is then a *strong bisimilarity* if $\mathcal{R}^{-1}$ is also a strong simulation. □

A corresponding definition for weak bisimilarity can be found in [God07].

Due to the contextuality of his bisimilarity definition, Godskesen restricts his attention to connection-closed networks.

Definition 3.40 (Connection-closed Network)
A network N is closed when each node in N is connected only to other nodes in N. □

Godskesen then applies his framework to recapture the flaw in the ARAN protocol he discovered in [God06]. This is done manually by use of behavioural equivalence checks of the ARAN model with and without a specific attacker.

3.5 Summary

We now thoroughly reflect upon the related frameworks presented in this chapter, comparing them to each other and to our objective, and emphasise what we desire from our framework as we go.

Aπ: The Aπ calculus is indeed a well-established, very flexible framework for the verification of security protocols. Secrecy and authenticity properties assume the tried-and-true Dolev-Yao threat model, which we deem to be a fairly sound set of assumptions. The appeal of concept of a frame, and deduction from a frame, is also that we are capable of expressing the knowledge that an arbitrary, *unspecified*, attacker can achieve, simply by analysing the syntax specification of our protocol model. This thus leads to a sound set of secrecy and authenticity definitions, which we would like to inherit to our calculus. The generalisation of the applied data language greatly increases the applicability of Aπ, as we can simply instantiate it as we see fit for a given modelling scenario.

However, were we to attempt to apply Aπ directly for the verification of MANETs, we encounter a few problems. First and foremost, the message-passing semantics of Aπ is point-to-point. Since we are interested in a broadcast semantics, and since it has been proven that true broadcast cannot be expressed in the π-calculus, and thus not in Aπ [EM99], we will need to at least give a broadcast semantics to Aπ.

However, we will soon discover that broadcast alone is not sufficient, since not all network topologies can be expressed neatly as a tree structure. For instance, let the names a and b represent links between the pairs of processes A_1 and A_2, and A_2 and A_3, respectively. Now, it is impossible to specify such a process such that none of A_1, A_2, and A_3 are caught under both restrictions a and b. As such, we will need to find a different approach towards representing the network topology.

ABπ: While ABπ bears much resemblance to Aπ, it has a couple of limitations in comparison. First, the data language is not arbitrary. However, we are fairly convinced that this is simply because Abadi and Blanchet are proving specific secrecy and authenticity results, and therefore regard a specific data language which provides the necessary primitives. Therefore, the data language could be replaced by any term rewrite system, altough doing so might make some of the results from [AB02] void.

The other peculiarity we note is that subjects of input and output are specified as being terms. In [AB02], the authors explain that when synchronisation occurs over subjects written as a term other than a name, the term must reduce to a name, or else the process fails. However, since destructors are never present in terms, any term which is not a name *cannot* reduce to a name, *without* applying destructors on the terms. This is not permitted in the semantics of ABπ.

Despite this, the automated verification technique provided in [AB02, Bla02] is very appealing, as we feel that it can be generalised to reason not only about secrecy, but about any conditions one would like to place on the control flow of a process.

CMAN: The arbitrary term rewrite system which can be applied as a data language in CMAN makes CMAN quite applicable, as most interesting data languages, particularly in our focus of secrecy and authenticity, can be formulated as such. Also, the neighbourhood-extension to network nodes allows the nodes to move during computation, thus dynamically changing the network topology.

In CMAN, we note that **let** $x = T$ **in** P is really just syntactic sugar for **let** $x = \mathsf{dummy}(T)$ **in** P **else** $\mathbf{0}$, where dummy is defined by the rule $r := \mathsf{dummy}(x) > x$. Also, like in AB$\pi$, the conditional **if** $T = U$ **then** P **else** Q is really just syntactic sugar for **let** $x = \mathsf{equals}(T, U)$ **in** P **else** Q, where equals is defined by the rule $r := \mathsf{equals}(x, x) > x$, and where $x \notin \mathsf{fv}(P)$. If these syntactic elements were defined as syntactic sugar instead, it would make the semantics of CMAN slightly easier to specify and understand.

Which brings us to our first critique of CMAN. We find that the semantics, while capturing all the intended network reductions of interest, is unappealling. This is due to the number of relations and the number of inference rules associated with each relation. The two main reasons for the vastness of the semantics of CMAN is the neighbourhood relation, and the fact that arbitrary terms are sent instead of variable names. The former, since the network topology is explicit in the syntax and thus not known at the start of reduction, requires that network topology information be carried along during reduction. This introduces undesired overhead, particularly since verification with CMAN is done manually. The latter requires that any restrictions associated with potential names in a broadcast term be moved to an appropriate location in the abstract syntax tree of the resulting network. However, if we regard the labelled reduction relation of Aπ, we see that restrictions are only extruded in the event that the element sent is the restricted object in question, in which case the restriction is simply removed. This (very desirable) simplicity is obtained by only permitting a name or a variable to be sent to the environment during a labelled reduction.

Our next comment involves the network layer of CMAN. Since parallel composition is not permittet in a process embedded in a network, the expressiveness of embedded processes is reduced significantly. For instance, making an embedded process which is *always* ready to receive input, even when computing on some input, is not always possible. Such a process P would need to be parted up into different locations. However, since different locations represent two nodes separated in space in CMAN, the parallel components of P can "drift" to such extent that they cannot send messages between each other.

Also, as Godskesen notes in the conclusion of [God07], the complexity of the labels of CMAN causes the bisimilarity definitions of CMAN to be difficult to handle and apply. While this can partly be solved by regarding only connection-closed networks, this makes it impossible to regard the environment as an attacker. As such, attackers must be specified in the analysis, which we wish to avoid. Instead, we are interested in being able to regard the *environment* as a hostile entity sending messages into the network, and receiving messages from the network.

CBS$^\sharp$: This calculus treats knowledge in a notably different manner than most other calculi applied for secrecy and authenticity analysis, in that each node has its own local memory storage for pooling of data terms. Nanz then defines his *routing correctness properties* as being that no node stores invalid routing information, where the invalidity property varies from protocol to protocol. Nanz also separates the network topology from the syntax of the protocol specification into connectivity graphs, which makes defining his operational semantics considerably easier. Also this means that the mobility of nodes is passive, which is the usual case in by far the most of the applications of MANETs mentioned in the introduction of this thesis.

However, we argue that while CBS$^\sharp$ is excellent for verifying *routing correctness properties*, the same does not hold for *secrecy and authenticity properties*, despite what is written in [NH06]. The reason is the data language applied in CBS$^\sharp$. Pattern-matching in this manner is ill-suited for treating cryptographic primitives, as the pattern-matching mechanism does not limit the manner in which terms can be destroyed, like term rewrite systems and universal algebrae do.

For instance, this network (possibly an attacker) can decode any message it receives, thus revealing its contents to anyone within communication range.

$$n[!(x).\mathbf{case}\ x\ \mathbf{of}\ \mathsf{enc}(;x_{\text{msg}},x_{\text{pubkey}})\ \langle x_{\text{msg}}\rangle.\mathbf{0}\ \mathbf{else}\ \mathbf{0},\emptyset]$$

The next network, being able to obtain the seed used to generate a public key, recreates the private key and reveals it.

$$n[!(x).\mathbf{case}\ x\ \mathbf{of}\ \mathsf{pubkey}(;x_{\text{seed}})\ \langle \mathsf{privkey}(x_{\text{seed}})\rangle.\mathbf{0}\ \mathbf{else}\ \mathbf{0},\emptyset]$$

We deem that these unconstrained term destructions as *unacceptable* for secrecy and authenticity analysis of protocols.

Furthermore, in the static analysis of CBS$^\sharp$, since there is but a single, public, broadcast channel, the assumption must be made that any input prefix in any location n can bind any message broadcast on the network from a location which broadcast range n is in.

At last, all participants in a protocol in CBS$^\sharp$ are considered trusted. This is apparent in the definition of topology consistency, where the equivalence requires that $\mathsf{V}(N) \cap \mathsf{V}(N \mid N_{\text{attacker}}) = \emptyset$.

To summarise, what we desire from our calculus is:

- That it should be closely related Aπ, with key differences being the use of explicit locations, mobility, and broadcast.
- To inherit key definitions from Aπ and other related work, particularly the definitions of secrecy and authenticity.
- To avoid the semantic complexity present in CMAN,

Lastly, we would like to obtain an automated verification technique similar to that of [AB02, Bla02].

CHAPTER 4

Distributed Applied π Calculus with Broadcast

We now present the Distributed Applied π Calculus with Broadcast, abreviated DAπ_β. Essentially, DAπ_β is Aπ, with broadcast channels, an added abstraction layer (networks) similar to that of Dπ, and connectivity graphs, akin to those in CBS$^\sharp$. The modelling and verification of routing and security protocols for MANETs has been a driving motivation for the development of DAπ_β.

These modifications enable a more fine-grained set of criteria for process synchronisation, as is the case in MANET. For a process A within location l to be able to receive a message T across channel a from l', the following conditions must hold:

i) There must be an edge from l' to l in the connectivity graph,

ii) A must be within the scope of a,

iii) A must be listening on a.

By letting the broadcast medium and locations be names, we obtain several new features. For instance, we can have multiple broadcast frequencies by representing each as a channel name. By using restriction on channel names, we obtain secret channels, known to a select few locations, or just one location, to model synchronisation within a location.

By restricting a location name l in a given specification N, we have a guarantee that an environmental context C *cannot* impersonate l by sending a message T from l. This is desirable when we wish to express that the behaviour of l is fully trusted, and that the behaviour of l cannot be intruded upon by external factors. By not restricting l, we express that the behaviour of l can be intruded upon by an attacker, in the sense that the attacker can send/receive messages from/at location l.

Typically, when performing secrecy analysis of an Aπ specification A, all messages which have been sent are assumed to have been obtained by an attacker. For any given environment context $C[\cdot]$ in which A is placed, this assumption is an overapproximation in the secrecy analysis, as there may be cases where $C[\cdot]$ cannot obtain a given message passed by A. By using connectivity graphs in DAπ_β, we can reduce this overapproximation, as the connectivity graph expresses the connectivity of the entire network, thus scoping the range of a broadcast on free channels.

Note that the environment is assumed to receive *all* messages sent on free channels *implicitly*, even those headed for a parallel component within the protocol specification (and thus not "physically" ending up in the environment). This is a property which is slightly bothersome to express in point-to-point calculi semantics, particularly in the presense of internal reduction, like in Aπ. Consider for instance the following example from [CRZ06].

$$A \overset{\text{def}}{=} (\nu s, k, r)(\bar{c}\langle \mathsf{enc}(s,k,r)\rangle. \mid c(z).\mathbf{if}\ \mathsf{dec}(z,k) = a\ \mathbf{then}\ \bar{c}\langle \mathsf{ok}\rangle)$$

In [CRZ06], the authors claim that

$$A' \to (\nu s, k, r)\{^{\mathsf{enc}(s,k,r)}/_{z}\} \mid \mathbf{if}\ s = a\ \mathbf{then}\ \bar{c}\langle \mathsf{ok}\rangle.$$

Notice the lack of the restriction (νz). The authors proceed to assume this curious property of $\to$ in the proof of one of their lemmas, which is applied to prove their main result. We presume that the authors assume this propety to ensure that the frame of A during reduction is consistent with the Dolev-Yao threat model. However, as the π-calculus family abstracts away from routing by specifying message-passing as a direct synchronisation between two processes, the environment *cannot* implicitly receive messages. Thus, to model implicit message leakage in the simulation of the process specification, the message must be explicitly routed to a process beyond the scope of the specification by use of labelled reduction. As DAπ_β uses broadcast as a communication primitive, expressing this property becomes considerably easier than in point-to-point calculi, and we shall see how this is done later.

We start off by giving the syntax of DAπ_β, along with a definition of contexts in this new setting. Next we define connectivity graphs in our setting, which differ mainly from those in CBS$^\sharp$ in that each node is connected to itself. We then give an operational semantics, and a labelled operational semantics, where each reduction conditions on admissible graphs. We give a couple of examples, which at the same time provide us with some new structural equivalences. After that we prove a couple of normal forms, which purpose is to bring terms in active substitutions into, and out of locations, respectively. The normal forms enable us to at last present the key notion of frames in this new setting, which gives us an indistinguishability relation, and the definitions from Aπ in Section 3.1.3 defined solely in terms of deduction from frames, and indistinguishability.

4.1 Syntax

We let $\mathcal{N}$ be the set of names ranged over by $a, \ldots, c, m, \ldots, t$, and $\mathcal{V}$ be the set of variables ranged over by $x, \ldots, z$, as before, and let $\mathcal{U} = \mathcal{N} \cup \mathcal{V}$ be the set of identifiers, ranged over by $u, \ldots, w$. Let $\mathcal{F}$ be the set of constructors, ranged over by f, and $\mathcal{G}$ be the set of destructors, ranged over by g. The syntax of DAπ_β is then given by the grammar specification in Table 4.1, where P is a primitive process, which are ranged over

$$
\begin{array}{rcl@{\qquad\qquad}rcl}
T & ::= & u & A & ::= & P \\
 & \mid & \mathsf{f}(T, \ldots, T) & & \mid & A \mid A \\
 & & & & \mid & (\nu u)A \\
P & ::= & u(x).P & & \mid & \{^T/_x\} \\
 & \mid & \overline{u}\langle T\rangle.P & & & \\
 & \mid & !P & N & ::= & l[A] \\
 & \mid & (\nu a)P & & \mid & N \mid N \\
 & \mid & P \mid P & & \mid & (\nu u)N \\
 & \mid & \mathbf{let}\ x = \mathsf{g}(\tilde{T})\ \mathbf{in}\ P\ \mathbf{else}\ P & & \mid & \{^T/_x\} \\
 & \mid & \mathbf{0} & & \mid & \mathbf{0}
\end{array}
$$

Table 4.1: Syntax of DAπ.

by $P, \ldots, R \in \mathscr{P}$, A is an extended process, which are ranged over by $A, B \in \mathscr{A}$, N is a network, which are ranged over by $M, \ldots, O \in \mathscr{N}$, and T is a term, which are ranged over by $T, \ldots, V \in \mathcal{T}$.

The syntax should all be familiar from the syntax specification of the calculi presented in Chapter 3. Like in Aπ, we assume that active substitutions are cycle-free, that there is at most one active substitution for each variable, and that there is exactly one active substitution for a restricted variable.

Similarly to Aπ, we assume the presense of a *term rewrite system*, specifying how destructors reduce constructed terms.

Notice here that, and unlike ABπ, we require that subjects of input and output prefixes are identifiers. If the subject is intended to be a subterm of a term received during reduction, then this forces us to apply "let" to destroy the received term to yield the name. We shall see later, in the semantics of DAπ_β, that input and output is only defined on *names*. We assume that terms which are not names are never substituted in place of a variable subject[1]. As such, if the value of a variable subject prefixing a parallel component is *not* a name, then that parallel component is incapable of reduction.

4.1.1 Contexts

We define *contexts* in DAπ_β by the grammar specification in Table4.2, which is in many ways similar to the definition of contexts we gave for Aπ, earlier. The only thing new here is that each syntactic category has gotten a new "layer" in the grammar specification.

[1]This can be defined by adding a battery of stuctural equivalence rules for select evaluation contexts to our semantics. For instance, we would have $\{^a/_x\} \mid C[x(y).P] \equiv \{^a/_x\} \mid C[a(y).P]$ for variables in input prefixes. For variables in output objects we would have, $\{^T/_x\} \mid C[\overline{u}\langle V[x]\rangle.P] \equiv \{^T/_x\} \mid C[\overline{u}\langle V[T]\rangle.P]$ where V is a term context, and so on. These rules would then replace (subst).

Non-Evaluation Contexts:

$$\begin{array}{lcl}
C[P] & ::= & C[P] \mid M \\
 & \mid & M \mid C[P] \\
 & \mid & (\nu u)C[P] \\
 & \mid & l[C'[P]] \\
C'[P] & ::= & C'[P] \mid A \\
 & \mid & A \mid C'[P] \\
 & \mid & (\nu u)C'[P] \\
 & \mid & C''[P] \\
C''[P] & ::= & P \\
 & \mid & C''[P] \mid Q \\
 & \mid & Q \mid C''[P] \\
 & \mid & (\nu a)C''[P] \\
 & \mid & !C''[P] \\
 & \mid & \textbf{let } x = T \textbf{ in } C''[P] \textbf{ else } Q \\
 & \mid & \textbf{let } x = T \textbf{ in } Q \textbf{ else } C''[P] \\
 & \mid & u(x).C''[P] \\
 & \mid & \overline{u}\langle T\rangle.C''[P]
\end{array}$$

Evaluation Contexts:

$$\begin{array}{lcl}
C[A] & ::= & C[A] \mid M \\
 & \mid & M \mid C[A] \\
 & \mid & (\nu u)C[A] \\
 & \mid & l[C'[A]] \\
C'[A] & ::= & C'[A] \mid A \\
 & \mid & A \mid C'[A] \\
 & \mid & (\nu u)C'[A] \\
 & \mid & A
\end{array}$$

Network Contexts:

$$\begin{array}{lcl}
C[N] & ::= & N \\
 & \mid & C[N] \mid M \\
 & \mid & M \mid C[N] \\
 & \mid & (\nu u)C[N]
\end{array}$$

Table 4.2: Contexts of DAπ_β.

Network contexts are simply contexts parameterised by networks. Evaluation contexts and non-evaluation contexts are as defined in Aπ.

4.2 Connectivity Graphs

We adopt the notion of connectivity graphs from CBS$^\sharp$, except that our graphs are finite, directed, with each node being connected to itself (to allow internal reduction).

Definition 4.1 (Connectivity Graph)
A directed graph G is a connectivity graph if $\mathsf{V}(G)$ is a finite set, $\mathsf{V}(G) \subseteq \mathcal{N}_{\text{loc}}$ and $\forall l \in \mathsf{V}(G)[(l,l) \in \mathsf{E}(G)]$ holds. □

Definition 4.2 (Admissibility of Connectivity Graphs)
A connectivity graph G is admissible on a network N if $\mathsf{l}(N) \subseteq \mathsf{V}(G)$. □

That is, all the locations which appear in N appear amongst the locations in G. As such, the connectivity graph expresses the connectivity of locations in the model N, as well as other (potentially unsafe) locations which are not part of N.

Definition 4.3 (Network Topology)
A network topology τ is a set of connectivity graphs. □

Typically, we use network topologies to specify invariants on the connectivity of locations in our networks, such as

$$\{G \mid (l,l') \in \mathsf{E}(G),\ (l',l) \in \mathsf{E}(G)\},$$

which is the set of all connectivity graphs wherein the vertices l and l' are connected.

Definition 4.4 (Admissibility of Network Topologies)
A network topology τ is admissible to a network N if each graph in τ is. □

We define $\tau_{\max}$ as the set containing all graphs on $\mathcal{N}_{\text{loc}}$. Any network topology is thus a subset of $\tau_{\max}$. Like in CBS$^\sharp$, we also define an abstract network topology $\mathcal{G}(\tau)$ of a topology τ as

$$\mathcal{G}(\tau) \overset{\text{def}}{=} \left(\bigcup_{G\in\tau} \mathsf{V}(G), \bigcup_{G\in\tau} \mathsf{E}(G) \right).$$

Lastly, when we wish to disregard connectivity graphs, we use the graph abstraction $\mathcal{G}_N$, which is a complete graph on $\mathsf{l}(N)$. When we wish to model a hostile environment in a graph abstraction, we use $\mathcal{G}_N^{\text{Att}}$, which is a complete graph on $\mathsf{l}(N) \cup \{l\}$ for some $l \notin \mathsf{l}(N)$.

4.3 Semantics

Here we inherit the operational semantics of Aπ and ABπ to our calculus in the natural way.

Definition 4.5 (Structural Equivalence in DAπ_β)
Structural equivalence, denoted $\equiv$, defined on $\mathscr{N}$, extends the structural equivalence rules of Aπ in Table 3.3, except for (repl) and (rewrite), (with A's replaced by N's), with the following rules.

