Equivalence Properties and Probabilistic Reasoning in Symbolic Security Protocol Analysis

Bruno Conchinha Montalto
MSc. Mathematics and Applications, IST Lisbon
Born November 4th, 1985
Citizen of Portugal

June 27th, 2014

Abstract

Security protocols are distributed programs designed to ensure secure communication in a network controlled by an adversary. They are widely used today for securing on-line services such as personal communication and electronic voting. Their design is notoriously error-prone, and a great deal of research has been devoted to their analysis. In this thesis we provide two main contributions towards the automated analysis of such protocols: algorithms for the symbolic analysis of equivalence properties, and a symbolic probabilistic framework for security protocol analysis.

We consider the problem of verifying two equivalence properties relevant in protocol analysis: static equivalence and trace equivalence. Both notions model the property that an attacker cannot distinguish between two protocol executions, and they have been used to model security goals such as off-line guessing, electronic voting anonymity, and RFID untraceability. Static equivalence is concerned with an attacker who passively eavesdrop on a network and then tries to distinguish between possible executions by performing off-line computations. Trace equivalence is used in the analysis of security properties against an attacker who may participate actively in protocol execution.

We present an efficient decision procedure for static equivalence under equational theories generated by subterm convergent rewriting systems. This class of theories encompasses the most common cryptographic primitives, including symmetric and asymmetric encryption and decryption, hash functions and digital signatures. Our algorithm achieves a better asymptotic complexity than competing algorithms, albeit with a narrower scope. We discuss its implementation in the FAST tool and show that it indeed performs much more efficiently than other existing tools for the same task.

We also present a procedure for deciding the trace equivalence of bounded simple processes under equational theories generated by convergent rewriting systems and for which a finitary unification algorithm exists. Although we do not have a termination result, our procedure is correct for the largest class of equational theories handled by any existing procedure for deciding trace equivalence and, together with the work by Cheval et. al [64], it is the only one to handle non-trivial `else` branches.

Finally, we introduce a symbolic probabilistic framework for the analysis of security protocols. Our framework provides a general method for expressing weak-

nesses of cryptographic primitives. These weaknesses are represented as *property statements* relating subsets of the domain with subsets of the range of a cryptographic primitive. We show that the class of property statements that we consider is sufficient to express, for example, weaknesses in random generation algorithms or partial information leakage, as might occur, for example, when using a length-revealing encryption system or in the presence of an adversary using differential cryptanalysis. Therefore, the attacker in our model is considerably more powerful than conventional symbolic attackers. These properties can be used within our framework to automatically find attacks and estimate their success probability. Existing symbolic methods can neither model such properties nor find such attacks. We show that the probability estimates obtained by using our framework are negligibly different from those yielded by a generalized random oracle model based on sampling terms into bitstrings while respecting the stipulated properties of cryptographic primitives. As case studies, we use a prototype implementation of our framework to model non-trivial properties of RSA encryption and automatically estimate the probability of off-line guessing attacks on the EKE protocol.

Riassunto

I protocolli di sicurezza sono programmi distribuiti ideati per assicurare comunicazioni sicure in una rete sotto il controllo di un avversario. Oggigiorno sono utilizzati largamente per proteggere servizi on-line quali corrispondenza personale e votazione elettronica. La loro progettazione è notoriamente propensa ad errori, e una buona parte della ricerca è stata votata alla loro analisi. In questa tesi forniamo due contributi principali verso l'analisi automatica di tali protocolli: algoritmi per l'analisi simbolica di proprietà d'equivalenza, e un framework per l'analisi di protocolli di sicurezza.

Consideriamo il problema di verificare due proprietà di equivalenza rilevanti nell'analisi di protocolli: equivalenza statica ed equivalenza delle tracce. Entrambe le nozioni modellano la proprietà che un attaccante non possa distinguere un'esecuzione del protocollo da un'altra, e sono state impiegate per modellare obiettivi di sicurezza quali off-line guessing, anonimità nelle votazioni elettroniche, e intracciabilità degli RFID. L'equivalenza statica studia un attaccante che intercetta passivamente comunicazioni da una rete e tenta di distinguere tra differenti esecuzioni attraverso calcoli off-line. L'equivalenza delle tracce viene utilizzata per l'analisi di proprietà di sicurezza contro un attaccante che possa partecipare attivamente all'esecuzione del protocollo.

Presentiamo una procedura decisionale efficiente per l'equivalenza statica subordinata a teorie equazionali generate da sistemi convergenti di riscrittura in sottotermini. Questa classe di teorie include le primitive crittografiche più comuni, tra cui cifratura e decifratura simmetrica e asimmetrica, funzioni di hash e firme digitali. Il nostro algoritmo raggiunge una complessità asintotica migliore dei competitori, sebbene con una portata inferiore. Ne discutiamo l'implementazione nel tool FAST e mostriamo che le sue prestazioni sono molto più efficienti di altri tool preesistenti per lo stesso compito.

Presentiamo anche una procedura per decidere l'equivalenza delle tracce di processi semplici limitati subordinata a teorie equazionali generate da sistemi convergenti di riscrittura e per i quali esiste un algoritmo di unificazione che produca risultati di dimensione finita. Sebbene non disponiamo di una dimostrazione di terminazione, la procedura risulta corretta per la più larga classe di teorie equazionali supportata da qualsiasi procedura preesistente per decidere l'equivalenza delle tracce, e, assieme al lavoro di Cheval et. al [64], è l'unica a supportare ramificazioni `else` non banali.

Infine, presentiamo un framework probabilistico e simbolico per l'analisi di protocolli di sicurezza. Il nostro framework fornisce un metodo generale per esprimere debolezze delle primitive crittografiche. Queste debolezze sono rappresentate come affermazione di proprietà che mettono in relazione sottoinsiemi del dominio con sottoinsiemi del codominio di una primitiva crittografica. Mostriamo che la classe di affermazioni di proprietà è sufficiente ad esprimere, per esempio, debolezze in algoritmi di generazione di valori casuali o rivelazione parziale di informazione, come si può verificare, per esempio, nell'utilizzo di un sistema crittografico che riveli la lunghezza dell'input o in presenza di un avversario che effettui criptoanalisi differenziale. Di conseguenza, l'attaccante nel nostro modello è considerevolmente più forte degli attaccanti simbolici convenzionali. Queste proprietà possono essere utilizzate all'interno del nostro framework per ricercare automaticamente attacchi e stimare la loro probabilità di successo. Metodi simbolici preesistenti non sono in grado né di modellare tali proprietà né di rilevare tali attacchi. Mostriamo che le stime di probabilità ottenute utilizzando il nostro framework differiscono in misura trascurabile da quelle prodotte da un modello di oracolo casuale basato sul campionamento di termini in stringhe di bit che rispettino le proprietà delle primitive crittografiche stipulate. Come casi di studio, impieghiamo un'implementazione prototipale del nostro framework per modellare proprietà non banali della cifratura RSA e per stimare automaticamente la probabilità di attacchi di guessing off-line sul protocollo EKE.

Acknowledgments

This thesis would not have been possible without the professional and personal support of a number of people, to whom I would like to express my gratitude.

First of all, I must thank my supervisors David Basin and Carlos Caleiro for many helpful, stimulating and fun discussions, that have ranged from the low-level technical details of my work to the high level questions of "Where is my thesis going?" and "What do I do with my life?". Just as importantly, I must also thank them for being endlessly supportive and believing in my work even through my least productive periods. I would also like to thank Mohammad Torabi Dashti, Simone Frau, Benedikt Schmidt, Steve Kremer, Jannik Dreier and Ralf Sasse for helpful technical discussions and for reviewing my drafts.

At a personal level, I would like to thank Simone Frau, for his unwavering affection, indefatigable support and multi-flavored hugs, and to Mohammad Torabi Dashti, for all our surreal late-night meta-philosophical discussions. Simone and Mohammad have kept me sane and laughing through my regular existential crises, that I have had more often than anyone would have liked to put up with. An especial word of thanks is due to Ramona Sorecau, for her support during the last stage of my Ph.D. and for making me happy with her very own unique brand of cute and adorable. Wow!