(nil)	$l[\mathbf{0}] \equiv \mathbf{0}$
(split)	$l[A \mid B] \equiv l[A] \mid l[B]$
(repl)	$l[!P] \equiv l[P] \mid l[!P]$
(new$_{\text{exp}}$)	$l[(\nu u)A] \equiv (\nu u)l[A]$, if $u \neq l$
(subst)	$\{{}^{T}/_{x}\} \mid l[A] \equiv \{{}^{T}/_{x}\} \mid l[A\{{}^{T}/_{x}\}]$
(exit)	$l[\{{}^{T}/_{x}\}] \equiv \{{}^{T}/_{x}\}$

□

Definition 4.6 (Internal Reduction in DAπ_β)
Internal reduction, denoted by $\rightarrow_G$, is the least preorder on $\mathscr{N}$ closed by structural equivalence and evaluation contexts, and satisfying the following rules.

(sync) $$\prod_{i\in I} l_i[a(x).P_i] \mid l[\overline{a}\langle x\rangle.P] \rightarrow_G \prod_{i\in I} l_i[P_i] \mid l[P], \text{ if } \forall i \in I \exists (l,l_i) \in \mathsf{E}(G)$$

(lt$_1$) $l[\textbf{let } x = \mathsf{g}(T_1,\dots,T_n) \textbf{ in } P \textbf{ else } Q] \rightarrow_G l[(\nu x)(\{{}^{T}/_{x}\} \mid P)]$, if $\mathsf{g}(T_1,\dots,T_n) > T$

(lt$_2$) $l[\textbf{let } x = \mathsf{g}(T_1,\dots,T_n) \textbf{ in } P \textbf{ else } Q] \rightarrow_G l[Q]$, if $\mathsf{g}(T_1,\dots,T_n) \not> T$

(repl) $l[!P] \rightarrow_G l[P \mid !P]$ □

Note that in our semantics, we are really only interested in reducing network N successfully, when the graph G we reduce N with regards to is admissible to N. We impose this limit now. Let $\mathbf{G}$ be the set of all connectivity graphs, and define the higher-order function

$$\mathsf{admit} : (\mathscr{N} \times \mathbf{G} \longrightarrow \mathscr{N}) \longrightarrow (\mathscr{N} \times \mathbf{G} \longrightarrow \mathscr{N}),$$

where $\mathsf{admit}(\rightarrow) = \rightarrow'$, where

$$\rightarrow'(N,G) = \begin{cases} \rightarrow(N,G), & \text{if } G \text{ is admissible to } N \\ \text{undefined} & \text{otherwise.} \end{cases}$$

We are now ready to define the operational semantics of Aπ.

Definition 4.7 (Operational Semantics of DAπ_β)
The *operational semantics* of DAπ_β is defined by the labelled transition system

$$(\mathscr{N}, \mathbf{G}, \mathsf{admit}(\equiv \cdot \rightarrow_G \equiv)),$$

where $\mathsf{admit}(\equiv \cdot \rightarrow_G \equiv) \subseteq (\mathscr{N} \times \mathbf{G}) \times \mathscr{N}$. □

We proceed to defining the labelled reduction relation $\xrightarrow{l,\alpha}_G$, where α is the same syntactic category as the α in Aπ.

Definition 4.8 (Labelled Reduction Relation in DAπ_β)
The *labelled reduction relation*, denoted $\xrightarrow{l,\alpha}_G$, extends $\rightarrow_G$ with the rules in Table 4.4. □

Notice that $\xrightarrow{l,\alpha}_G$ is defined in terms of $\xrightarrow{l,\alpha}\bullet_G$, defined in Table 4.3. Basically, $\xrightarrow{l,\alpha}\bullet_G$ consists of the labelled reduction rules from Aπ, adapted to networks, with a couple of modifications. The first is (par$_1$), which is a consequence of our broadcast semantics; even when the network N broadcasts from location l to the environment, there may still be locations within N, connected to l, and capable of receiving messages. The other is the slightly-peculiar (out-locscope) rule. (out-locscope) expresses that broadcasting a message does not influence the scope of the broadcasting location. Notice that location names do *not* scope the range of the broadcast message! Broadcast range is dictated by the connectivity graph, which the (in) rule conditions on. A restriction on a location name thus only functions as a security from intrusion from the environment, since when a location k is free, the environment can send messages from k. When k is bound, the rule (input) prevents the environment from sending from k.

Notice the choice of k in the rules (output), (leak), and (input) in Table 4.4. Here, k is either a name that is free in N, or not present in N. If there is such an k, we must assume that the environment is capable of receiving messages there, and sending messages from there.

Define the higher-order function

$$\mathsf{admitlab} : (\mathscr{N} \times \mathbf{G} \times \mathcal{N}_{\mathrm{loc}}\mathbf{A} \longrightarrow \mathscr{N}) \longrightarrow (\mathscr{N} \times \mathbf{G} \times \mathcal{N}_{\mathrm{loc}}\mathbf{A} \longrightarrow \mathscr{N}),$$

(in) $$\frac{}{l[a(x).P] \xrightarrow{l',a(T)}\bullet_G l[(\nu x)(\{T/x\} \mid P)]} \quad \text{where} \quad (l',l) \in \mathsf{E}(G)$$

(out) $$\frac{}{l[\overline{a}\langle u\rangle.P] \xrightarrow{l,\overline{a}\langle u\rangle}\bullet_G l[P]}$$

(par$_1$) $$\frac{N \xrightarrow{l,a(u)}\bullet_G N' \quad M \xrightarrow{l,\overline{a}\langle u\rangle}\bullet_G M'}{N \mid M \xrightarrow{l,\overline{a}\langle u\rangle}\bullet_G N' \mid M'}$$

(scope) $$\frac{N \xrightarrow{l,\alpha}\bullet_G N' \quad u \notin \{l\} \cup \mathsf{n}(\alpha)}{(\nu u)N \xrightarrow{l,\alpha}\bullet_G (\nu u)N'}$$

(out-locscope) $$\frac{N \xrightarrow{l,\overline{a}\langle u\rangle}\bullet_G N'}{(\nu l)N \xrightarrow{l,\overline{a}\langle u\rangle}\bullet_G (\nu l)N'}$$

(open-atom) $$\frac{N \xrightarrow{l,\overline{a}\langle u\rangle}\bullet_G N' \quad u \neq a}{(\nu u)N \xrightarrow{l,(\nu u)\overline{a}\langle u\rangle}\bullet_G (\nu u)N'}$$

(par$_2$) $$\frac{N \xrightarrow{l,\alpha}\bullet_G N' \quad \mathsf{bu}(\alpha) \cap \mathsf{fu}(M) = \emptyset}{N \mid M \xrightarrow{l,\alpha}\bullet_G N' \mid M}$$

Table 4.3: Dotted Transition in DAπ_β.

where $\mathsf{admitlab}(\rightarrow) = \rightarrow'$, where

$$\rightarrow'(N,G,l,\alpha) = \begin{cases} \rightarrow(N,G,l,\alpha), & \text{if } G \text{ is admissible to } N \\ \text{undefined} & \text{otherwise.} \end{cases}$$

We are now ready to define the labelled operational semantics of Aπ.

Definition 4.9 (Labelled Operational Semantics of DAπ_β)
The *labelled operational semantics* of DAπ_β is defined by the labelled transition system

$$(\mathscr{N}, (\mathbf{G} \times \mathcal{N}_{\text{loc}} \times \mathbf{A}), \mathsf{admitlab}(\equiv \cdot \xrightarrow{\alpha}_G \cdot \equiv)),$$

where $\mathsf{admitlab}(\equiv \cdot \xrightarrow{\alpha}_G \equiv) \subseteq (\mathscr{N} \times \mathbf{G} \times \mathcal{N}_{\text{loc}} \times \mathbf{A}) \times \mathscr{N}$. □

4.4 Examples

We now give a few examples of applying the semantics rules of DAπ_β, while at the same time derive some useful reductions and equivalences. First, we derive a couple of structural equivalences familiar from Aπ.

(input) $$\frac{N \xrightarrow{k,a(T)}\!\!\bullet_G N' \quad k \notin \mathsf{bl}(N) \quad \exists l \in \mathsf{l}(N)\,.\,(k,l) \in \mathsf{E}(G)}{N \xrightarrow{k,a(T)}_G N'}$$

(output) $$\frac{N \xrightarrow{l,\overline{a}\langle T\rangle}\!\!\bullet_G N' \quad \exists k \notin \mathsf{bl}(N)\,.\,(l,k) \in \mathsf{E}(G)}{N \xrightarrow{l,\overline{a}\langle T\rangle}_G N'}$$

(leak) $$\frac{N \xrightarrow{l,(\nu u)\overline{a}\langle T\rangle}\!\!\bullet_G N' \quad \exists k \notin \mathsf{bl}(N)\,.\,(l,k) \in \mathsf{E}(G)}{N \xrightarrow{l,(\nu u)\overline{a}\langle T\rangle}_G N'}$$

Table 4.4: Labelled Transition in DAπ_β.

Example 4.1 (Local (alias) **and** (subst) **equivalence)**
We show that

(alias$_{\text{loc}}$) $\quad l[(\nu x)\{T/x\}] \equiv l[\mathbf{0}]$

(subst$_{\text{loc}}$) $\quad l[\{T/x\} \mid A] \equiv l[\{T/x\} \mid A\{T/x\}].$

The first congruence follows from

$$\begin{aligned}
l[\mathbf{0}] &\equiv \mathbf{0} \mid l[\mathbf{0}] && \text{by (nil), (par-C)}\\
&\equiv (\nu x)\{T/x\} \mid l[\mathbf{0}] && \text{by (alias)}\\
&\equiv (\nu x)(\{T/x\} \mid l[\mathbf{0}\{T/x\}]) && \text{by (new-par), (subst)}\\
&\equiv (\nu x)(\{T/x\} \mid l[(\nu x)(\{T/x\} \mid \mathbf{0})]) && \text{by Example 3.1}\\
&\equiv (\nu x)\{T/x\} \mid (\nu x)l[\{T/x\} \mid \mathbf{0}] && \text{by (new-par), (new}_{\text{exp}})\\
&\equiv \mathbf{0} \mid (\nu x)(l[\{T/x\}] \mid l[\mathbf{0}]) && \text{by (alias), (split)}\\
&\equiv (\nu x)l[\{T/x\}] \mid l[\mathbf{0}] && \text{by (par-C), (nil), (par-}\mathbf{0})\\
&\equiv l[(\nu x)\{T/x\}] && \text{by (nil), (par-}\mathbf{0}\text{), (new}_{\text{exp}}),
\end{aligned}$$

while the second follows from

$$\begin{aligned}
l[\{T/x\} \mid A] &\equiv l[\{T/x\}] \mid l[A] && \text{by (split)}\\
&\equiv \{T/x\} \mid l[A] && \text{by (exit)}\\
&\equiv \{T/x\} \mid l[A\{T/x\}] && \text{by (subst)}\\
&\equiv l[\{T/x\}] \mid l[A\{T/x\}] && \text{by (exit)}\\
&\equiv l[\{T/x\} \mid A\{T/x\}] && \text{by (split).}
\end{aligned}$$
□

Next we show how (sync) and the structural equivalence rules can be used to express sending actual terms over a name, similar to what was shown for Aπ in Section 3.1.1.

Example 4.2
We show that

$$\text{(comm)} \qquad l[\overline{a}\langle T\rangle.P] \mid l[a(x).Q] \longrightarrow l[P] \mid l[Q\{^T/_x\}],$$

where $x \notin \mathsf{fv}(T) \cup \mathsf{fv}(P)$.

$$\begin{aligned}
l[\overline{a}\langle T\rangle.P] \mid l[a(x).Q] &\equiv l[\overline{a}\langle T\rangle.P] \mid l[a(x).Q] \mid \mathbf{0} && \text{by (par-}\mathbf{0}\text{)}\\
&\equiv l[\overline{a}\langle T\rangle.P] \mid l[a(x).Q] \mid l[\mathbf{0}] && \text{by (nil)}\\
&\equiv l[\mathbf{0}] \mid l[\overline{a}\langle T\rangle.P] \mid l[a(x).Q] && \text{by (par-C)}\\
&\equiv (\nu x)\{^T/_x\} \mid l[\overline{a}\langle T\rangle.P] \mid l[a(x).Q] && \text{by (alias)}\\
&\equiv (\nu x)(\{^T/_x\} \mid l[\overline{a}\langle T\rangle.P]) \mid l[a(x).Q] && \text{by (new-par)}\\
&\equiv (\nu x)(\{^T/_x\} \mid l[\overline{a}\langle x\rangle.P]) \mid l[a(x).Q] && \text{by (subst)}\\
&\equiv (\nu x)(\{^T/_x\} \mid l[\overline{a}\langle x\rangle.P] \mid l[a(x).Q]) && \text{by (new-par)}\\
&\to (\nu x)(\{^T/_x\} \mid l[P] \mid l[Q]) && \text{by (sync)}\\
&\equiv (\nu x)(\{^T/_x\} \mid l[P] \mid l[Q\{^T/_x\}]) && \text{by (subst)}\\
&\equiv (\nu x)(\{^T/_x\}) \mid l[P] \mid l[Q\{^T/_x\}] && \text{by (new-par)}\\
&\equiv \mathbf{0} \mid l[P] \mid l[Q\{^T/_x\}] && \text{by (alias)}\\
&\equiv l[P] \mid l[Q\{^T/_x\}] && \text{by (par-}\mathbf{0}\text{)}
\end{aligned}$$

□

4.5 Normal Forms

We consider two extreme normal forms with regard to the distribution of active substitutions and name restrictions in a network; one where all active substitutions are factored out of all locations to the network level, and one where all active substitutions have been factored into all locations in contact with it.

Proposition 4.3 (Outer Normal Form)
Let N be a network. Then there exists a network $\mathrm{nf_o}(N)$ on the form

$$\mathrm{nf_o}(N) \overset{\text{def}}{=} (\nu\tilde{x})(\nu\tilde{a})(\sigma \mid (\nu\tilde{b})\prod_{i=1}^{k} l_i[P_i]),$$

such that $N \equiv \mathrm{nf_o}(N)$, and

a) the locations l_i are pairwise different,

b) $\tilde{a} \subseteq \mathsf{n}(\sigma)$, $\tilde{b} \cap \mathsf{n}(\sigma) = \emptyset$, $\tilde{x} \subseteq \mathrm{dom}(\sigma)$, $\forall 1 \leq i \leq k[\mathsf{bv}(P_i) = \emptyset]$,

c) $\forall b \in \tilde{b} \exists 1 \leq i < j \leq k[b \in \mathsf{fn}(P_i) \cap \mathsf{fn}(P_j)]$

hold.

Proof
By induction on the structure of N.

Basis: Three cases to consider.

$N = \mathbf{0}$**:** Set $|\tilde{a}| = |\tilde{b}| = k = 0$ and $\mathrm{dom}(\sigma) = \emptyset$. Then $\mathrm{nf_o}(N) \equiv \mathbf{0} = N$.

$N = \{^T/_x\}$**:** Set $|\tilde{a}| = |\tilde{b}| = k = 0$ and $\sigma = \{^T/_x\}$. Then $N \equiv \mathrm{nf_o}(N)$.

$N = l[A]$**:** We have from Proposition 3.2 that $A \equiv (\nu\tilde{c})(\sigma \mid P)$. Factor $\tilde{c}$ out of l by use of (new-C). Apply (split) and (exit) to factor σ out of l. We get $N \equiv (\nu\tilde{c})(\sigma \mid l[P])$. For all $c \in \tilde{c}$ s.t. $c \notin \mathsf{n}(\sigma)$, apply (new-C), (new-par), and (new$_{\text{exp}}$) to factor c into l. The resulting network has the form $(\nu\tilde{c}')(\sigma \mid l[(\nu\tilde{c}'')P]) \equiv N$, where $\tilde{c} = \tilde{c}' \cup \tilde{c}''$. By setting $\tilde{a} = \tilde{c}'$ and $k = 1$, we get $N \equiv \mathrm{nf_o}(N)$.

Step: Assume N', where $\mathrm{nf_o}(N') = (\nu\tilde{x}')(\nu\tilde{a}')(\sigma' \mid (\nu\tilde{b}')\prod_{i=1}^{k'} l_i'[P_i'])$, satisfies the proposition. Three cases to consider.

$N = (\nu y)N'$**:** Safe to assume that there is an active substitution $\{^T/_y\}$ on y in σ', as $(\nu y)N'$ is only valid in that case[2]. $(\nu y)N'$ then satisfies conditions a, b, and c, is on the proposed form, and since $N = (\nu y)N'$, N satisfies the proposition.

$N = (\nu c)N'$**:** We have that $N \equiv (\nu c)\,\mathrm{nf_o}(N')$. If $c \in \mathsf{n}(\sigma')$, then applying (new-C) to factor c in amongst $\tilde{a}'$ yields a network congruent with $(\nu c)\,\mathrm{nf_o}(N')$ which is on the proposed form and satisfies conditions a, b, and c. Thus N satisfies the proposition. If $\exists i[c \in \mathsf{fn}(P_i)]$, then apply (new-C) and (new-par) to get $N \equiv (\nu\tilde{a}')(\sigma' \mid (\nu(\tilde{b}' \cup \{c\}))\prod_{i=1}^{k'} l_i'[P_i']) =: N''$. If $\exists j[j \neq i \wedge c \in \mathsf{fn}(P_j)]$, then N'' satisfies conditions a, b, and c, and since $N \equiv N''$ and N'' is on the proposed form, we then get that N satisfies the proposition. If $\nexists j[j \neq i \wedge c \in \mathsf{fn}(P_j)]$, then apply (new-C), (new-par), and (new$_{\text{exp}}$) to factor c into l_i'. The result will be congruent with N, be on the proposed form and satisfy conditions a, b, and c. Thus N satisfies the proposition. Lastly, if $\nexists i[c \in \mathsf{fn}(P_i)]$, then $(\nu c)N' \equiv N'$, and thereby, N satisfies the proposition.

$N = N' \mid N''$**:** We can assume N'' satisfies the proposition, as we have proven so for any N which contains no parallel composition. Let $\mathrm{nf_o}(N'') = (\nu\tilde{x}'')(\nu\tilde{a}'')(\sigma'' \mid (\nu\tilde{b}'')\prod_{i=1}^{k''} l_i''[P_i''])$. We have that $N \equiv \mathrm{nf_o}(N') \mid \mathrm{nf_o}(N'') =: N'''$. For all $c \in (\tilde{a}'' \cup \tilde{b}'') \cap \mathsf{n}(N')$, α-convert c in N'' to a name $c' \notin \mathsf{n}(N') \cup \mathsf{n}(N'')$. For all $y \in \tilde{x}'' \cap \mathsf{v}(N')$, α-convert y in N'' to a variable $y' \notin \mathsf{v}(N') \cup \mathsf{v}(N'')$. We can now assume $\mathrm{dom}(\sigma') \cap \mathrm{dom}(\sigma'') = \emptyset$[3]. Next, apply (new-par) on $\tilde{a}''$, $\tilde{a}'$, $\tilde{x}''$, and $\tilde{x}'$ in this order on network N'''. Let $\tilde{a} = \tilde{a}' \cup \tilde{a}''$ and $\tilde{x} = \tilde{x}' \cup \tilde{x}''$. This yields $(\nu\tilde{x})(\nu\tilde{a})((\sigma' \mid (\nu\tilde{b}')\prod_{i=1}^{k'} l_i'[P_i']) \mid (\sigma'' \mid (\nu\tilde{b}'')\prod_{i=1}^{k''} l_i''[P_i''])) \equiv N'''$. We get $(\nu\tilde{x})(\nu\tilde{a})(\sigma \mid (\nu\tilde{b}')\prod_{i=1}^{k'} l_i'[P_i'] \mid (\nu\tilde{b}'')\prod_{i=1}^{k''} l_i''[P_i''])) \equiv N'''$ by using (par-C), (par-A), and (par-$\mathbf{0}$) and letting $\sigma = \sigma' \mid \sigma''$. Next, apply (new-par) on $\tilde{b}''$ followed by $\tilde{b}'$ and let $\tilde{b} = \tilde{b}' \cup \tilde{b}''$ to obtain $(\nu\tilde{x})(\nu\tilde{a})(\sigma \mid (\nu\tilde{b})(\prod_{i=1}^{k'} l_i'[P_i'] \mid$

[2]Violating our assumptions in Section ?? otherwise.
[3]Otherwise $N' \mid N''$ would violate our assumptions in Section ??

$\prod_{i=1}^{k''} l_i''[P_i'])) \equiv N'''$. We get $(\nu\tilde{x})(\nu\tilde{a})(\sigma \mid (\nu\tilde{b})(\prod_{i=1}^{k'+k''} l_i'''[P_i''']))\equiv N'''$ by combining the products, where $\tilde{l}''' = \tilde{l}' \cdot \tilde{l}''$ (concatenated sequence), $P_i''' = P_i'$ for $1 \leq i \leq k'$ and $P_i''' = P_i''$ for $k'+1 \leq i \leq k''$. Finally, let k be the number of distinct l_i'''. Apply (split) $k'+k''-k$ times to obtain $(\nu\tilde{x})(\nu\tilde{a})(\sigma \mid (\nu\tilde{b})(\prod_{i=1}^{k} l_i[P_i])) \equiv N'''$ where all l_i are distinct and $P_i = \prod_{j\in\{j' \mid l_{j'}'''=l_i\}} P_j'''$. N''' is congruent to a network on the proposed form satisfying conditions a, b, and c. Since $N''' \equiv N$, we conclude that N satisfies the proposition. ■

Proposition 4.4 (Inner Normal Form)
Let N be a network. Then there exists a network $\mathrm{nf_i}(N)$ on the form

$$\mathrm{nf_i}(N) \overset{\text{def}}{=} (\nu\tilde{a})(\sigma \mid (\nu\tilde{b})\prod_{i=1}^{k} l_i[(\nu\tilde{c}_i)(\nu\tilde{x}_i)(\sigma_i \mid P_i)]),$$

such that $N \equiv \mathrm{nf_i}(N)$ and

a) the locations l_i are pairwise different,

b) $\tilde{a} \subseteq \sigma$, $\tilde{c}_i \subseteq \mathsf{n}(\sigma_i)$, $\tilde{x}_i = \mathrm{dom}(\sigma_i)$, $\forall 1 \leq i \leq k[\mathsf{bv}(P_i) = \emptyset]$,

c) $\forall b \in \tilde{b} \exists 1 \leq i < j \leq k[b \in (\mathsf{fn}(P_j) \cup \mathsf{n}(\sigma_j))\backslash\tilde{c}_j \wedge b \in (\mathsf{fn}(P_i) \cup \mathsf{n}(\sigma_i))\backslash\tilde{c}_i]$

hold.

Proof
Let $\mathrm{nf_o}(N) = (\nu\tilde{x'})(\nu\tilde{a'})(\sigma' \mid (\nu\tilde{b'})\prod_{i=1}^{k} l_i[P_i])$. We have by Proposition 4.3 that $N \equiv \mathrm{nf_o}(N)$. For each $\{T'/x'\}$ in σ', apply (new-par) s.t. the restriction of $\tilde{b}'$ also encompasses $\{T'/x'\}$ and apply (par-C) and (subst) to substitute $\{T'/x'\}$ into each location. Apply (new-par), (alias), (par-$\mathbf{0}$) and (par-C) on each $\{T'/x'\}$ in σ' where $x' \in \tilde{x}'$ to remove them from the network. Apply (new-par) on the remaining $\{T'/x'\}$ in σ' s.t. the restriction of $\tilde{b}'$ no longer encompasses $\{T'/x'\}$. This yields $(\nu\tilde{a'})(\sigma'' \mid (\nu\tilde{b'})\prod_{i=1}^{k} l_i[P_i\sigma']) \equiv N$, where $\mathrm{dom}(\sigma'') \subseteq \mathrm{dom}(\sigma')$. For each location l_i, apply (par-C), (new-par), (split), (alias$_{\text{loc}}$), (nil) and (par-$\mathbf{0}$) on each $\{T'/x'\}$ in σ' within l_i where $x' \notin \mathsf{v}(P_i)$ to $\{T'/x'\}$ from l_i. For each $a' \in \tilde{a}'$ where a' appears free in only one location, apply (new-C), (new-par) and (new$_{\text{exp}}$) to move a' into the location it solely appears in. This yields $(\nu\tilde{a})(\sigma'' \mid (\nu\tilde{b'})\prod_{i=1}^{k} l_i[(\nu\tilde{c}_i)(P_i\sigma_i''')]) \equiv (\nu\tilde{a})(\sigma'' \mid (\nu\tilde{b'})\prod_{i=1}^{k} l_i[(\nu\tilde{c}_i)(\nu\tilde{x}_i)(\sigma_i''' \mid P_i)])$, which is on the proposed form, satisfies conditions a, b, and c, and is congruent with N. Thus N satisfies the proposition. Since N was arbitrary, the proposition holds for any network. ■

4.6 Frames and Static Equivalence

At last we express the concepts of a frame and indistinguishability thereof in DAπ_β. We start with frames. While the definition of a frame in Aπ is indeed valid within locations in DAπ_β, we will need to extend the definition to make frames account for the network abstractions in DAπ_β. We extend the definition of a frame to a network in the natural way, as illustrated in the following grammar.

$$\begin{aligned}\varphi_N ::=\ & (\nu a)\varphi_N \\ \mid\ & l[\varphi] \mid \varphi_N \\ \mid\ & \{^T/_x\} \mid \varphi_N \\ \mid\ & \mathbf{0}\end{aligned}$$

To distinguish these two types of frames, we shall refer to φ_N as a *network frame*, and φ as a *location frame*. When it is either irrelevant or obvious from the context whether the frame is a network frame or a location frame, we simply refer to it as a *frame*.

As before, we *map a network N to its frame* $\varphi_N(N)$, by substituting all active substitutions in N into each location in contact with it, eliminating bound active substitutions not affecting a location by use of the Aπ structural equivalence rules (new-par), (par-C), (alias) and (par-**0**) at each location, and replacing all plain processes occuring in N with **0**. Again, as in Aπ, this mapping becomes straightforward when N is on inner normal form. Let $\mathrm{nf_i}(N) = (\nu\tilde{a})(\sigma \mid (\nu\tilde{b})\prod_{i=1}^{k} l_i[(\nu\tilde{c}_i)(\nu\tilde{x}_i)(\sigma_i \mid P_i)])$. Then

$$\varphi_N(\mathrm{nf_i}(N)) = (\nu\tilde{a})\sigma,$$

which is structurally equivalent with $\varphi_N(N)$, which is why we from now on write $\varphi_N(N)$ as a shorthand for $\varphi_N(\mathrm{nf_i}(N))$.

As locality has been removed in frames of networks on inner normal form, all the information exposed by the frame is exposed to the same environment. We can thus apply Definition 3.9 on frames of networks on inner normal form to reason about environment knowledge. This gives the following definition of static equivalence for networks.

Definition 4.10 (Static Equivalence)
Two networks N and M are static equivalent, written $N \approx_s M$, if

$$\varphi_N(N) \approx_s \varphi_N(M)$$

holds. □

Recall that $\varphi_N(N)$ and $\varphi_N(M)$ are Aπ frames. This is therefore merely an adoption of static equivalence to networks.

Thus, we can now directly adopt the definitions in Section 3.1.3of the following concepts, from Aπ, to DAπ_β:

- Deduction in a frame,
- Syntactic, and strong, secrecy in the passive case,
- Syntactic secrecy, in the active case.

4.7 Summary

Besides replacing the general data language with a different formalism, by making three small changes to the Applied π calculus, we obtain the calculus Distributed Applied π Calculus with Broadcast which is surprisingly expressive. The changes made are

The network layer, which essentially *is* one large Applied π-style evaluation context $C_N[A_1, \ldots, A_n]$, with several holes, where $C_N[\mathbf{0}, \ldots, \mathbf{0}]$ is incapable of performing any actions. Each A_i is then assigned a location name l_i, which is either open or closed by the *i*th hole in C_N.

Connectivity graphs, which purpose is, like name restrictions, to constrain synchronisation over named channels.

Broadcast semantics, perhaps being the most significant change, replaces the point-to-point semantics with a broadcasting one.

While one might think that these three concepts would significantly increase the conceptual complexity of DAπ_β, we deem this not to be the case. For one, the graph-, location-, and broadcast concepts are fairly basic, and well understood, and commonly seen in the context of distributed computing. Second, the semantics are very much alike the semantics of Aπ. Yet, we can easily express different overlaying broadcasting networks in the same model, untrusted nodes, and a changing network topology.

To compared to the discussion in the summary of the last chapter, we can, like in CBS$^\sharp$, examine the behaviour of a process specification in any network topology, without changing the syntax of the process specification. As opposed to CMAN, we have parallel composition on the internal process layer, and by separating the topology from the process syntax and only sending identifiers, obtain a much briefer semantics specification. We can even split a location into several parts, one part capable of being under different name and variable restrictions than another.

The next logical step is to provide means of automatically verifying security properties of process specifications in DAπ_β, which is the purpose of the next two chapters. We shall return to DAπ_β in Chapter 7, with new, interesting results.

CHAPTER 5

Enriching ABπ and H, and Revelation Semantics

Recall the Horn clause generation for ABπ in Section 3.2.3, which Abadi and Blanchet use for their static analysis of security properties of protocols modelled in their calculus. In this chapter and the next, we prove a particular relationship between the Horn clauses generated from the initial process P_0, and transition sequences in P_0. Namely that performing a particular transition sequence corresponds to instantiating particular Horn clauses. More specifically, for each message-passing transition, there occurs an output, either in P_0, or in the environment. That output then instantiates a conclusion of some clause, which premises are instantiated by prior transitions.

Before we present the proof, there are two issues that need to be addressed. First, we need to ensure that the names and variables occurring in the Horn clauses generated from P_0 remain syntactically consistent with the names and variables of P_0 during reduction. This includes ensuring that P_0 records sessions properly during reduction, and in a manner comparable to the session variables in H. Also, we will need the state which the Horn clause generator was in when it created each clause. Doing this requires minor changes in the syntax, semantics, and Horn clause generation of ABπ. The modified calculus will be presented in Section 5.1.

Second, we need a way of explicitly specifying the output occurring in a message-passing reduction in P_0. Furthermore, we need to ensure that the knowledge of the environment during reduction is consistent with what the applied threat model specifies. While the former can be accomplished by using $\xrightarrow{\alpha}$ for synchronisations with the environment, and investigating the local knowledge of P_0 before and after each internal reduction step, the latter is not satisfied by the semantics of ABπ. In [Bla02], Blanchet explains that the environment implicitly receives messages sent on public channels, like in the Dolev-Yao threat model. This is also made apparent in Algorithm 3.1, line 12. As explained in the beginning of Chapter 4, since the source and destination of a message communicated over a given channel in point-to-point calculi like ABπ is explicit, with 1 receiver per sender, and since information can only escape to the environment trough message-passing, information does not flow anywhere implicitly during internal reduction.

To address this, we define a new operational semantics for the modified ABπ, which we shall call the *revelation semantics*, in Secction 5.2.

5.1 Extending ABπ with Active Substitutions and Sessions

In this section we present the syntactic changes we make on the initial process enabling us to prove the relation between transition sequences in the initial process, and the Horn clauses generated from it. Before we begin, we wish to present one of the main cultprits in making the Horn clauses generated from the initial process differ syntactically from the initial process, and how it does so: the “let” expression.

Remark 5.1
In Table 3.11, in the encoding rule

$$\left[\!\!\left[\begin{array}{l}\textbf{let } x = \mathsf{g}(T_1,\ldots,T_n)\\ \textbf{in } P\\ \textbf{else } Q\end{array}\right]\!\!\right]_{\rho h} = \bigcup \left\{ [\![P]\!]_{((\sigma\rho)[x\mapsto\sigma' p'])(\sigma h)} \;\middle|\; \begin{array}{l}\mathsf{g}(p'_1,\ldots,p'_n) > p' \in \mathrm{def}(\mathsf{g}),\\ \text{and } (\sigma,\sigma') \text{ is the most gene-}\\ \text{ral pair of substitutions}\\ \text{st. } \sigma\rho(T_i) = \sigma' p'_i; 1 \le i \le n\end{array}\right\},$$

the substitutions (σ, σ') express a unification problem. The equations in question are $\{\sigma\rho(T_i) = p'_i \mid 1 \le i \le n\}$, and the variables we unify with regards to are the variables occurring in the p'_i in the term rewrite rule $\mathsf{g}(p'_1,\ldots,p'_n) > p'$. The result of the unification is the substitution σ', which has $\mathrm{dom}(\sigma') = \bigcup_{i=1}^{n} \mathsf{v}(p'_i)$ and which ranges over subterms of the $\sigma\rho(T_i)$ terms. $\square$

In the unification in Remark 5.1, the purpose of σ is twofold, as we will illustrate in the following two examples.

Example 5.2
σ ensures that the process variables in the T_i differ syntactically from the rewrite variables of the p'_i. This is relevant in, for instance, the process

$$P \overset{\text{def}}{=} a(x).a(y).\overline{a}\langle y\rangle.\textbf{let } z = \mathsf{dec}(\mathsf{enc}(\mathsf{pair}(x,a),y),y) \textbf{ in } \overline{a}\langle z\rangle \textbf{ else } \mathbf{0},$$

when we consider the rewrite rule $\mathsf{dec}(\mathsf{enc}(x,y),y) > x$. With $\sigma = \emptyset$, the unification problem becomes $\{\mathsf{enc}(\mathsf{pair}(x,a),y) = \mathsf{enc}(x,y), y = y\}$ (which variables are x and y), which reduces to $\{\mathsf{pair}(x,a) = x, y = y\}$ and further to $\{x = \mathsf{pair}(x,a), y = y\}$. By Algorithm 2.1, the unification fails, as x equals a term where x itself appears as a subterm. The error here is that the x on the left side of the equality $x = \mathsf{pair}(x,a)$ is, semantically, not the same x as the x on the right side of the equality (the x on the right is a 0-ary function constant, and the x on the left is a variable, in the unification problem). However, if we introduce the substitution $\sigma = \{x'/x, y'/y\}$, we get the unification problem

$\{\mathsf{enc}(\mathsf{pair}(x', a), y') = \mathsf{enc}(x, y), y' = y\}$ (again with unification variables x and y), which eventually reduces to $\{x = \mathsf{pair}(x', a), y = y'\}$, which is a successful unification. Thus,

$$[\![P]\!]_{\rho_{\text{init}}\emptyset} = \left\{ \begin{array}{lcl} \mathsf{msg}(a, x) \wedge \mathsf{msg}(a, y) & \Longrightarrow & \mathsf{msg}(a, y), \\ \mathsf{msg}(a, x') \wedge \mathsf{msg}(a, y') & \Longrightarrow & \mathsf{msg}(a, \mathsf{pair}(x', a)) \end{array} \right\}. \qquad \square$$

Notice in Example 5.2 that, although the required inputs for $\overline{a}\langle y\rangle$ and $\overline{a}\langle z\rangle$ to occur in P are the same, the variables in the premises in $[\![P]\!]_{\rho_{\text{init}}\emptyset}$ are not the same syntactically. This is not a problem for the static analysis of the Horn clauses, as $[\![P]\!]_{\rho_{\text{init}}\emptyset}$ abstracts from the syntax of P. Thus, these variables are there only to express that the clause is valid for any instantiation of x and y (and x' and y'). We wish to avoid this abstraction for the purpose of proving the soundness result, as we need to relate the variables in the Horn clauses to the actual variables in the original process.

Example 5.3
When necessary, σ imposes a condition on the instantiation of the variables appearing within the destructor context in the let expression. This is required in processes such as

$$P' \stackrel{\text{def}}{=} a(x_1).a(x_2).\overline{a}\langle x_2\rangle.\textbf{let } z = \mathsf{dec}(x_1, x_2) \textbf{ in } \overline{a}\langle z\rangle \textbf{ else } \mathbf{0},$$

when we consider the rewrite rule $\mathsf{dec}(\mathsf{enc}(z_1, z_2), z_2) > z_1$. With $\sigma = \emptyset$, the unification problem becomes $\{x_1 = \mathsf{enc}(z_1, z_2), x_2 = z_2\}$ (which variables are z_1 and z_2), which, by Algorithm 2.1, fails, as x_1 is a constant (a 0-ary function) in the unification problem. However, by imposing a limit on the structure of x_1, say, $\sigma = \{^{\mathsf{enc}(y_1, x_2)}/_{x_1}\}$, where y_1 does not appear in $\mathsf{v}(P')$ or in rewrite rules, we get the unification problem $\{\mathsf{enc}(y_1, x_2) = \mathsf{enc}(z_1, z_2), x_2 = z_2\}$ (which variables are z_1 and z_2), which reduces to $\{y_1 = z_1, x_2 = z_2, x_2 = z_2\}$, which eventually reduces to $\{z_1 = y_1, z_2 = x_2\}$. Thus,

$$[\![P']\!]_{\rho_{\text{init}}\emptyset} = \left\{ \begin{array}{lcl} \mathsf{msg}(a, x_1) \wedge \mathsf{msg}(a, x_2) & \Longrightarrow & \mathsf{msg}(a, x_2), \\ \mathsf{msg}(a, \mathsf{enc}(y_1, x_2)) \wedge \mathsf{msg}(a, x_2) & \Longrightarrow & \mathsf{msg}(a, y_1) \end{array} \right\}. \qquad \square$$

In Example 5.3, we have limited the possible instantiations of x_1, and by doing so, introduced a new variable y_1. Again, in the Horn clauses, this variable simply expresses that the latter Horn clause is true for any instantiation of y_1 (and x_2). When relating an execution of P' to its Horn clauses, we need to ensure that y_1 is instantiated properly.

5.1.1 Syntax and Semantics of ABπ'

We now present the syntax and semantics of the Extended Abadi/Blanchet calculus, or ABπ' for short. ABπ' extends ABπ with an extended process layer, akin to Aπ (that is, floating active substitutions), and with a session-enriched replication directly inspired by [Bla02].

Syntax

We let $\mathcal{N}$ be the set of names ranged over by $a, \ldots, c, o, \ldots, t$, and $\mathcal{V}_{\mathscr{A}}$ be the set of (process) variables ranged over by $x, \ldots, z$. Furthermore, let $\mathcal{U} = \mathcal{N} \cup \mathcal{V}_{\mathscr{A}}$ be ranged over by $u, \ldots, w$. Let $\mathcal{F}$ be the set of constructors, ranged over by f, and $\mathcal{G}$ be the set of destructors, ranged over by g. Let n range over $\mathbb{N}$. The syntax of ABπ' is then given by the grammar specification in Table 5.1.

$$
\begin{array}{rcl}
P & ::= & u(x).P \\
& | & \overline{u}\langle T\rangle.P \\
& | & !^{i\geq n}P \\
& | & (\nu a)P \\
& | & P \mid P \\
& | & \textbf{let } x = \mathsf{g}(T_1,\ldots,T_n) \textbf{ in } P \textbf{ else } P \\
& | & \mathbf{0}
\end{array}
\qquad
\begin{array}{rcl}
T & ::= & u \\
& | & \mathsf{f}(T,\ldots,T) \\
\\
A & ::= & P \\
& | & A \mid A \\
& | & (\nu u)A \\
& | & \{{}^{T}/_{x}\}
\end{array}
$$

Table 5.1: Syntax of ABπ'.

Like in Aπ and ABπ, the syntactic category P represents primitive processes, which are ranged over by $P, \ldots, R \in \mathscr{P}$, the syntactic category T represents term, which are ranged over by $T, \ldots, V \in \mathcal{T}$, and the syntactic category A represents extended processes, ranged over by $A, \ldots, B \in \mathscr{A}$.

The only new element in the syntax of ABπ' compared to the syntax of Aπ and ABπ is the treatment of replication. The idea is that to each replication, we associate a *session variable*, i in the grammar specification, which is chosen from the set $\mathcal{V}_S$ of session variables disjoint from $\mathcal{V}_{\mathscr{A}}$, and a natural number, n in the grammar specification, which we increment each time the replication replicates. When the replication replicates, the new parallel component instantiates i with the current value of n and binds i to its scope, and the replication increments n, such that the next time the replication replicates, that parallel component will instantiate its i to $n+1$. This will be clear in the semantics definitions below.