For the great time we have spent together and for keeping happy (and sane!) throughout my Ph.D., I would also like to leave a word of thanks to my friends David Henriques, Nuno Freitas, Lus Sousa, Lídia del Rio, Marc Lafrance, Luigi Tortola, Giuseppina Fascellaro, Erik Fumi and Prannoy Suraneni.

Most importantly, I want to thank my family, particularly my mother Faustina Conchinha, my sister Vanessa Montalto and my grandparents Estevão Conchinha and Catarina Silva. They have been a constant presence in my life, and their unconditional love and support have given me the confidence and affection that I needed throughout all the stages of my education. To them I extend my deepest gratitude and affection.

Contents

Chapter 1

Introduction

Security protocols are small distributed programs designed to provide guarantees such as authenticated, confidential or anonymous communication even when executed in an hostile environment, typically by making use of cryptography. Such protocols are often executed in networks where an attacker is able to impersonate users as well as read, block, modify or redirect sent messages. Due to the broad availability and usage of security-critical applications on the Internet, the deployment of security protocols has become critical to ensure that functionalities such as private communication, electronic commerce, and electronic voting can be safely realized. It is therefore crucial to obtain as much confidence as possible in the correctness of such protocols.

Symbolic methods for protocol analysis have contributed greatly to this goal. Such methods are usually based on the *Dolev-Yao model*, first introduced by Dolev and Yao in their seminal paper [98]. In this model, freshly generated data or publicly known constants are represented by atomic symbols in a term algebra, with function symbols representing the functions (cryptographic or not) used by the agents. Properties of cryptographic operators, such as the fact that one can obtain a secret message if one knows its encryption and the (symmetric) key used in the encryption, are represented either as deduction systems (e.g. [34,88]) or as equational theories (see [83] for a broad survey).

In such methods, cryptography is assumed to be perfect: each message and secret is either known or not known, and the attacker cannot learn anything about a plaintext by examining its encryption unless he also knows the corresponding decryption key. By abstracting cryptographic details, these methods have made verification amenable to automation and have thus uncovered protocol attacks that were overlooked by manual human analysis [32, 100]. Perhaps the most well-known example is Lowe's attack [121] on the Needham-Schroeder protocol proposed nearly twenty years earlier [130]. Such tools have also been used to prove security properties, e.g. [14, 38, 132].

Today, symbolic methods exist and have been automated and used for analyzing different security properties, for bounded and unbounded protocol execution,

and considering a broad range of cryptographic primitives [16, 34, 45, 102, 136]. However, symbolic methods often do not accurately represent an attacker's capabilities, as the modeling of cryptographic primitives is greatly simplified. As a consequence, broad classes of attacks that rely on weaknesses of cryptographic primitives fall outside the scope of such methods [40, 72, 143].

In contrast, computational methods aim to prove security of cryptographic protocols and primitives by reasoning directly about bitstrings. Security guarantees in computational models are typically a consequence of (often rather strong) assumptions on the security of the cryptographic primitives being used, such as indistinguishability under chosen ciphertext attacks [108] or key-dependent message security [44, 60]. These properties are usually formulated as *games* where a polynomial-time attacker, possibly with access to some computation oracles, tries to get some information that cryptographic primitives are designed to conceal. The standard technique in computational security protocol analysis is to show that if a protocol can be attacked, then the attacker can win one of these games against some of the cryptographic primitives involved, thereby reducing a protocol's security to the security of the cryptographic primitives that is employs.

A vast body of research has been devoted to proving security properties of cryptographic protocols in computational settings, e.g. [39,59,108,115]. [41] uses these methods to prove the (computational) security of the TLS protocol, one of the most significant successes of such approaches. Computational methods consider a much more realistic and powerful attacker, and thus yield much stronger security guarantees. However, such methods are usually hard to automate, and long, error-prone, hand-written proofs are usually required to establish the security of given protocols using specific cryptographic primitives.

The contributions of this thesis comprise algorithms for deciding equivalence properties in the symbolic model, as well as an automatable, symbolic framework for strengthening the security guarantees that can be provided by automated methods. In Section 1.1 we describe existing procedures for the analysis of equivalence properties in symbolic settings. Section 1.2 gives an overview of existing approaches to security protocol analysis that aim to provide strong security guarantees in an automatable or machine-checkable manner. We describe our contributions in greater detail in Section 1.3.

1.1 Equivalence Properties in Symbolic Models

Most existing tools and methods using the symbolic approach to protocol security are tailored for the analysis of *trace properties*, which encompass central security goals such as confidentiality and authentication [16,35,136]. However, many other relevant security properties are best expressed as *equivalence properties*. These include the security against off-line guessing attacks [4, 36, 79], strong secrecy [46] (an attacker cannot distinguish between the intended secret and a randomly generated value), electronic voting anonymity (an attacker cannot distinguish two vote

assignments that differ by swapping two voters' votes [93]) and RFID untraceability (the attacker cannot tell whether two transactions were made by the same RFID tag [13]).

1.1.1 The Applied-Pi Calculus

The *applied-pi calculus*, first introduced in [6], is a process algebra designed for the analysis of security protocols. It extends the original *pi calculus* [126] by introducing equational theories, which are necessary for its purpose of analyzing security protocols. It is more general than the similar spi calculus [7] because it allows arbitrary equational theories, in contrast with the fixed set of cryptographic primitives modeled by the spi calculus. In the applied-pi calculus, protocol participants are represented as *processes*, passing messages to each other via *channels*. Protocols are represented by the parallel composition of processes representing its participants. In this fashion it is possible to model the most important aspects of security protocols and the hostile environment in which they are designed to be executed: the possibility of each agent executing several sessions of possibly more than one security protocol by sending and receiving messages over communication channels, without *a priori* guarantees concerning the identity of its communication partners. In this model, the attacker is often represented as a context, able to send and read messages sent over channels that are not explicitly secret. In this fashion an attacker is able to redirect messages from one channel to another, a mechanism by which it may impersonate agents, prevent messages from being delivered or replace them with messages of its own choosing.

The applied-pi calculus is one of the most successful formalisms for symbolic security protocol analysis: Many important security notions have been defined and analyzed using the applied-pi calculus [13, 14, 36, 72, 93, 117], and the calculus itself has been the subject of significant research [15, 92, 99]. In particular, both notions of equivalence studied in this thesis, static equivalence and trace equivalence, have been formalized using the applied-pi calculus, and our symbolic probabilistic framework uses the notion of applied-pi calculus frames to represent an attacker's knowledge.

1.1.2 Static Equivalence

Static equivalence is used to model the notion that an attacker cannot distinguish between two sequences of messages. Its main limitation is that it is a static property, i.e., it only considers a fixed protocol execution without taking into account the communication between the agents and how an attacker may be able to interfere with it.

The first paper to study the decidability of deduction and static equivalence under general and abstract classes of equational theories was [5]. ProVerif [45] can be used to decide static equivalence and even use it to verify equivalence properties in a dynamic setting, taking into account attacker inputs to the network [49].

More recently, the YAPA [37] and KISS [69] tools implement efficient and general decision procedures for both problems. Other existing decidability results cover monoidal theories and other AC (associative-commutative) equational theories, including theories representing abelian groups [5, 82].

1.1.3 Trace Equivalence

Trace equivalence is another equivalence notion. It intuitively represents the property that, given the output of two execution traces generated by one of two distinct processes, the attacker cannot tell which of the two processes was used to generate the given trace. Trace equivalence has been used to model and reason about several important security properties relevant in the analysis of a broad range of protocols. Such security properties include unlinkability and anonymity [13, 68] (crucial, for example, in the analysis of RFID protocols [55,141]), strong notions of secrecy [46], resistance to off-line guessing attacks [36], and voting secrecy [93]). Hüttel shows that trace equivalence in the spi calculus is decidable for bounded processes and undecidable in the unbounded case [112].