Notice that it is trivial to encode a ABπ process P to a ABπ' process; simply replace each replication ! in A with $!^{i\geq n}$ for some session variable i (for instance, i, each time) and integer n (for instance, 0). The result A is an ABπ' process on normal form (as defined in Aπ).

Rewrite System

In much the same way that Aπ can be instantiated with an arbitrary equational theory, ABπ' can be instantiated by an arbitrary rewrite system $\mathcal{R}$. We can assume it is instantiated in the same manner as it is in Abadi and Blanchet's work [AB02, Bla02].

Note, however, that we assume that the variables applied in the definition of the term rewrite rules in $\mathcal{R}$ are all pairwise syntactically different, and chosen from the set $\mathcal{V}_{\mathcal{R}}$ of variables disjoint from $\mathcal{V}_{\mathscr{A}} \cup \mathcal{V}_S$. This will take care of the problem explained in Example 5.2.

Semantics

We give the *operational semantics* of ABπ' in the form of a structural equivalence relation and an internal reduction relation, as is tradition.

Definition 5.1 (Structural Equivalence in ABπ')
Structural equivalence, denoted by $\equiv$, is the smallest equivalence relation over processes that is closed by α-conversions, by application of evaluation contexts, and which satisfies the axioms in Table 5.2 $\square$

(par-**0**)	$A \equiv A \mid \mathbf{0}$
(par-A)	$A \mid (B \mid C) \equiv (A \mid B) \mid C$
(par-C)	$A \mid B \equiv B \mid A$
(repl-S)	$!^{i\geq n}P \equiv P\{^n/_i\} \mid !^{i\geq n+1}P$
(new-**0**)	$(\nu n)\mathbf{0} \equiv \mathbf{0}$
(new-C)	$(\nu u)(\nu v)A \equiv (\nu v)(\nu u)A$
(new-par)	$A \mid (\nu u)B \equiv (\nu u)(A \mid B)$, if $u \notin \mathsf{fv}(A) \cup \mathsf{fn}(A)$
(alias)	$(\nu x)\{^T/_x\} \equiv \mathbf{0}$
(subst)	$\{^T/_x\} \mid A \equiv \{^T/_x\} \mid A\{^T/_x\}$

Table 5.2: Structural equivalence in ABπ'.

Definition 5.2 (Internal Reduction in ABπ')
Internal reduction, denoted by $\to$, is the smallest relation on extended processes closed by $\equiv$ and application of evaluation contexts, and satisfying the axioms in Table 5.3. $\square$

(destr$_1$)	**let** $x = \mathsf{g}(T_1, \dots, T_n)$ **in** P **else** $Q \to (\nu x)(\{^T/_x\} \mid P)$, if $\mathsf{g}(T_1, \dots, T_n) > T \in \mathrm{def}(\mathsf{g})$
(destr$_2$)	**let** $x = \mathsf{g}(T_1, \dots, T_n)$ **in** P **else** $Q \to Q$, if $\nexists T[\mathsf{g}(T_1, \dots, T_n) > T \in \mathrm{def}(\mathsf{g})]$
(sync)	$\overline{a}\langle T\rangle.Q \mid a(x).P \to Q \mid (\nu x)(\{^T/_x\} \mid P)$

Table 5.3: Internal Reduction relation in ABπ'.

The rule (sync) may look surprising after seeing the definition of $\to$ from Aπ. However, as the structural equivalence $(\nu x)(\{^T/_x\} \mid P) \equiv P\{^T/_x\}$ also holds for ABπ', we can, using the (sync) rule, derive

$$\begin{aligned}\{{}^{T}/_{x}\} \mid \overline{a}\langle x\rangle.P \mid a(x).Q &\equiv \{{}^{T}/_{x}\} \mid \overline{a}\langle T\rangle.P \mid a(x).Q\\ &\rightarrow \{{}^{T}/_{x}\} \mid P \mid (\nu x)(\{{}^{T}/_{x}\} \mid Q)\\ &\equiv \{{}^{T}/_{x}\} \mid P \mid Q\{{}^{T}/_{x}\}\\ &\equiv \{{}^{T}/_{x}\} \mid P \mid Q,\end{aligned}$$

just like the reduction $\overline{a}\langle T\rangle.P \mid a(x).Q(\equiv^* \cdot \rightarrow \cdot \equiv^*)P \mid Q\{{}^{T}/_{x}\}$ could be derived in Aπ using the (comm) rule there.

We also inherit a labelled reduction relation for ABπ' from Aπ.

Definition 5.3 (Labelled Operational Semantics in ABπ')
The *Labelled Operational Semantics* of ABπ' extend the rules of $\equiv$ and $\rightarrow$ with a *labelled reduction* relation, denoted by $\xrightarrow{\alpha}$, which is defined by the rules in Table 5.4. □

(in) $$\frac{}{a(x).P \xrightarrow{a(M)} (\nu x)(\{{}^{M}/_{x}\} \mid P)}$$

(out-atom) $$\frac{}{\overline{a}\langle u\rangle \xrightarrow{\overline{a}\langle u\rangle} P}$$

(open-atom) $$\frac{A \xrightarrow{\overline{a}\langle u\rangle} A', \quad u \neq a}{(\nu u)A \xrightarrow{(\nu u)\overline{a}\langle u\rangle} A'}$$

(scope) $$\frac{A \xrightarrow{\alpha} A', \quad u \text{ does not occur in } \alpha}{(\nu u)A \xrightarrow{(\nu u)\overline{a}\langle u\rangle} A'}$$

(par) $$\frac{A \xrightarrow{\alpha} A', \quad \mathsf{bv}(\alpha) \cap \mathsf{fv}(B) = \mathsf{bn}(\alpha) \cap \mathsf{fn}(B) = \emptyset}{A \mid B \xrightarrow{\alpha} A' \mid B}$$

Table 5.4: Labeled Reduction in ABπ'.

We then define an *extended labelled reduction* relation as

$$(\xrightarrow{\alpha}) \overset{\text{def}}{=} (\equiv \cdot \xrightarrow{\alpha} \cdot \equiv),$$

a *weak extended labelled reduction* relation as

$$(\overset{\alpha}{\Longrightarrow}) \overset{\text{def}}{=} (\rightarrow^* \cdot \xrightarrow{\alpha} \cdot \rightarrow^*),$$

where $\rightarrow^*$ denotes the reflexive, transitive closure of $\rightarrow$, and finally, we let $\xrightarrow{\tilde{\alpha}}$ denote $\xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_{|\tilde{\alpha}|}}$, and similarly, $\overset{\tilde{\alpha}}{\Longrightarrow}$ denote $\overset{\alpha_1}{\Longrightarrow} \cdots \overset{\alpha_{|\tilde{\alpha}|}}{\Longrightarrow}$.

Notice that $\xrightarrow{\alpha}$ is *not* closed by evaluation contexts, since the context may bind a name which prohibits a labelled output[1]. Note also that we have in a sense "moved"

[1] This is in fact also true in Aπ.

replication from being a reduction rule, to being a structural equivalence rule. This is essentially a matter of how to interpret replication. As a reduction rule, replication is considered a "process that replicates", while as a structural equivalence rule, replication is considered an "infinite parallel composition of said process". The latter is the way replication is normally treated in point-to-point process calculi, like Aπ, and as $\rightarrow$ is closed by structural equivalence, and as structural equivalence can be applied between $\xrightarrow{\alpha}$-reductions, there will not be any process A' which a given process A can reduce to with replication as a reduction rule, but cannot with replication as a structural equivalence rule, and vice versa (although the number of *reduction* steps to reach A' might differ).

Frames and Normal Forms

We can derive the same normal form in ABπ' as the one we have in Aπ. As such, the frame of a process in ABπ' is defined in the same manner as the frame of an extended process in Aπ.

Furthermore, we define the *process substitutions* of a process A, $\varphi_{\mathrm{k}}(A)$, as the parallel composition of *every active substitution* in A. For instance, for the process

$$A \overset{\text{def}}{=} (\nu\tilde{a})(\sigma \mid (\nu\tilde{x})(\sigma' \mid P)),$$

we have

$$\varphi_{\mathrm{k}}(A) \overset{\text{def}}{=} \sigma \mid \sigma'.$$

We use this frame as a means of seeing what term was input after an input operation, which is possible provided you compare $\varphi_{\mathrm{k}}(A)$ and $\varphi_{\mathrm{k}}(A')$, where A' is the immediate process resulting from a single reduction in A.

5.1.2 Syntactic Consistency and Canonicalisation

We now present a property of extended processes which we will require to establish the relation between transitions in an extended processes, and the Horn clauses generated from it. We call this property *syntactic consistency*, and briefly, it expresses that all bound names and variables in an extended process A are pairwise different from each other, as well as from names and variables that are free in A.

Definition 5.4 (Syntactic Consistency)
An extended process A is said to be *syntactically consistent* if it holds that

$$\forall u \in \mathsf{u}(A)\ \nexists C, B[A = C[(\nu u)B] \land u \in \mathsf{u}(C[\mathbf{0}])]. \qquad \square$$

Rewriting a process to syntactic consistent form is easy; simply apply α-conversions selectively on bound names and variables until all bound names and variables differ syntactically from free ones, and each other. The encoding in Table 5.5 does exactly this.

The encoding has a side effect in the function get, which we define in Algorithm 5.1. The purpose of get is to supply $[\![\cdot]\!]_{\mathsf{f}}^{c}$ with syntactically distinct names and variables, which

$$\begin{aligned}
\llbracket \mathbf{0} \rrbracket^c_{\mathsf{f}} &= \mathbf{0}\\
\llbracket A \mid B \rrbracket^c_{\mathsf{f}} &= \llbracket A \rrbracket^c_{\mathsf{f}} \mid \llbracket B \rrbracket^c_{\mathsf{f}}\\
\llbracket (\nu u)A \rrbracket^c_{\mathsf{f}} &= \begin{cases} (\nu u)\llbracket A \rrbracket^c_{\mathsf{f}[u\mapsto u]}, & \text{if } u \notin \mathrm{dom}(\mathsf{f})\\ (\nu v)\llbracket A \rrbracket^c_{\mathsf{f}[u\mapsto v]} & \text{otherwise, where } v = \mathsf{get}(u) \end{cases}\\
\llbracket \{T/x\} \rrbracket^c_{\mathsf{f}} &= \{\llbracket T \rrbracket^c_{\mathsf{f}}/\llbracket x \rrbracket^c_{\mathsf{f}}\}\\
\llbracket T(u).P \rrbracket^c_{\mathsf{f}} &= \begin{cases} \llbracket T \rrbracket^c_{\mathsf{f}}(u).\llbracket P \rrbracket^c_{\mathsf{f}[u\mapsto u]}, & \text{if } u \notin \mathrm{dom}(\mathsf{f})\\ \llbracket T \rrbracket^c_{\mathsf{f}}(v).\llbracket P \rrbracket^c_{\mathsf{f}[u\mapsto v]} & \text{otherwise, where } v = \mathsf{get}(u) \end{cases}\\
\llbracket \overline{T}\langle U \rangle.P \rrbracket^c_{\mathsf{f}} &= \overline{\llbracket T \rrbracket^c_{\mathsf{f}}}\langle \llbracket U \rrbracket^c_{\mathsf{f}} \rangle.\llbracket P \rrbracket^c_{\mathsf{f}}\\
\llbracket !^{i\geq n} P \rrbracket^c_{\mathsf{f}} &= \begin{cases} !^{i\geq n}\llbracket P \rrbracket^c_{\mathsf{f}[i\mapsto i]}, & \text{if } i \notin \mathrm{dom}(\mathsf{f})\\ !^{i'\geq n}\llbracket P \rrbracket^c_{\mathsf{f}[i\mapsto i']} & \text{otherwise, where } i' = \mathsf{get}(i) \end{cases}\\
\left\llbracket \begin{array}{l} \mathbf{let}\ z = \mathsf{g}(T_1,\dots,T_n)\\ \mathbf{in}\ P\\ \mathbf{else}\ Q \end{array} \right\rrbracket^c_{\mathsf{f}} &= \begin{cases} \left.\begin{array}{l} \mathbf{let}\ z = \mathsf{g}(\llbracket T_1 \rrbracket^c_{\mathsf{f}},\dots,\llbracket T_n \rrbracket^c_{\mathsf{f}})\\ \mathbf{in}\ \llbracket P \rrbracket^c_{\mathsf{f}[z\mapsto z]}\\ \mathbf{else}\ \llbracket Q \rrbracket^c_{\mathsf{f}[z\mapsto z]} \end{array}\right\}, & \text{if } z \notin \mathrm{dom}(\mathsf{f})\\ \left.\begin{array}{l} \mathbf{let}\ z' = \mathsf{g}(\llbracket T_1 \rrbracket^c_{\mathsf{f}},\dots,\llbracket T_n \rrbracket^c_{\mathsf{f}})\\ \mathbf{in}\ \llbracket P \rrbracket^c_{\mathsf{f}[z\mapsto z']}\\ \mathbf{else}\ \llbracket Q \rrbracket^c_{\mathsf{f}[z\mapsto z']} \end{array}\right\} & \begin{array}{l}\text{otherwise,}\\ \text{where } z' = \mathsf{get}(z)\end{array} \end{cases}\\
\llbracket u \rrbracket^c_{\mathsf{f}} &= \mathsf{f}(u)\\
\llbracket \mathsf{f}(T_1,\dots,T_n) \rrbracket^c_{\mathsf{f}'} &= \mathsf{f}(\llbracket T_1 \rrbracket^c_{\mathsf{f}'},\dots,\llbracket T_1 \rrbracket^c_{\mathsf{f}'})
\end{aligned}$$

Table 5.5: Encoding to Syntactically Consistent Form.

do not occur in the process being encoded, and which have not been requested earlier in the encoding process. That last part is the reason for the side effect, for to obtain that property, get relies on the sets $\mathcal{N}_{\text{fresh}}$, $\mathcal{V}_{S_{\text{fresh}}}$ and $\mathcal{V}_{\mathscr{A}_{\text{fresh}}}$, which consist of names, session variables, and process variables, respectively, which do not occur in the syntax specification of the original process, and updates these sets each time a fresh name or variable is requested.

Thus, given an extended process A, then by first assigning $\mathcal{N}_{\text{fresh}} := \mathcal{N}\setminus\mathsf{n}(A)$, $\mathcal{V}_{S_{\text{fresh}}} := \mathcal{V}_S \setminus (\mathsf{v}(A) \cap \mathcal{V}_S)$ and $\mathcal{V}_{\mathscr{A}_{\text{fresh}}} := \mathcal{V}_{\mathscr{A}} \setminus (\mathsf{v}(A) \cap \mathcal{V}_{\mathscr{A}})$, and then encoding A with $\llbracket\cdot\rrbracket^c_{\mathsf{f}_{\text{init}}}$, where f_{init} is a bijection on $\mathsf{fu}(A)$, we claim that we get a syntactically consistent process, $\llbracket A\rrbracket^c_{\mathsf{f}_{\text{init}}}$.We write A^c as a shorthand for $\llbracket A\rrbracket^c_{\mathsf{f}_{\text{init}}}$ when discussing both A and its syntactically consistent form.

As the sets $\mathcal{N}$, $\mathcal{V}_S$ and $\mathcal{V}_{\mathscr{A}}$ are infinite, and $\mathsf{n}(A)$ and $\mathsf{v}(A)$ are finite, the sets $\mathcal{N}_{\text{fresh}}$, $\mathcal{V}_{S_{\text{fresh}}}$ and $\mathcal{V}_{\mathscr{A}_{\text{fresh}}}$ will be infinite, and as such, the get function will always be capable of supplying the encoding with fresh names and variables. Lastly, we claim that applying $\llbracket\cdot\rrbracket^c_{\mathsf{f}_{\text{init}}}$ to a process A corresponds to applying a sequence of α-conversions to A.As such, we have that $A \equiv_\alpha A^c$, and, since we have from [MPW92] that $\equiv_\alpha$ is a strong bisimula-

Algorithm 5.1: Function used to get fresh names and variables, $\mathsf{get}(u)$.

```
1  Input: A name or a variable, $u$.
2  Result: A fresh name or variable.
3  Side-effect: Affect the sets $\mathcal{N}_{\mathsf{fresh}}$, $\mathcal{V}_{S_{\mathsf{fresh}}}$ and $\mathcal{V}_{\mathscr{A}_{\mathsf{fresh}}}$ denoting names, session
   variables, and process variables, respectively, not occurring in the syntax specification of
   the original process.
4  if $u \in \mathcal{N}$ then
5      $u' := a$ for some $a \in \mathcal{N}_{\text{fresh}}$
6      $\mathcal{N}_{\text{fresh}} := \mathcal{N}_{\text{fresh}} \setminus \{a\}$
7      return $u'$
8  else if $u \in \mathcal{V}_S$ then
9      $u' := i$ for some $i \in \mathcal{V}_{S_{\text{fresh}}}$
10     $\mathcal{V}_{S_{\text{fresh}}} := \mathcal{V}_{S_{\text{fresh}}} \setminus \{i\}$
11     return $u'$
12 else
13     $u' := v$ for some $v \in \mathcal{V}_{\mathscr{A}_{\text{fresh}}}$
14     $\mathcal{V}_{\mathscr{A}_{\text{fresh}}} := \mathcal{V}_{\mathscr{A}_{\text{fresh}}} \setminus \{v\}$
15     return $u'$
```

tion, then A and A^c are strongly bisimilar.We illustrate how $[\![\cdot]\!]^c_{\mathsf{f}}$ works in the following example.

Example 5.4
Consider the process

$$P \stackrel{\text{def}}{=} (\nu b)(a(x).\overline{a}\langle \mathsf{pair}(x,b)\rangle \mid a(x).\overline{a}\langle \mathsf{enc}(x,b)\rangle).$$

We have

$$\mathsf{H}_{\mathsf{prot}}(P) = [\![P]\!]_{\rho_{\text{init}}\emptyset} = \left\{ \begin{array}{lcl} \mathsf{msg}(a,x) & \Longrightarrow & \mathsf{msg}(a,\mathsf{pair}(x,b[x])) \\ \mathsf{msg}(a,x) & \Longrightarrow & \mathsf{msg}(a,\mathsf{enc}(x,b[x])) \end{array} \right\}.$$

The Horn clauses in $\mathsf{H}_{\mathsf{prot}}(P)$ (correctly) state that, after any message (which we call x) has been sent on channel a, it is possible that $\mathsf{pair}(x,b)$ and $\mathsf{enc}(x,b)$ may be sent on a.

Recall that we wish to make the Horn clauses in $\mathsf{H}_{\mathsf{prot}}(P)$ syntactically comparable to input/output prefixes in P. The choice of x as the input variable for both parallel components causes problems in that regard. Consider, for instance, the reduction

$$P \xrightarrow{a(\mathsf{data})} (\nu b)(a(x).\overline{a}\langle \mathsf{pair}(x,b)\rangle \mid ((\overline{a}\langle \mathsf{enc}(x,b)\rangle)\{^{\mathsf{data}}/_{x}\})) \stackrel{\text{def}}{=} P'.$$

While data was a message sent over a and stored in x, and it could have been the left-most parallel component that received the message and stored it in its x, the fact of the matter is that it was the right-most parallel component that received the message and stored it in its x. Thus, naïvely matching the $\xrightarrow{a(\mathsf{data})}$ input action with the premise in the first clause in $\mathsf{H}_{\mathsf{prot}}(P)$ and then concluding that a message $\mathsf{pair}(\mathsf{data}, b)$ can now be sent over a (without performing further input actions in P') would be fallacious.