In [64] it is shown that the problem of deciding trace equivalence in the applied-pi calculus can be reduced to the problem of deciding equivalence between sets of constraint systems. Constraint systems are a widely known technique in symbolic security protocol analysis, and techniques for solving constraint systems under a broad range of equational theories have been given, e.g., in [56, 58, 66, 70, 80, 94, 125].

Indeed, most existing procedures for deciding trace equivalence rely on constraint solving techniques: [36] and [67] provide decision procedures for the problem of trace equivalence of bounded, simple processes with trivial `else` branches under subterm convergent equational theories. [64, 65] provides another decision procedure which considers a more restricted set of cryptographic primitives (symmetric and asymmetric encryption, hashing, pairing, and digital signatures), but a broader class of processes that need not be simple and may have non-trivial `else` branches. Non-trivial `else` branches allow reasoning about protocols in which a participant may test two terms for equality and take some action other than simply aborting in case the two are not equal, such as the e-passport protocol discussed in [13]. A recent implementation of this procedure is described in [62]. Another decision procedure is given in [91] for simple processes with trivial `else` branches and group theories, namely XOR and Abelian groups, with an additional unary homomorphic function symbol.

AKISS is a tool that can be used to decide trace equivalence of bounded processes with trivial `else` branches under optimally-reducing equational theories [61]. It is different from other existing algorithms in that it uses a resolution-based technique, as opposed to the constraint solving technique of other approaches. Termination is conjectured but not proved for subterm convergent theories.

1.1.4 Other Equivalence Properties in Security Protocol Analysis

Several other useful notions of process equivalence exist for the applied-pi calculus [49, 81, 92]. An excellent summary is provided in [65]. Testing equivalence [7] is closely related to trace equivalence: it is implied by it, and coincides with it for *image-finite* processes, such as processes without replication [65].

In the context of the spi calculus, [101] and [139] present tools for verifying testing equivalence and open bisimulation, respectively. Note that in the spi calculus only a fixed set of common cryptographic primitives is considered. Since [49], ProVerif has been able to verify diff-equivalence, an equivalence notion stronger than observational equivalence, and [63] extends the scope of equivalence notions handled by this tool. [81] shows that observational equivalence and trace equivalence coincide for a class of processes called *determinate* processes and gives a procedure for deciding the observational equivalence of (bounded) simple processes with trivial `else` branches.

1.2 Strengthening Symbolic Models

Much research has been devoted to bridging the gap between symbolic and computational models. Such works aim to combine the automatability of symbolic methods with the strong security guarantees provided by computational approaches, as discussed in the introduction to this section. [48] and [85] provide excellent surveys of such works. We can broadly classify these approaches into three main directions: computational soundness results, automating computational security proofs, and the more recent computationally complete symbolic attacker framework. The state of the art in each of these approaches is discussed in greater detail in the following sections.

1.2.1 Computational Soundness Results

Computational soundness results aim to find security properties of cryptographic primitives that are sufficiently strong to ensure that symbolic security (easier to verify) implies computational security (much stronger).

The first such result was presented in Abadi and Rogaway's seminal paper [9]. Such results now exist using both trace mapping approaches [124] and the simulatability framework [25], for a variety of equational theories, and considering different security properties. Results now exist for passive and active adversaries under symmetric [9,22] and public-key encryption [24,111,124], hash functions [106], modular exponentiation [54], bilinear pairings [116], zero-knowledge proofs [18], and non-malleable commitment [104]. Soundness results for equivalence properties also exist for symmetric and public-key encryption, hash functions, and digital signatures [71, 84, 86, 113]. Significant effort has also been devoted to weakening the assumptions on the cryptographic primitives, e.g. by considering length-revealing encryption systems [11] or key-dependent message security [23, 52], and

to broadening the class of protocols covered by such results [20], e.g. by analyzing key exchange protocols [89, 118]. More recently, [21] offers a soundness result for ProVerif [45].

The limitations of these results include the very strong assumptions on the security of cryptographic primitives, the requirement that messages are tagged so that their structure is known to any observer, and the difficulty of extending the results to new cryptographic primitives, which usually involves re-doing most of the work. More recent papers aim to solve some of these limitations, e.g. by establishing composability results for the computational soundness of a class of primitives [51].

1.2.2 Automated Computational Security Proofs

Another line of research is to obtain computational security proofs in an automated, machine-assisted, or machine-checkable manner. This is typically done by a sequence of transformations between different formulations of security properties. The technique is to ensure that each of these transformations preserves the desired security properties up to a negligible probability, and to reach a final formulation which is trivially equivalent to the desired security property. In this way computational security properties may be proved automatically, and such methods may sometimes provide upper bounds on the probability of an attack.

The original ideas in this direction were first presented in [19, 50, 110]. Today various tools and frameworks exist for this task, such as CryptoVerif [47], CertiCrypt [31] and EasyCrypt [29]. [30] introduces a logic for reasoning about cryptographic primitives in such models, and [87] uses automated methods to reason about the computational security of asymmetric encryption. Typically termination is not guaranteed in such approaches, and lack of a proof does not necessarily entail the existence of an attack (i.e., these methods may output "I don't know" [48]). However, attacks may often be obtained by the human inspection of a failed security proof.

1.2.3 The Computationally Complete Symbolic Attacker

The computationally complete symbolic attacker framework is a more recent research direction originally proposed in [27]. Rather than using a symbolic representation of the actions which an attacker *may* take, this framework relies on a symbolic representation of the actions that an attacker *may not* take as a consequence of the security properties assumed for the primitives considered. Security properties may thus be modeled as reachability properties, as is the case in traditional symbolic methods. Moreover, such a framework may provide security results with much weaker assumptions than those required by computational soundness results: For example, dynamic corruption and key-cycles are allowed, and bitstrings are not assumed to be tagged, i.e., unambiguous parsing of bitstrings is not required [28]. This framework has been used for a complete analysis of a security protocol for

the first time in [26]. [73] shows that satisfiability can be decided in polynomial time for the class of clauses considered in this framework, thereby showing that this security protocol analysis technique is amenable to automated reasoning.

1.3 Contributions

In this thesis we give two main contributions towards the automation of security protocol analysis. First, we present new algorithms for deciding static equivalence and trace equivalence. Second, we introduce a new framework for the automated, symbolic probabilistic analysis of security protocols. Within this framework it is possible to express security relevant properties of cryptographic primitives that can be exploited by an attacker and that are out of the scope of existing symbolic methods, thereby strengthening the security guarantees that can be automatically obtained. We now discuss each of these contributions in greater detail.

1.3.1 The FAST Algorithm and Tool

The FAST algorithm and tool is an efficient decision procedure for static equivalence under subterm convergent equational theories. It takes advantage of a DAG-representation of terms (as in [5, 69]) and of several properties of subterm convergent equational theories to significantly optimize the saturation procedure in which most existing techniques for deciding static equivalence rely. In particular, the algorithm does not rely in unification in any way, instead performing a form of matching which is sufficient for reasoning about these equational theories.

As a result of these optimizations, FAST achieves a significantly better asymptotic complexity than other tools for the same task [37, 69], albeit with a narrower scope. The practical results show that FAST is also significantly faster in practice: for many examples, FAST terminates in under a second, whereas other tools take several minutes.

The theoretical algorithm has first been published in [76]. Its implementation as the FAST tool is described, and its performance analyzed, in [77]. Both are joint work with David Basin and Carlos Caleiro.

1.3.2 Symbolic Verification of Equivalence Properties

We present a constraint solving procedure for deciding the equivalence of constraint systems under equational theories generated by convergent rewriting systems for which a finitary unification algorithm exists. This procedure supports constraint systems with negative constraints but not sets of constraint systems. Relying on the results of [65], we show that such a procedure can be used to decide the trace equivalence of bounded simple processes, a class of processes first defined in [81] which is expressive enough to model most security protocols. Our procedure uses a few techniques which we believe can be used for symbolic reasoning in general, and especially in constraint solving.

Generalized frames. First, we introduce the notion of *generalized frame*. As is usual, we consider a set of *recipes* that represent ways in which messages can be built, by using function symbols, publicly known names, and the terms in the range of the frame. However, our notion of generalized frame differs from traditional applied-pi calculus frames because terms in such a frame may contain *recipe variables*, i.e., variables which are instantiated by recipes rather than by terms. This idea is not entirely novel: It has long been used in constraint systems (somewhat implicitly), expressed as the requirement that variables representing adversary inputs must be deducible given the attacker's knowledge when the message is sent. Moreover, the *deduction facts* used in the KISS algorithm [69] use a similar idea: intuitively, these facts may be used to express that a term with a hole is deducible as long as the hole is filled with a deducible term. Note that these two uses are in fact distinct: Variables in constraint solving algorithms must always be instantiated with the same recipe, corresponding to the recipe used by the attacker to compute it before sending it over to the network; in contrast, variables occurring in deduction facts are used to finitely represent an attacker's knowledge in the presence of certain equational theories (such as blind signatures or trap-door commitment), and they may be instantiated with different recipes. We allow these recipe variables in the frame representing the attacker's knowledge, with the constraints imposed on variables corresponding to attacker inputs being captured by means of our deducibility constraint system. In this way, a single uniform technique may be used to reason about the complex equational theories considered and the deducibility constraint system to be solved.

Φ-unification. Related to the first technique, we introduce the notion of *Φ-unification*, i.e., unification with respect to a (generalized) frame. Such a unification problem may contain variables of two types: *term variables* and *recipe variables*, with the former being instantiated with terms, as usual, and the latter with recipes, as described in the paragraph above. Recipes are then mapped to terms by the frame Φ. A solution of such a problem is a pair of substitutions (α, γ), with α being a term substitution as usual and γ a recipe variable substitution. By using these two types of variables we are able to reason about equivalence properties in constraint systems, which require reasoning about the recipes used by the attacker to deduce a message. We show that, despite Φ-unification being syntactical, complete sets of Φ-unifiers do not necessarily exist. We present a sound but possibly non-terminating algorithm for Φ-unification.

D-saturation. Finally, we introduce the notion of *D-saturation*. Frame saturation has been extensively used to reason about static equivalence by finitely and conveniently representing all the terms deducible by an attacker [5, 37, 69, 76]. D-saturation is a generalization of this notion in the presence of deducibility constraints: In such a context, it is possible that an attacker can deduce *either* a term t or a term t' but, because deducing each of them requires sending different messages

over the network at some step of the protocol execution, he is not able to deduce *both*. Therefore, a D-saturation consists of a set of pairs $(\theta, \Theta_F(\theta))$, in which θ is an instantiation of the variables in the constraint system (corresponding to the variables which represent attacker inputs to the network), and $\Theta_F(\theta)$ represents a saturation of the resulting frame, providing a finite representation of the attacker's knowledge when θ represents the messages he has sent during protocol execution. We define D-saturation, analyze its properties, and provide an algorithm for computing them. As our Φ-unification algorithm, our D-saturation algorithm is sound but possibly non-terminating.

Termination of our constraint solving procedure depends only on the termination of the Φ-unification and D-saturation procedures. We conjecture that these procedures always terminate for the class of inputs that may occur when the equational theory considered is subterm convergent. Moreover, we believe that it is possible to generalize our technique so that it can be used for deciding the equivalence of sets of constraint systems. For these reasons, we believe that these notions are of independent interest.

Related work. A brief summary of existing procedures for deciding trace equivalence is given in Section 1.1.3. Our procedure handles a more general set of equational theories and processes than [36, 67], and a more general set of equational theories than [64, 65]. However, [64, 65] handles a wider class of processes (not necessarily simple). Moreover, unlike these two algorithms, no termination result exists for our procedure. Compared to AKISS [61], our algorithm handles a more general set of equational theories and an incomparable set of processes: namely, AKISS handles processes that are not necessarily simple, but does not support non-trivial `else` branches.

In conclusion, our procedure handles the largest class of equational theories among those currently proposed, and it is the only procedure besides [65] to handle non-trivial `else` branches. However, it can only handle deterministic processes, and we have not proven any termination result. Table 1.1 provides a quick overview of related work with respect to the class of equational theories and processes handled, termination guarantees and the existence of an implementation.

Table 1.1: Trace Equivalence Decision Procedures

	Equational Theories	Processes (Bounded)	Termination	Implementation
This	Finitely Unifiable	Deterministic	No	No
[36]	Subterm Convergent	Deterministic, trivial else	Yes	No
[65]	Standard DY	Any	Yes	Yes ([62])
[61]	Optimally Reducing	Trivial else	No	Yes

1.3.3 Symbolic Probabilistic Analysis of Security Protocols

The last main contribution of this thesis is a framework for the symbolic probabilistic analysis of security protocols. This framework represents a fundamentally new

approach to strengthening the security guarantees provided by automated methods, which is in a sense dual to current research that aims to bridge the gap between symbolic and computational models (discussed in Section 1.2): Rather than assuming strong security properties of cryptographic primitives and using them to prove security, we explicitly describe (computational) weaknesses of cryptographic primitives and random number generation algorithms and use them to find attacks. In other words, our approach is designed for attack-finding rather than for security verification.

A case for stronger attack-finding. We believe that there is a strong argument for the usefulness of such an approach. Indeed, security protocols may be flawed by simple implementation details or weaknesses of the underlying cryptographic primitives [40, 143]. Moreover, commonly used cryptographic primitives do not satisfy the security properties required for most computational soundness results: Elgamal encryption [105] does not provide IND-CCA security [143], and admits a so-called *second decryption* function which allows an attacker to learn a plaintext without knowing the (private) decryption key, by using the (public) encryption key and the random coins used in the encryption. Naive RSA [134] implementations do not employ probabilistic encryption, and in any case public keys are easily distinguishable from random bitstrings [40]. Moreover, linear and differential cryptanalysis techniques exist for a variety of commonly used symmetric encryption and hashing schemes [12, 42, 96, 97, 122, 123, 133]. While such schemes may not be completely broken (as is the case of MD5, e.g. [138]), they may nevertheless leak partial information which may be used by an attacker. Similarly, side-channel attacks can be deployed by an attacker to obtain partial knowledge about a secret message [10, 43, 107, 109, 131].

In practice, such cryptographic primitives are often deployed, rendering inapplicable the security guarantees provided by most computational methods and computational soundness results. Therefore, finding attacks that rely on such weaknesses is important, and may lead to changing industry standards more effectively than simply proving the security of a given protocol without showing how deployed protocols (or implementations thereof) may be attacked.

Finally, symbolic methods are also ill-suited for the analysis of protocols that necessarily rely on weak primitives for one reason or another. Such protocols include those using Short Authentication Strings (SAS) [119,120,142,144], in which the strings used for authentication must be manually or visually checked by a human, and distance-bounding protocols relying on rapid-bit exchange [53, 129], where the messages transmitted are multiple individual bits and security guarantees rely on the implausibility of the attacker guessing all of them. Such protocols are amenable to analysis with our probabilistic framework.

Our contribution. Our framework can be used to model security relevant properties of cryptographic primitives and their implementations by means of relations

between the input and the output of cryptographic primitives. Despite using a fairly simple language for describing such properties, our framework can be used to model, for example, random number generation algorithms that generate bitstrings representing primes of a certain length, hash function that leak partial information about the original message, or cryptosystems whose valid public keys have some recognizable structure. More complex properties, such as those employed in differential cryptanalysis, can also be modelled, as in [128]. The specified properties can then be used to find attacks and to estimate their success probability. Such properties cannot be modeled by existing symbolic methods and often lead to attacks on real-world implementations.