Now, if we, before performing reductions in P, first run it through our syntactic consistency encoding, $[\![P]\!]^c_{\mathsf{f}_{\text{init}}}$, with f_{init} being a bijection on $\mathsf{fu}(P)$, and with the assignments $\mathcal{N}_{\text{fresh}} := \mathcal{N} \setminus \mathsf{n}(P)$, $\mathcal{V}_{S_{\text{fresh}}} := \mathcal{V}_S \setminus (\mathsf{v}(P) \cap \mathcal{V}_S)$ and $\mathcal{V}_{\mathscr{A}_{\text{fresh}}} := \mathcal{V}_{\mathscr{A}} \setminus (\mathsf{v}(P) \cap \mathcal{V}_{\mathscr{A}})$ to yield P^c,

$$P^c \overset{\text{def}}{=} [\![P]\!]^c_{\mathsf{f}_{\text{init}}} = (\nu a_1)(a(x_1).\overline{a}\langle \mathsf{pair}(x_1, a_1)\rangle \mid a(x_2).\overline{a}\langle \mathsf{enc}(x_2, a_1)\rangle),$$

we get the set of Horn clauses

$$\mathsf{H}_{\mathsf{prot}}(P^c) = [\![P^c]\!]_{\rho_{\text{init}}\emptyset} = \left\{ \begin{array}{lcl} \mathsf{msg}(a, x_1) & \Longrightarrow & \mathsf{msg}(a, \mathsf{pair}(x_1, a_1[x_1])) \\ \mathsf{msg}(a, x_2) & \Longrightarrow & \mathsf{msg}(a, \mathsf{enc}(x_2, a_1[x_1])) \end{array} \right\}.$$

Performing the same reduction,

$$P^c \xrightarrow{a(\mathsf{data})} (\nu a_1)(a(x_1).\overline{a}\langle \mathsf{pair}(x_1, a_1)\rangle \mid ((\overline{a}\langle \mathsf{enc}(x_2, a_1)\rangle)\{\mathsf{data}/x_2\})) = P^{c\prime},$$

we see that the input now only matches the premise of the second Horn clause, which correctly states that without performing more input actions, $P^{c\prime}$ can output $\mathsf{enc}(\mathsf{data}, a_1)$ on a. □

While the approach illustrated above takes care of transforming syntactically inconsistent processes into syntactically consistent ones, there is still an issue with α-conversion between reduction steps (which is allowed since $\equiv_\alpha \subseteq \equiv$).

Example 5.5
Consider again the process P^c defined in Example 5.4, and the Horn clauses $\mathsf{H}_{\mathsf{prot}}(P^c)$ generated from P^c. We have that

$$P^c \xrightarrow{a(\mathsf{data})} (\nu a_1)(a(x_1).\overline{a}\langle \mathsf{pair}(x_1, a_1)\rangle \mid ((\overline{a}\langle \mathsf{enc}(x_1, a_1)\rangle)\{\mathsf{data}/x_1\})) = Q$$

since

$$P^c \equiv_\alpha (\nu a_1)(a(x_1).\overline{a}\langle \mathsf{pair}(x_1, a_1)\rangle \mid a(x_1).\overline{a}\langle \mathsf{enc}(x_1, a_1)\rangle).$$

Now Q is syntactically inconsistent. We have received a message data and stored it in a variable named x_1, which is not the same variable as the x_1 in the leftmost parallel component. As such, concluding from the Horn clauses that the message $\mathsf{pair}(\mathsf{data}, a_1)$ may be sent on a would be fallacious. □

In Example 5.5, while x_2 and the x_1 it got α-converted into are conceptually the same, they differ syntactically, which is what causes problems when making the syntactic relation between P^c reduced, and $\mathsf{H}_{\mathsf{prot}}(P^c)$.

We are interested in maintaining this "conceptual equality" in the presence of α-conversion. There are two common ways of addressing this issue: de Bruijn indices [dB72, Pie02], and Canonicalisation.

Briefly, de Bruijn indexing, originally introduced in the λ-calculus by Nicolaas Govert de Bruijn [Pie02], involes identifying variables by a natural number denoting the distance it has to its binder (with the distance being incremented for each binder occurring between the variable and its binder). As such, a λ-term written using deBruijn indices is invariant with respect to α-conversion, and as such, checking for α-equivalence amounts to checking for syntactic equality.

Canonicalisation is a general term for making something canonical (on standard, or normal, form). In the context of computer science, this involves converting data which has many possible representations into a canonical representation. In the context of our problem at hand, this would be ensuring that bound names and variables (identifiers) have a canonical "value", which is associated with the identifier. As such, α-converting the name or variable would not change the "value" of the name or variable.

In [BDNN01], canonicalisation is accomplished by annotating input prefixes and restrictions with binders and values, respectively, and by making a minor modification to the semantics of the π-calculus. For instance, consider these structural equivalence rules for α-conversion.

$$\begin{aligned} a(x^\beta)P &\equiv_\alpha a(y^\beta)P\{{}^{y}/_{x}\}, && \text{if } y \notin \mathsf{fv}(P) \\ (\nu u^\chi)P &\equiv_\alpha (\nu v^\chi)P\{{}^{v}/_{u}\}, && \text{if } v \notin \mathsf{fu}(P) \end{aligned}$$

In (νu^χ), χ is the "value" associated with the identifier u (we call such values χ-values henceforth). In $a(x^\beta)$, β is the "binder", which gets instantiated with the "value" of the identifier stored in x (we call such binders β-binders henceforth). Free names and variables (which cannot be α-converted, by the way) are assumed to have χ-values' associated with them. This can be represented by a map provided alongside the process specification (as those χ-values cannot be read from the process specification; the restrictions are beyond the process specification).

The other changes to the semantics of ABπ' are fairly minor, as can be seen in Tables 5.6, 5.7, and 5.8. All the other rules are near-identical. Note that when applying

(new-C)	$(\nu u^{\chi_1})(\nu v^{\chi_2})A \equiv (\nu v^{\chi_2})(\nu u^{\chi_1})A,$	if $u \neq v$
(alias)	$(\nu x^\chi)\{{}^{T}/_{x}\} \equiv \mathbf{0}$	

Table 5.6: Canonicalising the Structural Equivalence of ABπ'.

(sync)	$\overline{a}\langle u\rangle.Q \mid a(u^\beta).P \rightarrow Q \mid P$

Table 5.7: Canonicalising the Internal Reduction relation in ABπ'.

(in) $$\frac{}{u(x^{\beta}).P \xrightarrow{u(T)} (\nu x^{\chi}) \mid P}$$

(open-atom) $$\frac{A \xrightarrow{\bar{a}\langle u\rangle} A', \quad u \neq a}{(\nu u^{\chi})A \xrightarrow{(\nu u)\bar{a}\langle u\rangle} A'}$$

Table 5.8: Canonicalising the Labelled Reduction relation of ABπ'.

(in), the χ-value must be a previously-unused χ-values, and when applying (open-atom), the free identifier-χ-value map may need to be updated. Note also that when applying (alias), the χ-value chosen for x must be a previously-unused χ-value (we assume an infinite supply of χ-values). At last, the condition in (new-C) is required since if $u = v$, then the χ-values of the us in A will change (which is not desired).

Assume that for a given process A, we first bring it to a syntactically consistent form A^c, then canonicalise A^c by annotating each restriction and input prefix with a unique χ-value and β-binder, respectively, then supplying a map which maps free names and variables to χ-values which are pairwise different from χ-values already annotated to A^c and each other, and at last ensuring that this map is kept consistent during reductions in A^c annotated (which we denote A^{c^c}). We claim that when generating the Horn clauses from A^c, by registering the χ-values of each restricted name and variable, and the β-binder of each input prefix, instead of just registering the restrictions and input prefixes, then we can accurately associate names and inputs in A^c with their occurrences in $\mathsf{H}_{\mathsf{prot}}$.

5.1.3 Session Enrichment and Horn Clause Generation

Since we have insured that processes remain syntactically consistent, we can move on to handle replication and variable replacement during Horn clause generation.

Session Enrichment

In [Bla02], Blanchet parameterises names and variables with session variables in his Horn clause generation to ensure that all names and variables remain syntactically distinct, even in the presense of replication. To make A_0 (which is what we call the initial process now, since they have become extended processes) comparable to H during reduction, we introduce the same parameterisation in the *session-introducing encoding* $[\![\cdot]\!]^{S}_{\tilde{i},\mathsf{f}}$, which is defined in Table 5.9. We thus let $[\![A_0]\!]^{S}_{\emptyset,\mathsf{I}}$ (where I is the identity map) be the *session-enriched* A_0. We write A_0^S as a shorthand for $[\![A_0]\!]^{S}_{\emptyset,\mathsf{I}}$ when discussing both A_0 and its session-enriched form.

$$
\begin{aligned}
[\![\mathbf{0}]\!]^{S}_{\tilde{i},\mathsf{f}} &= \mathbf{0}\\
[\![A \mid B]\!]^{S}_{\tilde{i},\mathsf{f}} &= [\![A]\!]^{S}_{\tilde{i},\mathsf{f}} \mid [\![B]\!]^{S}_{\tilde{i},\mathsf{f}}\\
[\![(\nu u)A]\!]^{S}_{\tilde{i},\mathsf{f}} &= (\nu u[\tilde{i}])[\![A]\!]^{S}_{\tilde{i},\mathsf{f}[u\mapsto u[\tilde{i}]]}\\
[\![\{{}^{T}/_{x}\}]\!]^{S}_{\tilde{i},\mathsf{f}} &= \{{}^{[T]^{S}_{\tilde{i},\mathsf{f}}}/_{[x]^{S}_{\tilde{i},\mathsf{f}}}\}\\
[\![T(u).P]\!]^{S}_{\tilde{i},\mathsf{f}} &= [\![T]\!]^{S}_{\tilde{i},\mathsf{f}}([\![u]\!]^{S}_{\tilde{i},\mathsf{f}}).[\![P]\!]^{S}_{\tilde{i},\mathsf{f}}\\
[\![\overline{T}\langle U\rangle.P]\!]^{S}_{\tilde{i},\mathsf{f}} &= \overline{[\![T]\!]^{S}_{\tilde{i},\mathsf{f}}}\langle[\![U]\!]^{S}_{\tilde{i},\mathsf{f}}\rangle.[\![P]\!]^{S}_{\tilde{i},\mathsf{f}}\\
[\![!^{i'\geq n}P]\!]^{S}_{\tilde{i},\mathsf{f}} &= !^{i'\geq n}[\![P]\!]^{S}_{\tilde{i}i',\mathsf{f}}\\
\left[\!\!\left[\begin{array}{l}\textbf{let } z = \mathsf{g}(T_1,\ldots,T_n)\\ \textbf{in } P\\ \textbf{else } Q\end{array}\right]\!\!\right]^{S}_{\tilde{i},\mathsf{f}} &= \begin{array}{l}\textbf{let } [\![z]\!]^{S}_{\tilde{i},\mathsf{f}} = \mathsf{g}([\![T_1]\!]^{S}_{\tilde{i},\mathsf{f}},\ldots,[\![T_n]\!]^{S}_{\tilde{i},\mathsf{f}})\\ \textbf{in } [\![P]\!]^{S}_{\tilde{i},\mathsf{f}}\\ \textbf{else } [\![Q]\!]^{S}_{\tilde{i},\mathsf{f}}\end{array}\\
[\![u]\!]^{S}_{\tilde{i},\mathsf{f}} &= \mathsf{f}(u)\\
[\![\mathsf{f}(T_1,\ldots,T_n)]\!]^{S}_{\tilde{i},\mathsf{f}'} &= \mathsf{f}([\![T_1]\!]^{S}_{\tilde{i},\mathsf{f}'},\ldots,[\![T_1]\!]^{S}_{\tilde{i},\mathsf{f}'})
\end{aligned}
$$

Table 5.9: Session-introducing Encoding.

Variable Consistency Enrichment

We can assume that session identifiers are already present in the syntax of our initial process, as we have just shown how to do so. As a final modification to the static analysis of Abadi and Bruno, we add means of keeping track of the original variable names of those variables replaced with terms in ρ during the Horn clause generation. This occurs when "let" expressions are encountered, as reductions in the process under a successful "let" expression assigning $x = \mathsf{g}(\tilde{T})$ are contingent on the pattern of the value of x which made the "let" destruction possible.

We do this by adding *mappings* to the Horn clauses, which domain is the variable as it appears in the initial process, and value is the condition we place on its instantiations. The result is the Horn clause generator in Table 5.10. This, and the syntactic transformations presented prior in this chapter, are the key to enabling us to prove the relation between Horn clauses and reductions in the initial process.

Recall that in substitution σh, σ does *not* substitute on the subject of the mapping $u \mapsto T$; only the object. That is, $\sigma(u \mapsto T) = u \mapsto \sigma T$.

We assume assume that new variables introduced by (σ, σ') when a "let" is encountered during Horn clause generation always differ from all variables in A_0, and in ρ. This can be accomplished by an approach similar to the one applied to bring A_0 to syntactically consistent form.

$$\begin{aligned}
[\![\mathbf{0}]\!]_{\rho h} &= \emptyset \\
[\![P \mid Q]\!]_{\rho h} &= [\![P]\!]_{\rho h} \cup [\![Q]\!]_{\rho h} \\
[\![!^{i \geq n} P]\!]_{\rho h} &= [\![P]\!]_{\rho h} \\
[\![(\nu a)P]\!]_{\rho h} &= [\![P]\!]_{(\rho[a \mapsto a[\rho(\mathcal{V}_o)]])h} \\
[\![u(x).P]\!]_{\rho h} &= [\![P]\!]_{(\rho[x \mapsto x])(h \wedge \mathsf{msg}(u \mapsto \rho(u), x \mapsto x))} \\
[\![\overline{u}\langle v\rangle.P]\!]_{\rho h} &= [\![P]\!]_{\rho h} \cup \{((h \implies \mathsf{msg}(u \mapsto \rho(u), v \mapsto \rho(v))), (\rho, h))\}
\end{aligned}$$

$$\left[\!\!\left[\begin{array}{l}\textbf{let } x = \mathsf{g}(T_1, \ldots, T_n) \\ \textbf{in } P \\ \textbf{else } Q\end{array}\right]\!\!\right]_{\rho h} = \bigcup \left\{ [\![P]\!]_{((\sigma\rho)[x \mapsto \sigma' p'])(\sigma h)} \;\middle|\; \begin{array}{l} \mathsf{g}(p'_1, \ldots, p'_n) > p' \in \mathrm{def}(\mathsf{g}), \\ \text{and } (\sigma, \sigma') \text{ is the most gene-} \\ \text{ral pair of substitutions} \\ \text{st. } \sigma\rho(T_i) = \sigma' p'_i; 1 \leq i \leq n \end{array} \right\} \cup \; [\![Q]\!]_{\rho h}$$

Table 5.10: Horn clauses for the protocol, extended.

5.2 Revelation Semantics

Recall the issue mentioned in the beginning of this chapter regarding internal reduction and the consistency of the frame of a process. Also, to properly compare Horn clauses to transitions, we need the name and term involved in synchronisation actions in internal reductions. We address these two issues by defining a new semantics for ABπ' which, compared to the labelled reduction semantics of ABπ', enriches labels and ensures that frames remain consistent, even in the presence of internal reductions on free names. Briefly put, what we do is

- Remove the (sync) rule from $\rightarrow$, and
- Add corresponding rules to $\xrightarrow{\alpha}$, such that when A_0 synchronises internally on a free channel, the message is implicitly leaked.

We model implicit leakage by exposing the (bound) object sent during sychronisation, thus making the object appear in the frame of the reduced process. The result is the following two new relations.

Definition 5.5 (Silent Reduction)
Silent reduction, denoted by $\rightharpoonup$, is the least preorder on processes closed by structural equivalence and evaluation contexts, and satisfying the (destr$_1$) and (destr$_2$) axioms in Table 5.3. □

Definition 5.6 (Revelation Reduction)
Revelation reduction, denoted $\xrightarrow{\alpha}\!\!\triangleright$, extends the rules of $\xrightarrow{\alpha}$ with the rules in Table 5.11.□

Note that C and D in Table 5.11 are evaluation contexts. Also note the negation on the transition in the last premise of rule (sync$_{\text{int}}$) in Table 5.11. This does not cause problems,

$$(\text{sync}_{\text{leak}})\quad \frac{A = C[(\nu u)D[\overline{a}\langle u\rangle.P_1, a(x).P_2]]\ \ A \overset{(\nu u)\overline{a}\langle u\rangle}{\rightharpoondown} C[D[P_1, a(x).P_2]] \qquad D[\overline{a}\langle u\rangle.P_1, a(x).P_2] \overset{a(u)}{\rightharpoondown} D[\overline{a}\langle u\rangle.P_1, (\nu x)(\{{}^{u}/_{x}\} \mid P_2)]}{A \overset{(\nu u)\hat{a}(u)}{\rightarrowtriangle} C[D[P_1, (\nu x)(\{{}^{u}/_{x}\} \mid P_2)]]}$$

$$(\text{sync}_{\text{int}})\quad \frac{A = C[B]; B = D[\overline{a}\langle u\rangle.P_1, a(x).P_2] \quad B \overset{\overline{a}\langle u\rangle}{\rightharpoondown} D[P_1, a(x).P_2] \qquad B \overset{a(u)}{\rightharpoondown} D[\overline{a}\langle u\rangle.P_1, (\nu x)(\{{}^{u}/_{x}\} \mid P_2)] \quad A \overset{(\nu u)\overline{a}\langle u\rangle}{\not\rightharpoondown}}{A \overset{\hat{a}(u)}{\rightarrowtriangle} C[D[P_1, (\nu x)(\{{}^{u}/_{x}\} \mid P_2)]]}$$

Table 5.11: Revelation Semantics in ABπ.

since any given process specification is a finite strings, thus with a finite number of output prefixes to check.

Finally, we define some notation for our reduction relations. We let $\rightharpoonup^*$ denote the reflexive and transitive closure of $\rightharpoonup$, and define

$$\begin{aligned}
(\overset{\alpha}{\Rightarrow\!\triangleright}) &\overset{\text{def}}{=} (\rightharpoonup^* \cdot \overset{\alpha}{\rightarrowtriangle} \cdot \rightharpoonup^*)\\
(\overset{\alpha_1\cdots\alpha_n}{\rightarrowtriangle}) &\overset{\text{def}}{=} (\overset{\alpha_1}{\rightarrowtriangle} \cdots \overset{\alpha_n}{\rightarrowtriangle})\\
(\overset{\alpha_1\cdots\alpha_n}{\Rightarrow\!\triangleright}) &\overset{\text{def}}{=} (\overset{\alpha_1}{\Rightarrow\!\triangleright} \cdots \overset{\alpha_n}{\Rightarrow\!\triangleright})\\
(\overset{\alpha}{\longrightarrow\!\!\blacktriangleright}) &\overset{\text{def}}{=} (\equiv \cdot \overset{\alpha}{\rightarrowtriangle} \cdot \equiv).
\end{aligned}$$

Let $\overset{\tilde{\alpha}}{\rightarrowtriangle}_* \overset{\tilde{\alpha}}{\Rightarrow\!\triangleright}_*$ denote the reflexive and transitive closure of $\overset{\alpha}{\rightarrowtriangle}$ and $\overset{\alpha}{\Rightarrow\!\triangleright}$, respectively. If $A \overset{\alpha_1\cdots\alpha_n}{\rightarrowtriangle} A'$, then there exists a sequence $A_1, \ldots, A_{n-1}$ such that

$$A \overset{\alpha_1}{\rightarrowtriangle} A_1 \cdots A_{n-1} \overset{\alpha_n}{\rightarrowtriangle} A',$$

and likewise for $A \overset{\alpha_1\cdots\alpha_n}{\Rightarrow\!\triangleright} A'$. Finally, let $A \rightarrow'$ be some A' where $(A, A') \in \rightarrow'$, where $\rightarrow'$ is any one of the transition relations presented in this thesis.

Proposition 5.6 $((\equiv \cup \rightharpoonup \cup \overset{\alpha}{\rightarrowtriangle}) \subsetneq (\equiv \cup \rightarrow \cup \overset{\alpha}{\rightharpoondown})^*)$
The labelled operational semantics of ABπ' emulates the revelation operational semantics of ABπ'. That is,

i) If $A \overset{\alpha}{\rightarrowtriangle} A'$, then $A \overset{\tilde{\alpha}}{\longrightarrow} A'$.

ii) If $A \rightharpoonup A'$, then $A \rightarrow A'$.