We model cryptographic functions using a generalized random oracle model. Given a specification of the cryptographic primitives used and their properties, symbolic terms are sampled to bitstrings in a way that ensures that the specification properties are always satisfied, but otherwise functions behave as random oracles. Under reasonable assumptions on the specification, we can define such generalized random oracles and prove that they yield valid probability measures. We believe that this model is interesting in its own right. It is a non-trivial generalization of the standard random oracle model for hash functions, and it captures the intuitive idea that cryptographic primitives satisfy stated properties (which may be exploited by an attacker) but otherwise behave ideally. Moreover, we show that probabilities in this model can be effectively computed, and we provide a prototype implementation that calculates these probabilities.

We illustrate the usefulness of our framework by representing the redundancy of RSA keys and using this to model and estimate the success probability of off-line guessing attacks on variants of the EKE protocol [40]. Although these attacks are well-known, their analysis was previously outside the scope of symbolic methods.

This symbolic probabilistic framework for security protocol analysis is joint work with David Basin and Carlos Caleiro, and it has been published in [78].

1.4 Outline

In Chapter 2 we introduce the basic definitions that are used throughout this thesis. Namely, we provide a brief description of the applied-pi calculus, and introduce our treatment of frames and equational theories. Chapter 3 describes the FAST algorithm and tool. We provide a detailed complexity analysis and compare it with that of competing tools. The algorithm's performance is also assessed empirically by means of a set of benchmarks. Chapter 4 describes our constraint-solving based procedure for deciding the trace equivalence of simple bounded processes. In Chapter 5 we present our symbolic probabilistic framework for the analysis of security protocols. As a case study, we analyze different attacks on the EKE protocol (described in [40]) to show how our framework can be used to find non-trivial attacks that depend on weaknesses of the cryptographic primitives being used. Finally, we summarize our contributions and outline future work in Chapter 6.

To keep the presentation as simple and readable as possible, we present most of our proofs in separate Appendices. Proofs for the results presented in Chapters 3, 4 and 5 can be found in Appendices A, B and C, respectively.

Chapter 2

Background and Basic Definitions

In this section we introduce the applied-pi calculus, a symbolic formalism which we use to reason about security protocols, as well as notation and technical notions used throughout this thesis.

2.1 The Applied-Pi Calculus

The applied-pi calculus is a symbolic formalism for modeling security protocols. Messages are represented as terms in a term algebra. Protocol participants are represented as *processes*, passing messages to each other via *channels*. Protocols are represented by the parallel composition of the processes representing the protocol participants.

In this section we briefly introduce the most important definitions in the applied-pi calculus. We closely follow the presentation given in [65], since our procedure for deciding trace equivalence relies on the reduction of trace equivalence of bounded processes to symbolic equivalence of constraint systems presented there.

2.1.1 Term Algebra

We assume fixed disjoint, countably infinite sets $\mathcal{N}_b$, $\mathcal{N}_c$, $\mathcal{X}_b$ and $\mathcal{X}_c$ of *names of base type*, *names of channel type*, *variables of base type*, and *variables of channel type*, respectively. We write $\mathcal{X}$ for the set $\mathcal{X}_b \cup \mathcal{X}_c$ of all variables. We also fix a finite *signature* $\Sigma = \biguplus_{n\in\mathbb{N}} \Sigma$ containing *function symbols*, with Σ_n containing the symbols of arity n. We write $ar(f)$ for the arity of a function symbol $f \in \Sigma$. Given a set A of *atoms*, $T_\Sigma(A)$ is the set of Σ-terms over A, i.e., $A \subseteq T_\Sigma(A)$ and $f(t_1, \ldots, t_n) \in T_\Sigma(A)$ whenever $t_1, \ldots, t_n \in T_\Sigma(A)$ and $f \in \Sigma_n$. *Terms* in the applied-pi calculus are elements of $T_\Sigma(\mathcal{N}_b \cup \mathcal{X})$; *ground terms* are elements of $T_\Sigma(\mathcal{N}_b)$.

Associated to the signature Σ is an equational theory $=_{\mathsf{E}}$, i.e., a congruence relation on the set of terms that is closed under sort-respecting substitution of terms for variables (that is, substitutions that replace variables of base type and channel

type by channel names and terms built over names of base type, respectively). Equational theories are used to represent algebraic properties of the cryptographic primitives described by function symbols in Σ.

Example 1. Symbolic models for security protocol analysis typically consider a set of function symbols representing some of the most commonly used cryptographic primitives. These usually consist of a hash function, pairing and projection, and symmetric and asymmetric encryption.

To model these primitives in the applied-pi calculus we use the signature $\Sigma^{\mathcal{DY}}$ given by $\Sigma^{\mathcal{DY}} = \Sigma_1^{\mathcal{DY}} \cup \Sigma_2^{\mathcal{DY}} \cup \Sigma_3^{\mathcal{DY}}$, where

$$\Sigma_1^{\mathcal{DY}} = \left\{\mathsf{h}, \pi_1, \pi_2, \cdot_{\mathsf{pub}}, \cdot_{\mathsf{priv}}\right\},$$

$$\Sigma_2^{\mathcal{DY}} = \left\{\{|\cdot|\}_\cdot, \{|\cdot|\}_\cdot^{-1}, \{\cdot\}_\cdot^{-1}\langle\cdot,\cdot\rangle\right\},$$

and

$$\Sigma_3^{\mathcal{DY}} = \left\{\{\cdot\}_\cdot^\cdot\right\}.$$

Terms are obtained by using names and variables as atomic terms and applying the function symbols in $\Sigma^{\mathcal{DY}}$. The application of function symbols in $\Sigma^{\mathcal{DY}}$ is interpreted as follows:

- $\mathsf{h}(M)$ represents the hash of some term M.
- $\cdot_{\mathsf{pub}}$ and $\cdot_{\mathsf{priv}}$ are functions used to extract a public-private key pair from some seed term. More concretely, if M is a term, then M_{pub} is a public key and its corresponding (inverse) private key is M_{priv}.
- $\langle M, N\rangle$ is the pairing of terms M and N.
- π_1 (respectively, π_2) is a projection function: if M is a pair, then $\pi_1(M)$ (respectively, $\pi_2(M)$) represents the extraction of the first (respectively, second) element of the pair.
- $\{|M|\}_K$ (respectively, $\{|M|\}_K^{-1}$) represents the symmetric encryption (respectively, symmetric decryption) of a term M using term K as key.
- $\{M\}_K^r$ represents the (randomized) public key encryption of a message M, using K as the encryption key and r as the random coins used in the encryption.
- $\{M\}_K^{-1}$ represents the assymmetric decryption of a term M using term K as the decryption key.

The equational properties of these primitives are expressed by the equations

$$\pi_1(\langle x, y\rangle) =_{\mathsf{E}} x, \qquad \pi_2(\langle x, y\rangle) =_{\mathsf{E}} y,$$

$$\left\{\left|\{|x|\}_y\right|\right\}_y^{-1} =_{\mathsf{E}} x, \quad \text{and} \quad \left\{\{x\}_{y_{\mathsf{pub}}}^z\right\}_{y_{\mathsf{priv}}}^{-1} =_{\mathsf{E}} x.$$

Example 2. In our examples we use a slight variant $\Sigma^{\mathcal{DY},d}$ of the signature $\Sigma^{\mathcal{DY}}$ introduced above where asymmetric encryption is *deterministic*. The signature $\Sigma^{\mathcal{DY},d}$ is defined by $\Sigma^{\mathcal{DY},d} = \bigcup_{n\in\mathbb{N}} \Sigma_n^{\mathcal{DY},d}$, with

$$\Sigma_1^{\mathcal{DY},d} = \Sigma_1^{\mathcal{DY}}, \qquad \Sigma_2^{\mathcal{DY},d} = \Sigma_2^{\mathcal{DY}} \cup \{\{\cdot\}_\cdot\},$$

and $\Sigma_n^{\mathcal{DY},d} = \emptyset$ for all $n > 2$. The corresponding equational theory $=_{\mathsf{E},d}$ is defined by the following equations:

$$\pi_1(\langle x, y\rangle) =_{\mathsf{E},d} x, \qquad \pi_2(\langle x, y\rangle) =_{\mathsf{E},d} y,$$

$$\left\{\left|\{|x|\}_y\right|\right\}_y^{-1} =_{\mathsf{E},d} x, \quad \text{and} \quad \left\{\{x\}_{y_{\mathsf{pub}}}\right\}_{y_{\mathsf{priv}}}^{-1} =_{\mathsf{E},d} x.$$

2.1.2 Processes

Processes in the applied-pi calculus are used to represent protocol participants. *Plain process* are defined recursively: $\mathbf{0}$ is a plain process (the *null process*) and, if P, Q are plain processes, $n \in \mathcal{N}$, $x \in \mathcal{X}$, $u \in \mathcal{N}_b \cup \mathcal{X}$, and $M, N \in T_\Sigma(\mathcal{N}_b \cup \mathcal{X})$, then $P \mid Q$ (parallel composition), $!P$ (replication), $\nu n.P$ (name restriction), $\texttt{if } M = N \texttt{ then } P \texttt{ else } Q$ (conditional), $\texttt{in}(u, x).P$ (message input), and $\texttt{out}(u, N).P$ (message output) are plain processes.

Extended processes are also defined recursively: every plain process is an extended process and, if A, B are extended processes and n, x, M are as above, then $A \mid B$ (parallel composition), $\nu n.A$ (name restriction), $\nu x.A$ (variable restriction), and $\{^M/_x\}$ (active substitution) are extended processes.

An extended process is *closed* if all its variables are either bound by ν or defined by an active substitution. An *evaluation context* is an extended process $C[\_]$ with a hole instead of an extended process. The hole in an evaluation context must not be under replication or a conditional, and must not be an input nor an output.

As in the pi calculus, names and variables have *scopes* determined by restrictions and by inputs. If A is an extended process, $fv(A)$, $fn(A)$, $bv(A)$ and $bn(A)$ denote the sets of *free variables in* A, *free names in* A, *bound variables in* A, and *bound names in* A, respectively.

Example 3. Figure 1 introduces a simple, toy example of an applied-pi calculus process. This process will also be used in Chapter 4 to illustrate the definitions and techniques introduced there. It is meant to ease the presentation and not as a real-world case study.

The process P_E represents the concurrent unbounded execution of the protocols described by processes P_E^1 and P_E^2. In Figure 1, A and B are agent names, $K\mathsf{AB}$ represent a symmetric key shared between A and B, and $K\mathsf{A}$ and $K\mathsf{B}$ represent the random data used to generate A and B's asymmetric key pair, respectively. Note then that $K\mathsf{A}_{\mathsf{pub}}$ and $K\mathsf{B}_{\mathsf{pub}}$ represent A and B's public key. γ_K is used as shorthand for $\left\{\{|x_1|\}_{kIR}^{-1}\right\}_{skR_{\mathsf{priv}}}^{-1}$. The names r_1, r_2 represent random data used to

randomize asymmetric encryption and are therefore secret. K is used as a symmetric session key in P_E^2. Messages M and N are meant to be randomly generated by the initiator and the responder, respectively, and kept secret.

$$
\begin{aligned}
&P_\nu = \nu K\mathsf{A}.\, \nu K\mathsf{B}.\, \nu K\mathsf{AB}.\, \left\{{}^{K\mathsf{A}_{\mathsf{pub}}}/_{x\mathsf{A}}, {}^{K\mathsf{B}_{\mathsf{pub}}}/_{x\mathsf{B}}\right\}\\
\\
&P_E = P_\nu.(!\left(\nu \mathit{skR}.\left(P_E^1 \mid \left\{{}^{K\mathsf{A}_{\mathsf{pub}}}/_{\mathit{pkI}}, {}^{K\mathsf{B}_{\mathsf{pub}}}/_{\mathit{pkR}}, {}^{K\mathsf{B}}/_{\mathit{skR}}\right\}\right)\right)\\
&\qquad \mid !\left(\nu \mathit{skR}.\left(P_E^1 \mid \left\{{}^{K\mathsf{B}_{\mathsf{pub}}}/_{\mathit{pkI}}, {}^{K\mathsf{A}_{\mathsf{pub}}}/_{\mathit{pkR}}, {}^{K\mathsf{B}}/_{\mathit{skR}}\right\}\right)\right)\\
&\qquad \mid !\left(\nu \mathit{kIR}.\, \nu \mathit{skR}.\left(P_{E,2}\left\{{}^{K\mathsf{B}_{\mathsf{pub}}}/_{\mathit{pkR}}, {}^{K\mathsf{B}}/_{\mathit{skR}}, {}^{K\mathsf{AB}}/_{\mathit{kIR}}\right\}\right)\right)\\
&\qquad \mid !\left(\nu \mathit{kIR}.\, \nu \mathit{skR}.\left(P_{E,2}\left\{{}^{K\mathsf{A}_{\mathsf{pub}}}/_{\mathit{pkR}}, {}^{K\mathsf{A}}/_{\mathit{skR}}, {}^{K\mathsf{AB}}/_{\mathit{kIR}}\right\}\right)\right)\\
\\
&P_E^1 = \nu M.\, \nu r_1.\, \nu r_2.\\
&\quad \mid \mathtt{out}\left(c_1, \{M\}^{r_1}_{\mathit{pkR}}\right)\\
&\quad \mid (\mathtt{in}(c_1, y).\mathtt{out}\left(c_1, \left\{\{y\}^{-1}_{\mathit{skR}_{\mathsf{priv}}}\right\}^{r_2}_{\mathit{pkI}}\right)).\\
\\
&P_E^2 = \nu K.\, \nu r_1.\, \nu M.\, \nu N.\, \nu y.\\
&\quad \mid (\mathtt{out}\left(c_2, \left\{\left|\{K\}^{r_1}_{\mathit{pkR}}\right|\right\}_{\mathit{kIR}}\right).\\
&\qquad \mathtt{in}(c_2, y).\\
&\qquad \mathtt{out}\left(c_2, \left\{\left|\langle\{|x_2|\}^{-1}_K, N\rangle\right|\right\}_K\right))\\
&\quad \mid (\mathtt{in}(c_2, y).\\
&\qquad \mathtt{out}(c_2, \{|M|\}_{\gamma_K}).\\
&\qquad \mathtt{in}(c_2, y).\mathtt{if}\ \pi_1\left(\left\{\left|\left\{\left|\langle\{|x_2|\}^{-1}_K, N\rangle\right|\right\}_K\right|\right\}^{-1}_{\gamma_K}\right) = M\\
&\qquad\qquad \mathtt{then\ out}\left(c_2, \left\{\left|\pi_2(\{|x_3|\}^{-1}_{\gamma_K})\right|\right\}_{\gamma_K}\right)\\
&\qquad\qquad \mathtt{else}\ \mathbf{0})
\end{aligned}
$$

Figure 1: The process P_E.

Structural equivalence. *Structural equivalence* is the smallest relation $\equiv$ on extended processes that is closed under α-conversion of names and variables, by application of extended processes, and such that $A \mid \mathbf{0} \equiv A$, the operator $\mid$ is commutative and associative, ν is a binding operator, $\nu x.\, \left\{{}^{M}/_{x}\right\} \equiv \mathbf{0}$, $\left\{{}^{M}/_{x}\right\} \mid A \equiv \left\{{}^{M}/_{x}\right\} \mid A\left\{{}^{M}/_{x}\right\}$ and, if $M =_{\mathsf{E}} N$, then $\left\{{}^{M}/_{x}\right\} \equiv \left\{{}^{N}/_{x}\right\}$. Structural equivalence is the simplest and strictest form of process equivalence. Intuitively it means that two processes are the same modulo the equational theory and syntactic details of the applied-pi calculus.