PROOF
Follows from the fact that $\rightharpoonup \subsetneq \rightarrow$, and that $\overset{\alpha}{\rightarrowtriangle}$ is defined in terms of $\overset{\alpha}{\rightharpoondown}$. ■

Proposition 5.7 $((\equiv \cup \rightarrow \cup \xrightarrow{\alpha}) \not\subseteq (\equiv \cup \rightharpoonup \cup \xrightarrow{\alpha})^*)$
The converse of Proposition 5.6 does not hold. That is, there exist extended processes A, A' such that $A(\equiv \cup \rightarrow \cup \xrightarrow{\alpha})A'$, but not $A(\equiv \cup \rightarrow \cup \xrightarrow{\alpha})^*A'$.

Proof
By counterexample. Assume the converse of Proposition 5.6 holds. Consider

$$A \overset{\text{def}}{=} (\nu x)(\{^T/_x\} \mid \overline{a}\langle x\rangle.P \mid a(y).Q) \rightarrow (\nu x)(\{^T/_x\} \mid P \mid (\nu x)(\{^T/_y\} \mid Q)) \overset{\text{def}}{=} A',$$

for suitable T, P and Q. The combination that a is free and x is bound makes it impossible to apply (sync$_{\text{leak}}$) or (sync$_{\text{int}}$) to reduce A to A': For (sync$_{\text{leak}}$) to not remove the restriction on x, a must be bound. Since there is no "close"-rule in $\equiv$, $\rightharpoonup$, and $\xrightarrow{\alpha}$, (sync$_{\text{leak}}$) cannot be applied. For (sync$_{\text{int}}$) to be applied, then either, x must be free, or a must be bound, neither of which is true, and neither of which can be accomplished by applying $\equiv$ or $\rightharpoonup$.

As there are no rules in $\equiv$ and $\rightharpoonup$ to handle reductions, and since applying $\equiv$ or $\rightharpoonup$ rules does not enable the desired reduction, the result follows. ■

In summary, while revelation reduction can be emulated by select labelled reductions, the act of synchronising a bound object over a free subject, without exposing the object, is impossible in the revelation semantics. This is, in fact, desired, as this ensures that the frame of the process is kept up-to-date through reduction.

5.3 Summary

We have now overcome the issue with syntactically comparing an initial process A_0 with the Horn clauses H generated from it, and thus paved the way towards proving the soundness of deduction from H. The issues we addressed can be summarised as follows.

(1) The initial process could contain restrictions on names and variables which were syntactically (but not conceptually) equal.

(2) Replicated processes produced new name and variable restrictions which, again, were syntactically (but not conceptually) equal.

(3) Restricted names and variables could be replaced through α-conversion.

(4) After a process received a message, the input variable disappeared from the process, making it impossible to discover the value assigned to it during the input operation (which we require to perform unifications in the proof). The same issue was present in the semantics of successful "let" reductions in ABπ'.

(5) Variables could be replaced by terms during Horn clause generation, some terms containing fresh variables which did not appear in the syntax of the initial process.

(6) The environment knowledge was not updated correctly during internal reduction involving the sending of a bound object over a free subject.

We used the *syntactic consistency* encoding to address issues (1). *Canonicalisation* enables us to safely disrecard issue (3). By using *session enrichment*, issue (2) is no longer a problem, since, in the semantics of ABπ', the session variables are instantiated during replication in the enriched initial process. We addressed issue (4) through the *semantics of ABπ'*. Since we have no analogue of the (subst) rule from Aπ, we can be certain that the active substitution is still "floating" in the process immediately after reduction. Issue (5) we address with our *variable consistency* encoding, which makes sure to keep track of the variables originally in place of the terms substituted into the Horn clauses due to unification in "let" destruction. Finally, issue (6) is taken care of by our *revelation semantics*.

We note that the revelation semantics can, with slight variations, be applied in Aπ and similar calculi, and doing so is particularly desirable when applying the Dolev-Yao threat model.

We are now ready to commence the proof of soundness of H.

CHAPTER 6

Soundness of H

It is time to prove two of the main results of this thesis. Specifically, we prove that "for any sequence of transitions in an initial process, there is a Horn clause in the Horn clauses generated from the initial process which the transition sequence initialises". From this, we can deduce that performing transitions in the initial process has a clear relation to performing resolution in the Horn clauses H generated from the initial process. It turns out that deduction of truths from H is *sound* with regards to the behaviour of the initial process. As such, if we cannot deduce from H that a message T can be sent on channel a, then the same holds for the initial process.

To prove this fact, we impose the following restrictions on the Horn clause generation, and the initial process p.

- We *never* α-convert names and input prefixes during reduction.
- We assume p is syntactically-consistent, and session-enriched.
- We assume the Horn clauses were generated with the variable-consistent Horn clause generator.
- We disregard parameterisation of restricted names and variables[1].
- Perform reductions in p in the *revelation semantics* of ABπ'.

As explained in the summary of the previous chapter, imposing these restrictions will not be harmful for the accuracy of our result.

[1]This will not cause problems for session identifiers, since each parallel component which contains names or variables parameterised with i, only has one value assignment to i. Thus, unifying session variables in the Horn clauses with session variable instances in the process during reduction is trivial.

We apply the following transition relations in the proofs.

$$(\rightharpoonup\!\!\!\!=) \overset{\text{def}}{=} (\equiv \cdot \rightharpoonup \cdot \equiv)$$
$$(\xRightarrow{\alpha}) \overset{\text{def}}{=} (\rightharpoonup\!\!\!\!=^{*} \cdot \xrightarrow{\alpha} \cdot \rightharpoonup\!\!\!\!=^{*})$$

6.1 H Overapproximates All Reachable Frames

We begin by proving an interesting consequence of our enriched calculus, and the revelation semantics. Note that by $P \rightarrow$, where $\rightarrow$ is any reduction relation, we mean a process P' such that $P \rightarrow P'$.

Theorem 6.1 (Overapproximation of all reachable frames)
Let $A \in \mathscr{A}$ and $S = \mathsf{fn}(A)$, and assume the infinite set of names used in the condition of (ax$_2$)*, Definition 3.11, is defined by $\{b[i] \mid i \in \mathbb{N}\}$ for some $b \notin \mathsf{n}(A)$, and that this b is the same b selected in Algorithm 3.1 during the creation of* H*. It holds for any sequence of reductions $\xRightarrow{\alpha_1 \cdots \alpha_{n-1}} \xrightarrow{\alpha_n}$ in A that*

$$\varphi(A \xRightarrow{\alpha_1 \cdots \alpha_{n-1}} \xrightarrow{\alpha_n}) \vdash T \implies \mathsf{H} \vdash \mathsf{att}(T).$$

Proof
By induction in n.

Basis: $n = 0$. Here we are considering $\varphi(A)$. By Algorithm 3.1 lines 4, 6, 7, and 9, and by the definition of a frame from Section 3.1.2, we see that deduction in $\mathsf{H}_{\mathsf{att}}$ is exactly deduction in $\varphi(A)$. That is, $\varphi(A) \vdash T \iff \mathsf{H}_{\mathsf{att}} \vdash \mathsf{att}(T)$. Since $\mathsf{H}_{\mathsf{att}} \subseteq \mathsf{H}$, the result follows.

Induction: Assume the theorem holds for n. That is, the theorem holds for the transition sequence $\xRightarrow{\alpha_1 \cdots \alpha_{n-1}} \xrightarrow{\alpha_n}$. The theorem then also holds for the transition sequence $\xRightarrow{\alpha_1 \cdots \alpha_n}$, as no new messages have been passed since $\xRightarrow{\alpha_1 \cdots \alpha_{n-1}} \xrightarrow{\alpha_n}$. We seek to prove the theorem for the transition sequence $\xRightarrow{\alpha_1 \cdots \alpha_n} \xrightarrow{\alpha_{n+1}}$.

By the definition of a frame from Section 3.1.2, the only time there is a term T st. $\varphi(A \xRightarrow{\alpha_1 \cdots \alpha_n}) \nvdash T$ but $\varphi(A \xRightarrow{\alpha_1 \cdots \alpha_n} \xrightarrow{\alpha_{n+1}}) \vdash T$ is if something new is exposed in $\xrightarrow{\alpha_{n+1}}$ enabling this deduction. We can assume (without loss of generality) that T is this exposed term (for then it follows from the autoepistemy of $\vdash$ for Horn clauses that if $\mathsf{H} \vdash \mathsf{att}(T)$, then $\mathsf{H} \vdash \mathsf{att}(U)$ for any $U \in \varphi(A \xRightarrow{\alpha_1 \cdots \alpha_n} \xrightarrow{\alpha_{n+1}}) \setminus \varphi(A \xRightarrow{\alpha_1 \cdots \alpha_n}$ $))$.

Terms are only exposed to the environment if $\alpha_{n+1} \in \{(\nu u)\overline{a}\langle u\rangle, (\nu u)\hat{a}(u)\}$, where $\varphi_{\mathrm{k}}(A \xRightarrow{\alpha_1 \cdots \alpha_n})(u) = T$. By Definition 5.6, since an exposing output occurs in $\xrightarrow{(\nu u)\hat{a}(u)}$, it is enough to consider $\alpha_{n+1} = (\nu u)\overline{a}\langle u\rangle$.

From the validity of $A \xRightarrow{\alpha_1 \cdots \alpha_n} \xrightarrow{\alpha_{n+1}}$ and by Definition 5.6, we get that $\mathsf{H} \vdash \mathsf{att}(a)$ and $\mathsf{H} \vdash \mathsf{msg}(a, u)$. Then by Algorithm 3.1, we get that $\mathsf{H} \vdash \mathsf{att}(u)$, and thus, $\mathsf{H} \vdash \mathsf{att}(T)$. This completes the proof. ∎

This means that we can apply the Horn clauses generated from a process to (overapproximately) prove whether a process satisfies *active syntactic secrecy.* We state (and prove) this fact in Section 6.3.

6.2 Soundness Theorem

A couple of functions we use in the soundness proof:

$$\begin{aligned} \mathsf{H}_{\mathsf{att}}(\varphi(A)) &\overset{\text{def}}{=} \mathsf{H}_{\mathsf{att}}(A) \cup \{\mathsf{att}(T) \mid T \in \mathrm{im}(\varphi(A))\} \\ \mathsf{in}(A, B) &\overset{\text{def}}{=} \mathrm{dom}(\varphi_{\mathrm{k}}(B)) \setminus \mathrm{dom}(\varphi_{\mathrm{k}}(A)) \end{aligned}$$

Note that $\mathsf{H}_{\mathsf{att}}(\varphi(A \xRightarrow{\tilde{\alpha}})) \vdash p \implies \mathsf{H}(A) \vdash p$, for any $\tilde{\alpha}$, where by $\mathsf{H}_{\mathsf{att}}(\varphi(A \xRightarrow{\tilde{\alpha}}))$ we mean the set of Horn clauses obtained by converting any term deducible from $\varphi(A \xRightarrow{\tilde{\alpha}})$ to predicates, and adding them to $\mathsf{H}_{\mathsf{att}}(\mathbf{0}, \mathsf{fn}(A))$.

The idea with $\mathsf{in}(A, B)$ is that when $A \xrightarrow{\hat{a}(y)} B$ or $A \xrightharpoondown{a(T)} B$, then $\mathsf{in}(A, B)$ will yield which variable was written to (since no structural equivalence rules can be applied during the input operation in $\xrightarrow{\alpha}$ and $\xrightharpoondown{\alpha}$ (we made sure of this in the semantics definition of $\xrightarrow{\alpha}$ and $\xrightharpoondown{\alpha}$), the input variable will still be present, with a floating active substitution present in B expressing its value). Then, if the input variable is x, $\varphi(B)(\mathsf{in}(A, B))$ will yield the value of x in B.

Theorem 6.2 (Soundness of H' w.r.t. A^S)
Let A be on syntactic consistent form, and on inner normal form. It holds that for any transition sequence $\xRightarrow{\alpha_1 \cdots \alpha_{n-1}} \xrightarrow{\alpha_n}$ from A^S, then either

i) $\mathsf{H}_{\mathsf{att}}(\varphi(A^S \xRightarrow{\alpha_1 \cdots \alpha_{n-1}})) \vdash \mathsf{msg}(a, T)$, *where $\alpha_n = a(T)$, or*

ii) There exists a clause

$$\left(\left(\bigwedge_{1 \leq i < m} \mathsf{msg}(u_i \mapsto U_i, x_i \mapsto T_i) \implies \mathsf{msg}(u_m \mapsto U_m, T_m)\right), (\rho, h)\right) \in \mathsf{H}'_{\mathsf{prot}}(A^S),$$

and a strictly increasing sequence $\mathsf{f} : \mathbb{N} \longrightarrow \mathbb{N}$, such that the set of equations

$$S = \left\{ \begin{aligned} \varphi_{\mathrm{k}}(A^S \xRightarrow{\alpha_1 \cdots \alpha_{\mathsf{f}(i)-1}} \xrightarrow{\mathsf{f}(i)})(x'_{\mathsf{f}(i)}) &= \delta T_i, \\ a_{\mathsf{f}(i)} &= \delta U_i \end{aligned} \right\}$$

is a system of syntactic identities, δ maps each $u \in \bigcup_{T \in \mathrm{im}(\rho)} \mathsf{v}(T) \setminus \mathsf{v}(A^S)$ to a closed term, and where $\mathsf{in}(A^S \xRightarrow{\alpha_1 \cdots \alpha_{\mathsf{f}(i)-1}}, A^S \xRightarrow{\alpha_1 \cdots \alpha_{\mathsf{f}(i)-1}} \xrightarrow{\alpha_{\mathsf{f}(i)}}) =: x'_{\mathsf{f}(i)} = x_i$ (last equality being syntactic), $\alpha_i \in \{a_i(V_i), \hat{a_i}(v_i)\}$, $\alpha_n \in \{\overline{a_n}\langle v_n\rangle, \hat{a_n}(v_n)\}$ for $1 \leq i < n$, and $\mathsf{f}(m) = n$. $\square$

Intuitively, the theorem states that for a given initial process A_0, if we bring A_0 to syntactically consistent form A_0^c and extend A_0^c with sessions (yielding A_0^{cS}), then any transition sequence in A_0^{cS} will have a matching function f which associates each predicate in some Horn clause cl in $\mathsf{H}'(A_0^{cS})$, with the transition in the transition sequence instantiating the predicate (and do so in proper order).

Proof
By induction in n.

Basis: $(n = 1)$. Five cases to consider.

$A^S \rightleftharpoons^* \xrightarrow{a(T)} A^{S\prime}$**:** Then $\varphi(A^S) \vdash a$ and $\varphi(A^S) \vdash T$. It then follows from Theorem 6.1 that $\mathsf{H}' \vdash \mathsf{att}(a)$ and $\mathsf{H}' \vdash \mathsf{att}(T)$, and thus by Algorithm 3.1 that $\mathsf{H}' \vdash \mathsf{msg}(a, T)$.

$A^S \rightleftharpoons^* \xrightarrow{\hat{a}(u)} A^{S\prime}$**:** Then by Definition 5.6, there exists an evaluation context C such that $A_0 = C[\overline{a}\langle u\rangle.Q]$ for some Q. As h in $[\![\cdot]\!]'_{\rho h}$ is only extended when an input token is encountered in the parameterised process, we get by Table 5.10 that there is (at least one) Horn clause $cl := (\mathsf{msg}(u', T'), (\rho, h)) \in \mathsf{H}'$ with no premises.

We now inductively find the values assigned during each "let" destruction in the $\rightleftharpoons^*$ transition sequence (from A to $A \rightleftharpoons^*$): Let $C_j[R] = C'_j[\mathbf{let}\ z_j = \mathsf{g_j}(\tilde{T_j})\ \mathbf{in}\ C_{j+1}[R]\ \mathbf{else}\ Q_j]$ for some Q_j, and let $C_{k+1}[R]$ have no successful let above R, where C'_j for $1 \leq j \leq k$ and C_{k+1} are evaluation contexts. Then let k be the largest integer such that $A^S = C_0[C_1[\overline{v}\langle w\rangle.Q]]$ (with C_0 an evaluation context), where

$$\begin{aligned} C_1[\overline{u}\langle v\rangle.Q] &\rightleftharpoons^* C'_1[(\nu z_1)(\{{}^{T_1}/_{z_1}\} \mid C'_2[(\nu z_2)(\{{}^{T_2}/_{z_2}\} \mid \cdots C_{k+1}[\overline{v}\langle w\rangle.Q] \cdots)])] \\ &\equiv C'_1[(\nu z_1)(\{{}^{T_1}/_{z_1}\} \mid C'_2[(\nu z_2)(\{{}^{T_2}/_{z_2}\} \mid \cdots C_{k+1}[\overline{a}\langle u\rangle.Q] \cdots)])] \\ &\xrightarrow{\overline{a}\langle u\rangle} C'_1[(\nu z_1)(\{{}^{T_1}/_{z_1}\} \mid C'_2[(\nu z_2)(\{{}^{T_2}/_{z_2}\} \mid \cdots C_{k+1}[Q] \cdots)])] \\ &=: P, \end{aligned}$$

and

$$C_0[\mathbf{0}] \xrightarrow{a(u)} C'_0[\mathbf{0}]$$
$$C_0[C_1[\overline{v}\langle w\rangle.Q]] \rightleftharpoons^* \xrightarrow{a(u)} C'_0[P] \equiv A^{S\prime}.$$

(and these contexts exist, as per Definition 5.6 and Definition 5.5). Let $A_0 \mathrel{\underline{\rightharpoonup}}^{*}= A_0(\equiv \cdot \rightharpoonup)_k$, where $(\equiv \cdot \rightharpoonup)_k$ represents k sequential relation compositions of the $(\equiv \cdot \rightharpoonup)$ relation. Now unify

$$S = \{\varphi_{\mathrm{k}}(A(\equiv \cdot \rightharpoonup)_i)(z'_i) \stackrel{?}{=} \rho(z_i)\}_{1\leq i\leq k}$$

with regards to the variables

$$\bigcup_{1\leq i\leq k} \mathsf{v}(\rho(z_i)) \setminus \mathsf{v}(A),$$

where $\mathsf{in}(A^S(\equiv \cdot \rightharpoonup)_{i-1}, A^S(\equiv \cdot \rightharpoonup)_i) =: z'_i = z_i$ (last equality being syntactic), to yield a solution δ.

For at least one of the cl clauses, there must exist a solution δ; this follows from Table 5.10, Definition 5.5, and Theorem 2.10 (from the generality of unification; since the Horn clauses are as general as possible, and the reduction $\mathrel{\underline{\rightharpoonup}}^{*}\xrightarrow{\hat{a}(u)}$ was possible in A^S, then this sequence of reductions must instantiate at least one cl). By picking a cl for which the unification problem

$$\left\{\begin{array}{l} \delta u' \stackrel{?}{=} a, \\ \delta T' \stackrel{?}{=} \varphi_{\mathrm{k}}(A \mathrel{\underline{\rightharpoonup}}^{*})(u) \end{array}\right\}$$

with regards to the variables $\mathsf{v}(T') \setminus \mathsf{v}(A)$ has a solution $\sigma_1 = \emptyset$ (the equations in the unification problem are syntactic equalities, thanks to δ), and by picking f as the identity map, we are done.

$A^S \mathrel{\underline{\rightharpoonup}}^{*}\xrightarrow{(\nu u)\hat{a}(u)} A^{S'}$**:** same argument as $A^S \mathrel{\underline{\rightharpoonup}}^{*}\xrightarrow{\hat{a}(u)} A'$.

$A^S \mathrel{\underline{\rightharpoonup}}^{*}\xrightharpoondown{\overline{a}\langle u\rangle} A^{S'}$**:** same argument as $A^S \mathrel{\underline{\rightharpoonup}}^{*}\xrightarrow{\hat{a}(u)} A'$.

$A^S \mathrel{\underline{\rightharpoonup}}^{*}\xrightharpoondown{(\nu u)\hat{a}(u)} A^{S'}$**:** same argument as $A^S \mathrel{\underline{\rightharpoonup}}^{*}\xrightarrow{\hat{a}(u)} A'$.