Structural equivalence allows us to define *frames*: namely, every closed extended process A is structurally equivalent to

$$\nu n_1 \ldots n_k.\, \left\{{}^{M_1}/_{x_1}, \ldots, {}^{M_n}/_{x_n}\right\} \mid P$$

for some terms $M_1, \ldots, M_n$, some set of names $\{n_1, \ldots, n_k\} \subseteq fn(M_1, \ldots, M_n)$, and some closed plain process P such that $fv(P) = fv(M_1, \ldots, M_n) = \emptyset$ [6]. We write $\tilde{n}$ for the set $\{n_1, \ldots, n_k\}$ and σ for the substitution $\left\{{}^{M_1}/_{x_1}, \ldots, {}^{M_n}/_{x_n}\right\}$. We say that $dom(\sigma) = \{x_1, \ldots x_n\}$. The process $\nu\tilde{n}.\sigma$ is A's *frame*, denoted by $\phi(A)$. A frame is *closed* if no free variables occur in the range of σ.

Frames represent the knowledge of an eavesdropper who observes all messages exchanged over the network, but does not take into account the dynamic behaviour of the process A. The variables in $dom(\sigma)$ can be seen as handles that refer to the corresponding messages that are sent over the network.

Internal reduction. The *internal reduction* relation $\xrightarrow{\tau}$ is the smallest relation on extended processes closed under structural equivalence and application of evaluation contexts such that

- $\mathtt{out}(a, M).P \mid \mathtt{in}(a, x).Q \xrightarrow{\tau} P \mid Q\left\{{}^{M}/_{x}\right\}$;
- if $M =_{\mathsf{E}} N$, then $\mathtt{if}\ M = N\ \mathtt{then}\ P\ \mathtt{else}\ Q \xrightarrow{\tau} P$;
- if $M \neq_{\mathsf{E}} N$ and M, N are ground terms, then $\mathtt{if}\ M = N\ \mathtt{then}\ P\ \mathtt{else}\ Q \xrightarrow{\tau} Q$.

Operational semantics. The operational semantics of the applied-pi calculus is given by a *transition relation* $\xrightarrow{\ell}$, combining the labeled transition rules in Figure 2 with the internal reduction rules defined above. Note that all rules are considered modulo structural equivalence. Given an extended process, $\xrightarrow{\ell}$ specifies which actions are allowed on that process. Actions include sending a message on a channel, accepting a message on a channel, and opening a channel. Transitions are labeled with the corresponding action, i.e., a transition corresponding to an action α is denoted by $\xrightarrow{\alpha}$.

2.1.3 Process Equivalences

Static equivalence. Two terms M and N are *equal in a frame* $\phi = \nu\tilde{n}.\sigma$, written $(M =_{\mathsf{E}} N)\phi$, if $\tilde{n} \cap (fn(M) \cup fn(N)) = \emptyset$ and $M\sigma =_{\mathsf{E}} N\sigma$. Two closed frames $\phi = \nu\tilde{n}.\sigma$ and $\phi' = \nu\tilde{n}'.\sigma'$ are *statically equivalent*, written $\phi \sim^s \phi'$, iff for all terms M, N we have $(M =_{\mathsf{E}} N)\phi$ iff $(M =_{\mathsf{E}} N)\phi'$.

Trace equivalence. Let $\mathcal{A}$ be the (infinite) set of actions on a process (defined by the labeled transition relation $\xrightarrow{\ell}$), with τ representing a distinguished unobservable action. If $w \in \mathcal{A}^*$, the transition relation $\xrightarrow{w}$ is defined as expected; if $s \in (\mathcal{A} \setminus \{\tau\})^*$, then $A \stackrel{s}{\Rightarrow} B$ if there is $w \in \mathcal{A}^*$ such that $A \xrightarrow{w} B$ and s is obtained from w by removing all occurrences of τ. Thus, $A \stackrel{s}{\Rightarrow} B$ means that there exists a sequence of actions that transform the extended process A into the extended process B and such that s is the sequence of its observable actions.

$$\textsc{In} \quad \mathtt{in}(a, y).P \xrightarrow{in(a,M)} P\left\{{}^{M}/_{y}\right\}$$

$$\textsc{Out-Ch} \quad \mathtt{out}(a, c).P \xrightarrow{out(a,c)} P$$

$$\textsc{Open-Ch} \quad \frac{A \xrightarrow{out(a,c)} A' \qquad c \neq a}{\nu c.A \xrightarrow{\nu x.out(a,x)} A'}$$

$$\textsc{Out-T} \quad \mathtt{out}(a, M).P \xrightarrow{\nu x.out(a,x)} P \mid \left\{{}^{M}/_{x}\right\} \quad (\text{if } x \notin fv(P) \cup fv(M))$$

$$\textsc{Scope} \quad \frac{A \xrightarrow{\ell} A' \qquad u \text{ does not occur in } \ell}{\nu u.A \xrightarrow{\ell} \nu u.A'}$$

$$\textsc{Par} \quad \frac{A \xrightarrow{\ell} A' \qquad bv(\ell) \cap fv(B) = \emptyset}{A \mid B \xrightarrow{\ell} A' \mid B}$$

$$\textsc{Struct} \quad \frac{A \equiv B \qquad B \xrightarrow{\ell} B' \qquad B' \equiv A'}{A \xrightarrow{\ell} A'}$$

where $a, b \in \mathcal{N}_c$, $x \in \mathcal{X}_b$, and $y \in \mathcal{X}_b \cup \mathcal{X}_c$.

Figure 2: Transition rules in the applied-pi calculus.

If A and B are extended processes, we write $A \sqsubseteq_t B$ if, for all s and A' such that $A \stackrel{s}{\Rightarrow} A'$, there exists a B' such that $B \stackrel{s}{\Rightarrow} B'$ and $\phi(A') \sim^s \phi(B')$ (i.e., $\phi(A')$ and $\phi(B')$ are statically equivalent). A and B are statically equivalent if $A \sqsubseteq_t B$ and $B \sqsubseteq_t A$. Intuitively, this means that for any sequence of actions that can be observed on process A, the same sequence of actions can be observed on process B, and the final resulting frames are statically equivalent. This models the fact that an observer cannot tell which process A or B he observes.

2.2 Our Framework

Given a function f, we denote by $dom(f)$ and $ran(f)$ its domain and range, respectively. When $X \subseteq dom(f)$, we write $f[X]$ for the image of X under f, that is, $f[X] = \{f(x) \mid x \in X\}$. If Y is a set, we write $f\mid_Y$ for the restriction of f to

Y, so that $f\mid_Y = \{y \mapsto f(y) \mid y \in Y\}$. The notation $id(X)$ is used to denote the identity function $\{x \mapsto x \mid x \in X\}$ of a set X. We write $f\colon A \nrightarrow B$ to denote a partial function f from A to B.

2.3 Terms and Recipes

Signatures. We assume fixed a signature Σ as in the definition of the applied-pi calculus above, and define term algebras $T_\Sigma(X)$ in the same way. We also assume fixed countably infinite, pairwise disjoint sets $\mathcal{N}$ and $\mathcal{X}$ of *names* and *variables*, respectively. Variables in our framework are used as handles for the messages transmitted over a network, as in the applied-pi calculus, and in equational reasoning.

The set $sub(t)$ of subterms of a term t is defined recursively as usual:

- if $t \in X$, then $sub(t) = \{t\}$, and
- if $t = f(t_1, \ldots, t_n)$, then

$$sub(t) = \{t\} \cup \left(\bigcup_{i=1}^{n} sub(t_i) \right).$$

The set $psub(t)$ of *proper subterms* of t is defined by $psub(t) = sub(t) \setminus \{t\}$.