Induction: Assume the result holds for n. We now prove that it then holds for $n+1$. Five cases to consider.

$A^S \xRightarrow{\alpha_1,\ldots,\alpha_n}\xrightharpoondown{a(T)} A'$**:** Same argument as that of $A_0 \mathrel{\underline{\rightharpoonup}}^{*}\xrightarrow{a(T)} A^{S'}$ in the basis step.

$A^S \xRightarrow{\alpha_1,\ldots,\alpha_n}\xrightarrow{\hat{a}(u)} A^{S'}$**:** We know by Definition 5.6 that the abstract syntax tree of A^S has a subtree $\overline{v}\langle w\rangle.Q$ for some Q where $\overline{v}\langle w\rangle$ is not executed during $\xRightarrow{\alpha_1,\ldots,\alpha_n}\xrightarrow{\hat{a}(u)}$ until the last action. Let $A^S = C[\overline{v}\langle w\rangle.Q]$ for some C, let $\beta_1,\ldots,\beta_{k'}$ be the input/output actions prefixing the hole in C, and let $C[R] = C'[\beta_{k'}.C''[R]]$.

We now apply the induction hypothesis to get closer to finding f and the Horn clause $\bigwedge_{1\leq i<m} \mathsf{msg}(u_i \mapsto U_i, x_i \mapsto T_i) \implies \mathsf{msg}(u_m \mapsto U_m, T_m) \in \mathsf{H}'_{\mathsf{prot}}(A^S)$. We condition on $\beta_{k'}$.

$\beta_{k'}$ **is an output:** By induction hypothesis, we have (at least one) clause

$$\left(\bigwedge_{1\le i<m} \mathsf{msg}(u_i \mapsto U'_i, x_i \mapsto T'_i) \implies \mathsf{msg}(u'_m \mapsto U'_m, T'_m), (\rho', h')\right)$$

in $\mathsf{H}'_{\mathsf{prot}}(A^S)$, a function f', and a solution δ_{pre} satisfying the criteria in this theorem for a prefix of the sequence $\alpha_1, \dots, \alpha_n$.

By Table 5.10, Theorem 2.10, and Definition 5.5, this Horn clause must be more general than (at least one) clause

$$\left(\bigwedge_{1\le i<m} \mathsf{msg}(u_i \mapsto U_i, x_i \mapsto T_i) \implies \mathsf{msg}(u_m \mapsto U_m, T_m), (\rho, h)\right)$$

which has the same premises (albeit in a less general form).

Following the same approach as in $A^S \rightharpoonup^* \xrightarrow{\hat{a}(u)} A^{S'}$ in the basis step to find δ and σ_i, we unify

$$S' = \{\varphi_{\mathrm{k}}(A \xRightarrow{\alpha_1\cdots\alpha_{\mathsf{f}'(m)}} (\equiv \cdot \rightharpoonup)_i)(z'_i) \stackrel{?}{=} \delta_{\text{pre}}\rho(z_i)\}_{1\le i\le k}$$

with regards to the variables

$$\bigcup_{1\le i\le k} \mathsf{v}(\rho(z_i)) \setminus \mathsf{v}(A),$$

where $\mathsf{in}(A^S \xRightarrow{\alpha_1\cdots\alpha_{\mathsf{f}'(m)}} (\equiv \cdot \rightharpoonup)_{i-1}, A^S \xRightarrow{\alpha_1\cdots\alpha_{\mathsf{f}'(m)}} (\equiv \cdot \rightharpoonup)_i) =: z'_i = z_i$ (last equality being syntactic), to yield a solution δ, and use δ to solve the system of unification problems

$$S_i = \left\{ \begin{array}{r} \varphi_{\mathrm{k}}(A^S \xRightarrow{\alpha_1\cdots\alpha_{\mathsf{f}(i)-1}} \xrightarrow{\mathsf{f}(i)})(x'_{\mathsf{f}(i)}) \stackrel{?}{=} \sigma_{i-1,\dots,1}\delta T_i, \\ a_{\mathsf{f}(i)} \stackrel{?}{=} \sigma_{i-1,\dots,1}\delta U_i \end{array} \right\},$$

each with solution $\sigma_i = \emptyset$ (again, δ makes the equations in each unification problem into syntactic equalities), where

$$\mathsf{in}(A^S \xRightarrow{\alpha_1\cdots\alpha_{\mathsf{f}(i)-1}}, A^S \xRightarrow{\alpha_1\cdots\alpha_{\mathsf{f}(i)-1}} \xrightarrow{\alpha_{\mathsf{f}(i)}}) =: x'_{\mathsf{f}(i)} = x_i$$

(again, the last equality is syntactic), $\alpha_{\mathsf{f}(i)} \in \{a_i(v_i), \hat{a}_i(v_i)\}$, for $1 \le i < n$, $\alpha_n \in \{\overline{a_n}\langle v_n\rangle, \hat{a_n}(v_n)\}$, and $\mathsf{f} := \mathsf{f}'[m \mapsto n+1]$.

$\beta_{k'}$ **is an input:** Recall that

$$A = C'[\beta_{k'}.C''[\overline{a}\langle u\rangle.Q]].$$

Now let

$$C'[\beta_{k'}.C''[\overline{a}\langle u\rangle.Q]] \xRightarrow{\alpha_1,\ldots,\alpha_{i-1}}\xrightarrow{\alpha_i} C'_i[\beta_{k'}.C''[\overline{a}\langle u\rangle.Q]] = A^S_i,$$

where $A^S_i = A^S \xRightarrow{\alpha_1,\ldots,\alpha_{i-1}}\xrightarrow{\alpha_i}$ for $i \leq l < n$ for some l. We now seek to find l. Let $A^{S'} = C'[\mathbf{0}]$ and let $C'[\mathbf{0}] \xRightarrow{\alpha_1,\ldots,\alpha_{i-1}}\xrightarrow{\alpha_i} C'_i[\mathbf{0}] = A^{S'}_i$. Let l be the smallest integer s.t. either $A^{S'} \xRightarrow{\alpha_1,\ldots,\alpha_{l-1}}\not\xrightarrow{\alpha_l}$ or C'_l in A^S_l differs from C'_l in $A^{S'}_l$. We now know α_l is the action matching $\beta_{k'}$. Let

$$A^{S''} = C'[\overline{v}\langle \mathsf{device}\rangle],$$

where device is a new function symbol (a proof device), and $\beta_{k'} = v(w)$. By induction hypothesis, there is (at least one) clause

$$\left(\bigwedge_{1\leq i<m} \mathsf{msg}(u_i \mapsto U'_i, x_i \mapsto T'_i) \implies \mathsf{msg}(u'_{m-1} \mapsto U'_{m-1}, \mathsf{device}), s'\right)$$

in $\mathsf{H}'_{\mathsf{prot}}(A^{S''})$ where $s' = (\rho', h')$, a function f', and a solution δ_{pre} satisfying the criteria in this theorem for the sequence $\alpha_1, \ldots, \alpha_{l-1}, \overline{v}\langle\mathsf{device}\rangle$.

By Table 5.10, Theorem 2.10, and Definition 5.5, the $m-2$ premises of this Horn clause must be more general than (at least one) Horn clause in $\mathsf{H}'(A^S)$,

$$\left(\bigwedge_{1\leq i<m} \mathsf{msg}(u_i \mapsto U_i, x_i \mapsto T_i) \implies \mathsf{msg}(u_m \mapsto U_m, T_m), (\rho, h)\right)$$

with $m-1$ premises (of which the first $m-2$ are in a less general form), where $u_{m-1} = v$.

At last, if we add the equation

$$\{\varphi_{\mathrm{k}}(A \xRightarrow{\alpha_1\cdots\alpha_{l-1}}\xrightarrow{\alpha_l})(w) \overset{?}{=} \delta_{\text{pre}}\rho(w)\}$$

to the unification problem S' in the "$\beta_{k'}$ is an output" case, then the rest of this case will be the same argument as the "$\beta_{k'}$ is an output" case, where we furthermore extend f' with $m-1 \mapsto l$.

$A^S \xRightarrow{\alpha_1,\ldots,\alpha_n}\xrightarrow{(\nu u)\hat{a}(u)} A^{S'}$: Same argument as $A^S \xRightarrow{\alpha_1,\ldots,\alpha_n}\xrightarrow{\hat{a}(u)} A^{S'}$.

$A^S \xRightarrow{\alpha_1,\ldots,\alpha_n}\xrightarrow{\overline{a}\langle u\rangle} A^{S'}$: Same argument as $A^S \xRightarrow{\alpha_1,\ldots,\alpha_n}\xrightarrow{\hat{a}(u)} A^{S'}$.

$A^S \xRightarrow{\alpha_1,\ldots,\alpha_n}\xrightarrow{(\nu u)\overline{a}\langle u\rangle} A^{S'}$: Same argument as $A^S \xRightarrow{\alpha_1,\ldots,\alpha_n}\xrightarrow{\hat{a}(u)} A^{S'}$. ■

6.3 Corollaries

The two theorems proven in the prior two sections are quite powerful, as they directly (syntactically) associate Horn clauses to reductions in the initial process. From these theorems, we present a collection of useful corollaries. The first corollary is important enough to state as a theorem, though.

Theorem 6.3 (Soundness of H w.r.t. A)
Deduction from H *is sound with regards to any (syntactically consistent) initial process* A

Proof
Let $\rightarrow^*$ be any sequence of transitions in A ending in an output action. State an analogue of Theorem 6.2 for Horn clauses in H and un-enriched initial processes. Then, enrich A and generate the appropriate H'. Let m be a map which associates clauses in H' to their corresponding un-enriched counterparts in H. By Theorem 6.2 we have a clause $cl \in \mathsf{H}'$ and a δ satisfying the conditions in Theorem 6.2. By repeatedly applying Theorem 6.2 backwards for each premise of the Horn clause associated with $\rightarrow$ (and to each premise of that clause, and so on) until you reach clauses with no premises (which is possible from the induction hypothesis of Theorem 6.2), we obtain a logic *proof* of $\rightarrow^*$ in H'. The result now follows from applying δ on the corresponding clause $\mathsf{m}(cl)$ (and the other clauses forming the proof), and since $\rightarrow^*$ was arbitrary. ■

We shall refer to this repeated application of the main result as the *backwards proof path.*
Recall that we usually identify processes up to α-conversion.

Theorem 6.4 (Secrecy)
P preserves the secrecy of T from S if $\neg\exists c \in S \,.\, \mathsf{H}(P) \vdash \mathsf{msg}(c, T)$ holds.

Proof
Since $\mathsf{H}(P) \nvdash \mathsf{msg}(c, T)$, we have by Theorem 7.2 that

$$\neg\exists \xRightarrow{\alpha_1\cdots\alpha_{n-1}}\xrightarrow{\alpha_n} .\, \alpha_n \in \{\hat{c}(T), \overline{c}\langle T\rangle\} \text{ and } P \xRightarrow{\alpha_1\cdots\alpha_{n-1}}\xrightarrow{\alpha_n} .$$

Thus, P does not output T on c. The result now follows from the definition of secrecy in [Bla02] since c is arbitrarily chosen from S. ■

Corollary 6.5 (Syntactic Secrecy, in the active case)
If $\mathsf{H}(P, \mathsf{fn}(P)) \nvdash \mathsf{att}(s)$, then s is syntactically secret in P.

Proof
Follows from Theorem 6.1. ■

CHAPTER 7

Secrecy and Authenticity Analysis Framework

The last phase of this thesis is to give the resolution-based automated verification technique for the Distributed Applied π Calculus with Broadcast. While the results from the last chapter make proving that deduction from Horn clauses generated from DAπ_β networks easy, the rigorous path to this result is a long one. Therefore, for convenience and due to space constraints, we will permit ourselves to omit much of the trivial legwork.

Section 7.1 presents a much-needed overview of the proof, explains the steps involved, and which steps will be skipped. Section 7.2 proves the central argument here that if we replace the semantics of the ABπ calculus with a broadcast calculus, then deduction from Horn clauses generated from processes will still be sound with regards to the broadcasting revelation semantics. Section 7.3 explains what changes we need to make to the Horn clause generation, and how we obtain a soundness proof for the Horn clause generator for DAπ_β. Finally, in Section 7.4 we present a small example as of how the exchangeable network topology can be applied to recapture the spoofing attack on the ARAN protocol, as proven originally by Jens Chr. Godskesen in [God06, God07]

7.1 Overview

Figure 7.1 illustrates the steps involved in proving the soundness of the Horn clauses $\mathsf{H}^{\mathcal{T}}_\beta(N)$ generated from some DAπ_β network N.

The darkened vertices are concepts that are already either defined or proven in previous chapters. Defining the enriched ABπ calculus with broadcast, (ABπ'_β), and proving the soundness of the enriched, and subsequently, unenriched Horn clauses for ABπ'_β will be briefly presented in Section 7.2. Recall that DAπ_β only differs from ABπ in that it has a network level (which is really just a context), broadcast, and *connectivity graphs.* We claim that by encoding the message passing constraints which a static abstraction of the

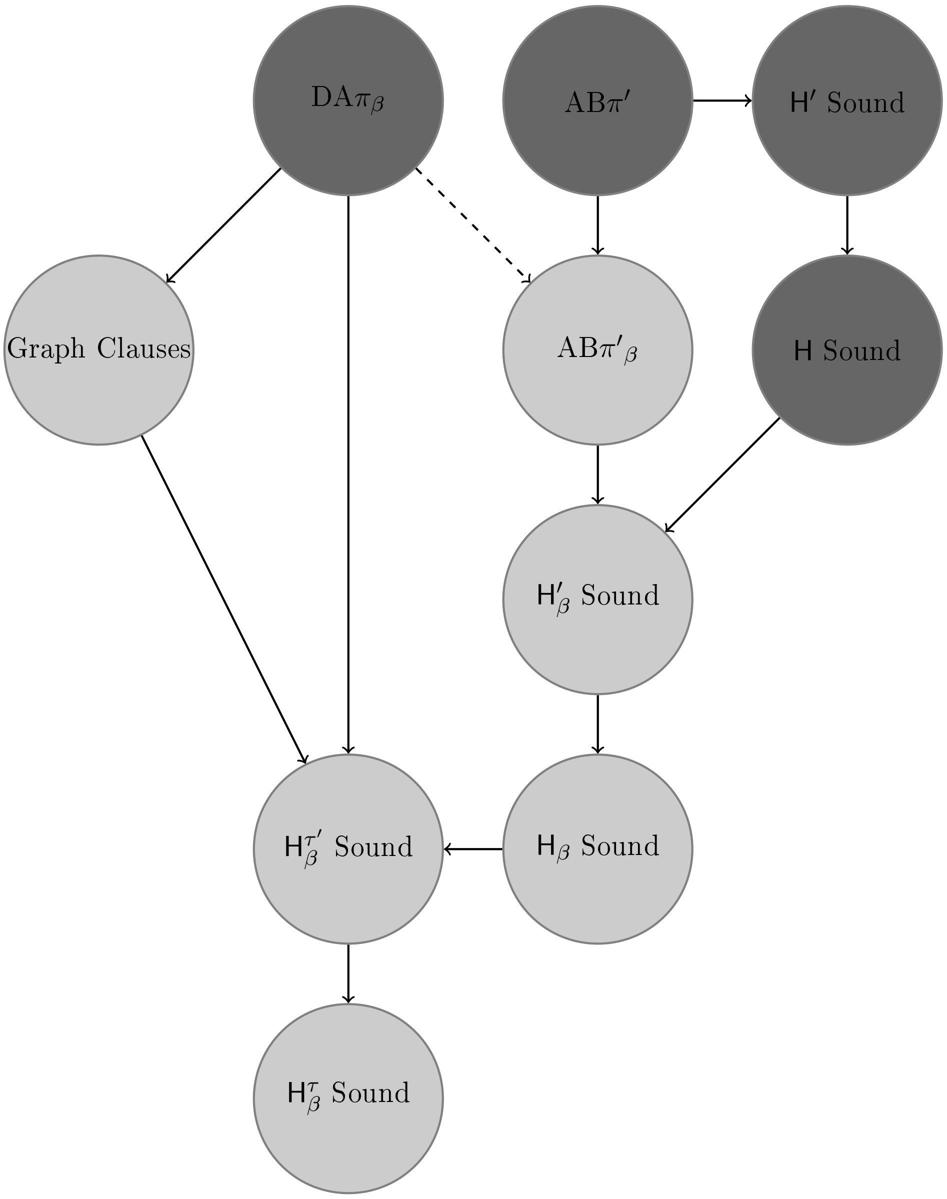

Figure 7.1: Path of the soundness proof of $\mathsf{H}^{\tau}_{\beta}$ for DAπ_β.

network topology imposes as Horn clauses as *graph clauses*, and making otherwise minor changes to the Horn clause generation, that we, through previously proven results, obtain a sound automatic verification technique. We present the idea behind this in Section 7.3.

7.2 H and Broadcast

We now establish that the Abadi/Blanchet Horn clause generator algorithm generalises over point-to-point communication. As such, this resolution-based protocol verification technique can be applied to any broadcast calculus, provided it already can for a point-to-point calculus with the same syntax. First, consider

$$P = (\nu b)(\nu a)(\nu k)\left(\overline{a}\langle k\rangle \mid a(y_1).\overline{b}\langle y_1\rangle \mid a(y_2).b(y_3).\overline{c}\langle \mathsf{oops}(y_2)\rangle\right).$$

In a point-to-point calculus, this process would never output the term $\mathsf{oops}(y_2)$. In a broadcast calculus, however, it would. Now, most importantly, *in both cases*, the predicate $\mathsf{message}(c, \mathsf{oops}(y_2))$ is deducible from H.

Now *assume a broadcast semantics* for ABπ'[1], and all the other restrictions we imposed to A in the previous chapter. The following then holds.

Theorem 7.1 (Soundness of H'_β w.r.t. A^S)
Let A be on syntactic consistent form, and on inner normal form. It holds that for any transition sequence $\xRightarrow{\alpha_1\cdots\alpha_{n-1}}\xrightarrow{\alpha_n}$ from A^S, then either

i) $\mathsf{H}'_{\mathsf{att}\beta}(\varphi(A^S \xRightarrow{\alpha_1\cdots\alpha_{n-1}})) \vdash \mathsf{msg}(a, T)$, *where $\alpha_n = a(T)$, or*

ii) There exists a clause

$$\left(\left(\bigwedge_{1\leq i<m} \mathsf{msg}(u_i \mapsto U_i, x_i \mapsto T_i) \implies \mathsf{msg}(u_m \mapsto U_m, T_m)\right), (\rho, h)\right) \in \mathsf{H}'_{\mathsf{prot}\beta}(A^S),$$

and a strictly increasing sequence $\mathsf{f} : \mathbb{N} \longrightarrow \mathbb{N}$, such that the set of equations

$$S = \left\{ \begin{array}{rl} \varphi_{\mathrm{k}}(A^S \xRightarrow{\alpha_1\cdots\alpha_{\mathsf{f}(i)-1}}\xrightarrow{\mathsf{f}(i)})(x'_{\mathsf{f}(i)}) &= \delta T_i, \\ a_{\mathsf{f}(i)} &= \delta U_i \end{array} \right\}$$

is a system of syntactic identities, δ maps each $u \in \bigcup_{T\in \mathrm{im}(\rho)} \mathsf{v}(T) \setminus \mathsf{v}(A^S)$ to a closed term, and where $\mathsf{in}(A^S \xRightarrow{\alpha_1\cdots\alpha_{\mathsf{f}(i)-1}}, A^S \xRightarrow{\alpha_1\cdots\alpha_{\mathsf{f}(i)-1}}\xrightarrow{\alpha_{\mathsf{f}(i)}}) =: x'_{\mathsf{f}(i)} = x_i$ (last equality being syntactic), $\alpha_i \in \{a_i(V_i), \hat{a}_i(v_i)\}$, $\alpha_n \in \{\overline{a_n}\langle v_n\rangle, \hat{a}_n(v_n)\}$ for $1 \leq i < n$, and $\mathsf{f}(m) = n$.

PROOF
Use the exact same proof as that of Theorem 6.2. ∎

Theorem 7.2 (Soundness of H_β w.r.t. A)
Deduction from H_β is sound with regards to any (syntactically consistent) initial process A

PROOF
Follows from Theorems 7.1 and 7.2. ∎

[1]Leave the syntax of ABπ' unchanged.

7.3 H in DAπ_β

Obtaining a sound Horn clause generator for DAπ_β processes is quite easy. First, we add the restrictions imposed by the network topology into the rules of the protocol.

$$\begin{aligned}\mathsf{msg}(x,y,z) \wedge \mathsf{connected}(z,z') &\implies \mathsf{msg}(x,y,z') \in \mathsf{H}^{\tau}_{\beta}\\ \forall(l,l') \in \mathcal{G}(\tau)\,.\,\mathsf{connected}(l,l') &\in \mathsf{H}^{\tau}_{\beta}\end{aligned}$$

In the original Horn clause generation algorithm, update the msg predicates to also account for the sending location.

Change the attacker clauses for receiving and sending messages as follows.

$$\begin{array}{llcl}\text{(send)} & \mathsf{att}(x) \wedge \mathsf{att}(y) \wedge \mathsf{att}(z) & \implies & \mathsf{msg}(x,y,z)\\ \text{(receive)} & \mathsf{att}(x) \wedge \mathsf{att}(y) \wedge \mathsf{msg}(x,z,w) \wedge \mathsf{connected}(w,y) & \implies & \mathsf{att}(z)\end{array}$$

Given these changes, we claim that the following is true.

Theorem 7.3 (Soundness of $\mathsf{H}^{\tau}_{\beta}$ w.r.t. A)
Deduction from $\mathsf{H}^{\tau}_{\beta}$ is sound with regards to any (syntactically consistent) initial network N

Proof
Follows from Theorems 7.1 and 7.2, and the Horn clause changes expressed above. ■

7.4 Routing Example: ARAN

As a last act, we briefly illustrate how useful the added connectivity graph abstraction is. This example, although grossly simplified, recaptures the essentials of the weakness of the ARAN protocol, proven present by Jens Chr. Godskesen [God06].

Example 7.4 (ARAN (overapproximating idea))
Let $L = \{l_S, l_1, \ldots, l_n, l_A, l_D\}$ be the locations occurring in a model N of the ARAN protocol (except l_A), where l_S is the source of the route discovery message, l_D is the intended destination, and l_A our unspecified attacker, where $l_A \notin \mathsf{l}(N)$, and consider the network topology (admissible to N)

$$\tau = \{G \mid \forall l \in L \backslash \{l_A\}.(l, l_D), (l_D, l) \notin \mathsf{E}(G)\}\,.$$

Assume that upon completion (and only upon completion) of a route discovery, N performs $\xrightarrow{l_S, \mathsf{ok}(k)}_{\mathcal{G}(\tau)}$, where k is a nonce (unknown to the environment). If $\mathsf{H}(N, \mathcal{G}(\tau)) \nvdash \mathsf{msg}(l_S, a, \mathsf{ok}(k))$ for all channels a, all is in order. This is, however, not the case: the static graph abstraction $\mathcal{G}(\tau)$ isolates the destination node in such a manner that *only* the attacker can send messages to it. We claim that since $\mathsf{H}^{\tau}_{\beta}(N) \vdash \mathsf{message}(l_S, a, \mathsf{ok}(k))$, then that must mean that the route went via. the unauthenticated attacker l_A. □

CHAPTER 8

Conclusion

We have given a thorough overview of the various frameworks for verification of secrecy, authenticity and routing properties of security protocols. During this process, we observed and argued that there was need for a generic, and conceptually simple framework for automatic verification of security protocols for MANETs. We proceeded to define the Distributed Applied π Calculus with Broadcast, which is a simple, yet expressive, extension to the well-established Aπ calculus with a network layer, connectivity graphs, and broadcast communication.

By making the observation that Horn clauses generated from a process specification relate to the process specification in a very specific way, we proceed to define a powerful soundness theorem for deduction from Horn clauses, which to our knowledge has not been done before in this manner. During this process we identified the syntactic restrictions we must impose to relate the Horn clauses to the process they were generated from. Another important note in this regard is the resolution semantics, which we defined to ensure that information in a frame is consistent with the learning capabilies of the hostile environment in the Dolev-Yao threat model. An interesting fact about said Horn clauses arose, stating that the Horn clauses overapproximate all reachable frames of a process, and thus, the Horn clauses can be used to prove syntactic secrecy in the active case. Worthy of noting is that Cortier, Rusinowitch and Zalinescu have proven in [CRZ06] that in some select cases, syntactic secrecy implies strong secrecy.

Finally, we obtain a soundness theorem from the above-mentioned results, thus concluding our intended goals. A biproduct of this, and the above-mentioned, results is that Horn clauses generated from a process can be just as readily applied to a process with broadcast semantics, as to process with point-to-point semantics. This may spark more interest in resolution-based automated verification of broadcasting systems.

8.1 Contribution

We briefly summarise the key results in this thesis as follows.

DAπ_β: A new, very simple, yet very flexible, broadcast calculus with explicit locations, which is a natural extension to well-established work. This calculus is also capable of expressing concepts which, while of great relevance in the verification of security protocols, have been in little focus. For instance, we can model untrusted locations in DAπ_β, and specify explicitly in our network specification that the behaviour of a given network is fully trusted.

Resolution-based automated verification for MANETs: Since verification of security protocols for MANETs is such a new research area, few formalisms and results exist for that purpose. We feel that a new framework for this purpose, along with a proven-sound automated verification technique, is a significant breakthrough in this regard.

Powerful soundness proof: From how easy it was to generalise the soundness proof of the Horn clauses generated from a process specification to different calculi, then we think that this result can easily be adabted by other resarchers interested in proving a similar soundness theorem in their own setting. Also, the only thing this proof needs to work is a labelled reduction semantics, which is present in all process calculi we know of. This is in contrast with Abadi and Blanchet's work [AB02], where a correctness proof is proven in comparison to a general type system.

Horn clauses for syntactic active secrecy: Thanks to our proof that the Horn clauses overapproximate all reachable frames of a process, we now have yet another (proven sound) means of verifying whether a process *ever* leaks some given information.

Horn clauses for broadcasting systems: We argue through our many corollaries that deduction from Horn clauses can just as easily be applied to broadcasting systems as it can point-to-point systems.

Revelation semantics: In this semantics, we capture an *important* part of the learning capabilities of the hostile environment in the Dolev-Yao threat model, which, from what we have seen, has been largely overlooked to date. The semantics are defined in terms of labelled reduction and evaluation contexts, making it convenient for others to adabt this semantics extension to their work.

8.2 Future Work

There are several, very interesting, directions to which this work can be taken. We shall summarise a few of them here.

Authenticity in DAπ_β: While the definitions can surely be derived from Aπ, and techniques such as the ones presented by Blanchet in [Bla02] can be applied to make authenticity easily computable, we have not paid much attention to authenticity, let alone attempted at verifying authenticity properties of protocols. We do, however, have some ideas in this regard, and we shall now give an impression. By extend the process syntax of DAπ with

$$\begin{aligned} P ::= &\ \mathbf{begin}(T).P \\ \mid &\ \mathbf{end}(T).P \\ \mid &\ \mathbf{begin\_ex}(T) \\ \mid &\ \mathbf{end\_ex}(T), \end{aligned}$$

the predicates with

$$\begin{aligned} F ::= &\ \mathsf{begin}(p, \rho) \\ \mid &\ \mathsf{end}(p, s), \end{aligned}$$

and the Horn clause generator in Table 5.10 with

$$\begin{aligned} [\![\mathbf{begin}(T).P]\!]_{\rho h} &= [\![P]\!]_{\rho h \wedge \mathsf{begin}(\rho(T),\rho)} \cup \{h \implies \mathsf{begin}(\rho(T), \rho)\} \\ [\![\mathbf{end}(T).P]\!]_{\rho h} &= [\![P]\!]_{\rho h} \cup \{h \implies \mathsf{end}(\rho(T), \rho(\mathcal{V}_S))\}, \end{aligned}$$

we claim the following to follow from our soundness result.

Corollary 8.1 (Soundness of H' w.r.t. Noninjective Agreement [Bla02]) *If for any derivation of* $\mathsf{H}(A) \vdash \mathsf{end}(T, s)$*, (for any* $\mathsf{end}(T, s)$*),* $\mathsf{begin}(T, \rho)$ *is in the backwards proof path of the derivation, then A satisfies non-injective agreement.* □

While computation based on this result is hardly feasible, we expect that we can prove a result similar to that in Section 3.2.4 [Bla02].

Universal Algebra: While there are many interesting algebras we can express by use of term rewrite systems, there exist algebras which cannot. To obtain utmost generality, it would be interesting to investigate the impact of using universal algebrae as a data language, like in Aπ.

Bisimilarity: We have neglected the treatment of network equivalence, which can indeed be of great use for verifying properties of MANETs. An interesting direction would be to obtain a definition of observational equivalence, which coincides with labelled bisimilarity, thus enabling us to apply the result by Cortier, Rusinowitch and Zälinescu to reason about strong secrecy in the active case.

Generalising the soundness proof: As of now, the proof of soundness of deduction from Horn clauses is specialised to particular predicates, namely message passing. For instance, during our work, we found the need to extend the Horn clause generator with graph-conditioning clauses. Generalising our result to any, or a particular class, of predicates would thus make our result much more applicable.

Encoding DAπ to Aπ with broadcast: To place DAπ_β into "the hierarchy" of process calculi, relating it to existing work is of great help to the scientific community. This would also help us understand exactly *how* expressive DAπ_β really is.

Logic for Local Knowledge: Since we have explicit locations to $\mathrm{DA}\pi_\beta$, it might be of interest to examine the knowledge present at specific locations in a network specification. Furthermore, the concept of local knowledge could also be used to analyse *which location* leaks a particular message, in contast to $\mathrm{A}\pi$ where we only see if the *whole system* leaks information. Applying the logic for $\mathrm{A}\pi$ [Ped06, HP07] might thus be an interesting study.

Implementation: In this thesis, we have supplied the theoretical foundation for an automated verification tool for MANET protocols. To make use of this result, an implementation of it should most definitely be made. A natural choice is, of course, Blanchet's ProVerif, but another very interesting choice is the Succinct Solver, which, by imposing some limitations on predicate logic to obtain the ALFP logic obtains good performance and termination properties. Investigating whether the Horn clauses can be modelled in ALFP would most certainly prove useful.

Bibliography

[AB02] Martín Abadi and Bruno Blanchet. Analyzing Security Protocols with Secrecy Types and Logic Programs. In *29th Annual ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages (POPL 2002)*, pages 33–44, Portland, Oregon, January 2002. ACM Press.

[Aba99] Martín Abadi. Secrecy by typing in security protocols. *Journal of the ACM*, 46(5):749–786, 1999.

[AC04] Martín Abadi and Véronique Cortier. Deciding knowledge in security protocols under equational theories. In *Proc. 31st Int. Coll. Automata, Languages, and Programming (ICALP'2004)*, volume 3142 of *Lecture Notes in Computer Science*, pages 46–58. Springer, 2004.

[AC05] Martin Abadi and Veronique Cortier. Deciding Knowledge in Security Protocols under (Many More) Equational Theories. In *CSFW '05: Proceedings of the 18th IEEE workshop on Computer Security Foundations*, pages 62–76, Washington, DC, USA, 2005. IEEE Computer Society.

[AF01] Martín Abadi and Cédric Fournet. Mobile Values, New Names, and Secure Communication. *28th ACM Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115, 2001.

[AG97] Martín Abadi and Andrew D. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. In *Fourth ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press, 1997.

[AILS07] Luca Aceto, Anna Ingólfsdóttir, Kim Guldstrand Larsen, and Jiri Srba. *Reactive Systems: Modelling, Specification and Verification.* Cambridge University Press, New York, NY, USA, 2007.

[AT91] Martín Abadi and Mark R. Tuttle. A semantics for a logic of authentication (extended abstract). In *PODC '91: Proceedings of the tenth annual ACM symposium on Principles of distributed computing*, pages 201–216, New York, NY, USA, 1991. ACM.

[BAF08] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 75(1):3–51, February–March 2008.

[BAN89] Michael Burrows, Martin Abadi, and Roger Needham. A Logic of Authentication. Technical Report 39, Digital Equipment Corporation, Systems Research Centre, February 1989.

[BDNN01] Chiara Bodei, Pierpaolo Degano, Flemming Nielson, and Hanne Riis Nielson. Static analysis for the pi-calculus with application to security. *INFCTRL: Information and Computation (formerly Information and Control)*, 168, 2001.

[Bla01] Bruno Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, pages 82–96, Cape Breton, Nova Scotia, Canada, June 2001. IEEE Computer Society.

[Bla02] Bruno Blanchet. From secrecy to authenticity in security protocols. In *SAS '02: Proceedings of the 9th International Symposium on Static Analysis*, pages 342–359, London, UK, 2002. Springer-Verlag.

[Bla08] Bruno Blanchet. Automatic verification of correspondences for security protocols, 2008.

[BM97] Annette Bleeker and Lambert Meertens. A Semantics for BAN Logic. In *Proc. DIMACS Workshop on Design and Formal Verification of Security Protocols*, New Brunswick, NJ, 1997.

[BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and All That.* Cambridge University Press, New York, NY, USA, 1998.

[BP05] Bruno Blanchet and Andreas Podelski. Verification of Cryptographic Protocols: Tagging Enforces Termination. *Theoretical Computer Science*, 333(1-2):67–90, March 2005. Special issue FoSSaCS'03.

[BPV05] Michael Baldamus, Joachim Parrow, and Björn Victor. A Fully Abstract Encoding of the π-Calculus with Data Terms. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *ICALP*, volume 3580 of *Lecture Notes in Computer Science*, pages 1202–1213. Springer, 2005.

[Bur98] Stanley N. Burris. *Logic for Mathematics and Computer Science.* Prentice Hall, Upper Saddle River, New Jersey 07458, 1998.

[CM02] M. Carbone and S. Maffeis. On the expressive power of polyadic synchronisation in pi-calculus, 2002.

[CRZ06] Véronique Cortier, Michaël Rusinowitch, and Eugen Zalinescu. Relating two standard notions of secrecy. In *CSL*, pages 303–318, 2006.

[CS01] Tom Chothia and Ian Stark. A Distributed Pi-Calculus with Local Areas of Communication. In John Reppy and Peter Sewell, editors, *Electronic Notes in Theoretical Computer Science*, volume 41. Elsevier Science Publishers, 2001.

[dB72] Nicolaas Govert de Bruijn. Lambda-Calculus Notation with Nameless Dummies: a Tool for Automatic Formula Manipulation with Application to the Church-Rosser Theorem. *Indagationes Mathematicae*, 34(5):381–392, 1972.

[Die06] Reinhard Diestel. *Graph Theory.* Springer-Verlag, Berlin and Heidelberg, Germany, 2006.

[DY81] Danny Dolev and Andrew C. Yao. On The Security of Public Key Protocols. Technical report, Department of Computer Science, Stanford University, Stanford, CA, USA, 1981.

[EM99] Christian Ene and Traian Muntian. Expressiveness of Broadcast Communication. In *Proceedings of FCT99*, Springer LNCS, pages 258–268, 1999.

[Fer04] Maribel Fernández. *Programming Languages and Operational Semantics.* King's College Publications, 2004.

[GJ01] A. Gordon and A. Jeffrey. Authenticity by typing for security protocols, 2001.

[God06] Jens Chr. Godskesen. Formal Verification of the ARAN Protocol Using the Applied Pi-calculus. In *Proceedings of the Sixth International IFIP WG 1.7 Workshop on Issues in the Theory of Security*, pages 99–113, 2006.

[God07] Jens Chr. Godskesen. A Calculus for Mobile Wireless Networks. Technical Report 39, ITU, Denmark, May 2007.

[HM05] Tony Hoare and Robin Milner. Grand Challenges for Computing Research. *Comput. J.*, 48(1):49–52, 2005.

[HP07] Hans Hüttel and Michael D. Pedersen. A logical characterisation of static equivalence. *Electron. Notes Theor. Comput. Sci.*, 173:139–157, 2007.

[HR98] Matthew Hennessy and James Riely. Resource Access Control in Systems of Mobile Agents. In *Proceedings of HLCL'98*, volume 16(3) of *Electronic Notes in Theoretical Computer Science.* Elsevier, 1998.

[HT91] Kohei Honda and Mario Tokoro. An Object Calculus for Asynchronous Communication. *Lecture Notes in Computer Science*, 512:133–148, 1991.

[Hü05] Hans Hüttel. *Pilen ved træets rod.* Aalborg Universitet, Aalborg, Denmark, 2005.

[Joh96] P. T. Johnstone. *Notes on Logic and Set Theory.* Cambridge Mathematical Textbooks. Cambridge University Press, 1996.

[Lau05] Niels Lauritzen. *Concrete Abstract Algebra.* Cambridge University Press, 2005.

[Low95] Gavin Lowe. An Attack on the Needham-Schroeder Public-Key Authentication Protocol. *Information Processing Letters*, 56(3):131–133, 1995.

[Mil82] Robin Milner. *A Calculus of Communicating Systems.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.

[Mil93] R. Milner. The Polyadic Pi-calculus: A Tutorial. In F. L. Bauer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*, pages 203–246. Springer-Verlag, 1993.

[Mil99] Robin Milner. *Communicating and Mobile Systems: The π-Calculus.* Cambridge University Press, 1999.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A Calculus of Mobile Processes, I and II. *Inf. Comput.*, 100(1):1–77, 1992.

[Nan06] Sebastian Nanz. *Specification and Security Analysis of Mobile Ad-Hoc Networks.* PhD thesis, Imperial College London, 2006.

[NH04] Sebastian Nanz and Chris Hankin. Static Analysis of Routing Protocols for Ad-Hoc Networks. In *Proceedings of the 2004 ACM SIGPLAN and IFIP WG 1.7 Workshop on Issues in the Theory of Security (WITS'04)*, pages 141–152, 2004.

[NH06] Sebastian Nanz and Chris Hankin. A Framework for Security Analysis of Mobile Wireless Networks. *Electronic Notes in Theoretical Computer Science*, 367:207–227, 2006.

[NNS$^+$04] F. Nielson, H. Riis Nielson, H. Sun, M. Buchholtz, R. R. Hansen, H. Pilegaard, and H. Seidl. The succinct solver suite. In Andreas Podelski Kurt Jensen, editor, *Proc. TACAS'04*, volume 2988 of *Lecture Notes in Computer Science*, pages 251–265. Springer-Verlag, 2004.

[Ped06] Michael David Pedersen. Logics for The Applied Pi Calculus. Technical report, BRICS, Denmark, 2006.

[Pie02] Benjamin C. Pierce. *Types and Programming Languages.* MIT Press, 2002.

[Pra95] K. V. S. Prasad. A Calculus of Broadcasting Systems. *Electronic Notes in Theoretical Computer Science*, 25:285–327, 1995.

[Shi88] O. Shivers. Control flow analysis in scheme. In *PLDI '88: Proceedings of the ACM SIGPLAN 1988 conference on Programming Language design and Implementation*, pages 164–174, New York, NY, USA, 1988. ACM.

[SS94] Leon Sterling and Ehud Shapiro. *The art of Prolog (2nd ed.): advanced programming techniques.* MIT Press, Cambridge, MA, USA, 1994.

[SW01] Davide Sangiorgi and David Walker. *PI-Calculus: A Theory of Mobile Processes.* Cambridge University Press, 2001.

[Syv91] Paul Syversen. The Use of Logic in the Analysis of Cryptographic Protocols. In *1991 IEEE Symposium on Security and Privacy*, page 156, Oakland, CA., 1991. IEEE Computer Society Press.

[WK96] Gabriele Wedel and Volker Kessler. Formal semantics for authentication logics. In *ESORICS '96: Proceedings of the 4th European Symposium on Research in Computer Security*, pages 219–241, London, UK, 1996. Springer-Verlag.