If $t \in T_\Sigma(X)$, we use $names(t) = sub(t) \cap \mathcal{N}$, and $vars(t) = sub(t) \cap \mathcal{X}$ to denote, respectively, the sets of names and variables occurring in t. We extend these notions as expected to sets $T \subseteq T_\Sigma(X)$: thus, $names(T) = \bigcup_{t \in T} names(t)$, and analogously for $vars(T)$. For simplicity, we write c instead of $c()$ if $c \in \Sigma_0$. We define the *head* of a term $t = f(t_1, \ldots, t_n)$ by $head(t) = f$.

The set of positions $pos(t) \subset \mathbb{N}^*$ in a term t and the notion of subterm $t\mid_p$ of t at position $p \in pos(t)$ is defined recursively as follows:

- $\epsilon \in pos(t)$ and $t\mid_\epsilon = t$, where ϵ is the empty sequence;
- if $p \in pos(t)$, $t\mid_p = f(t_1, \ldots, t_n)$, and $i \in \{1, \ldots, n\}$, then $p.i \in pos(t)$ and $t\mid_{p.i} = t_i$, where $p.i$ represents the sequence obtaining by appending i to the end of the sequence p.

Substitutions. A substitution is a function σ with finite domain; as usual, we write $x\sigma$ instead of $\sigma(x)$ for $x \in dom(\sigma)$. For any sets X, Y and Z, a substitution $\sigma\colon X \to T_\Sigma(Y)$ has an homomorphic extension $\hat{\sigma}\colon T_\Sigma(X \cup Z) \to T_\Sigma(Y \cup Z)$ defined by $t\hat{\sigma} = t\sigma$ if $t \in X$, $t\hat{\sigma} = t$ if $t \in Z \setminus X$, and $f(t_1, \ldots, t_n)\hat{\sigma} = f(t_1\hat{\sigma}, \ldots, t_n\hat{\sigma})$ if $f \in \Sigma_n$ and $t_1, \ldots, t_n \in T_\Sigma(X \cup Z)$. As usual, we abuse notation by using the symbol σ to refer also to $\hat{\sigma}$. A *variable substitution* is a substitution $\alpha\colon X \nrightarrow T_\Sigma(\mathcal{N} \cup X)$.

Frames and Recipes. A frame is a pair $(\tilde{n}, \sigma)$, written $\upsilon\tilde{n}.\sigma$, where $\tilde{n} \subseteq \mathcal{N}$ is a finite set of names and $\sigma\colon \mathcal{X} \nrightarrow T_\Sigma(\mathcal{N})$ is a substitution with finite domain. As in the applied-pi calculus, names in $\tilde{n}$ represent fresh data generated by agents and thus unavailable to the attacker, and σ represents the messages learned by the attacker by eavesdropping on the network. Given a frame $\phi = \upsilon\tilde{n}.\sigma$, we define the set of ϕ-recipes $T(\phi)$ by $T(\phi) = T_\Sigma((\mathcal{N} \setminus \tilde{n}) \cup dom(\sigma))$. We say that terms in $\sigma[T(\phi)]$ are the terms *constructible from* ϕ.

We often abuse notation by associating a frame with its corresponding substitution. Thus, if $\phi = \nu\tilde{n}.\sigma$ is a frame, we define the domain of ϕ by $dom(\phi) = dom(\sigma)$ and, if $x \in dom(\sigma)$, we define $x\phi = x\sigma$. Moreover, if ζ is a ϕ-recipe, we may also write $\zeta\phi$ instead of $\zeta\sigma$, and $\phi[T(\phi)]$ instead of $\sigma[T(\phi)]$.

2.4 Equational Reasoning

Rewriting systems. A *rewrite rule* is a pair (l, r), written $l \to r$, where $l, r \in T_\Sigma(\mathcal{X})$ and $vars(r) \subseteq vars(l)$. A *rewriting system* $\mathcal{R}$ is a finite set of rewrite rules. We denote by $\mathcal{R}_L$ the set of left-hand sides of rewrite rules in $\mathcal{R}$, that is, $\mathcal{R}_L = \bigcup_{(l\to r)\in\mathcal{R}} \{l\}$. We define $nvars(\mathcal{R}) = \max_{l\in\mathcal{R}_L} |vars(l)|$ to be the maximum number of variables occurring in the left-hand side of a rewrite rule in $\mathcal{R}$.

Given a rewriting system $\mathcal{R}$, the *$\mathcal{R}$-rewriting relation* $\to_\mathcal{R} \subseteq T_\Sigma(Y) \times T_\Sigma(Y)$ on $T_\Sigma(Y)$ is, as usual, the smallest relation such that:

- if $(l \to r) \in \mathcal{R}$ and $\alpha\colon vars(l) \to T_\Sigma(Y)$ is a substitution, then $l\alpha \to_\mathcal{R} r\alpha$;
- if $f \in \Sigma_n$, $t_1, \ldots, t_n, t_i' \in T_\Sigma(Y)$, and there exists i such that $1 \le i \le n$ and $t_i \to_\mathcal{R} t_i'$, then $f(t_1, \ldots, t_i, \ldots, t_n) \to_\mathcal{R} f(t_1, \ldots, t_i', \ldots, t_n)$.

Note that we assume that $\mathcal{X} \cap Y = \emptyset$. We denote by $\to_\mathcal{R}^*$ the reflexive and transitive closure of $\to_\mathcal{R}$.

A rewriting system $\mathcal{R}$ is *confluent* if, for all terms $t \in T_\Sigma(Y)$, if $t \to_\mathcal{R}^* t'$ and $t \to_\mathcal{R}^* t''$, then there exists $t^* \in T_\Sigma(Y)$ such that $t' \to_\mathcal{R}^* t^*$ and $t'' \to_\mathcal{R}^* t^*$. $\mathcal{R}$ is *terminating* if there exists no infinite rewriting sequence, i.e., there exists no sequence $(t_i)_{i\in\mathbb{N}}$ such that, for all $i \in \mathbb{N}$, $t_i \in T_\Sigma(Y)$ and $t_i \to_\mathcal{R} t_{i+1}$. $\mathcal{R}$ is *convergent* if it is confluent and terminating.

A *$\mathcal{R}$-normal form* of $t \in T_\Sigma(Y)$ is a term $t{\downarrow} \in T_\Sigma(Y)$ such that $t \to_\mathcal{R}^* t{\downarrow}$ and no rewrite rules apply to $t{\downarrow}$, that is, there is no term $t' \in T_\Sigma(Y)$ such that $t{\downarrow} \to_\mathcal{R} t'$. If $\mathcal{R}$ is convergent, then each $t \in T_\Sigma(Y)$ has a unique normal form $t{\downarrow_\mathcal{R}}$ (c.f., *e.g.*, [17]). We write $t{\downarrow}$ when $\mathcal{R}$ is clear from the context. In this case, it is simple to check that the relation $\approx_\mathcal{R} \subseteq T_\Sigma(Y) \times T_\Sigma(Y)$ given by $\approx_\mathcal{R} = \{(t, t') \mid t{\downarrow_\mathcal{R}} = t'{\downarrow_\mathcal{R}}\}$ is a Σ-congruence. We say that $\approx_\mathcal{R}$ is the *equational theory generated by $\mathcal{R}$ on $T_\Sigma(Y)$*. If $f\colon X \to T_\Sigma(Y)$, we may write $f{\downarrow}$ for the function defined by $f{\downarrow}\,(x) = f(x){\downarrow}$ for all $x \in X$.

We say that a rewriting system is *subterm convergent* if $r \in sub(l) \cup \Sigma_0$ and $l \notin \Sigma_0$ for all $(l \to r) \in \mathcal{R}$. We say that equational theories generated by subterm convergent rewriting systems are *subterm theories*.

Throughout this thesis we always use equational theories generated by convergent rewriting systems; therefore, for simplicity, we often replace the symbol $\approx_{\mathcal{R}}$ by simply $\mathcal{R}$.

Example 4. Orienting from left to right the equations that define $=_{\mathsf{E}}$ in Example 1, we obtain a subterm convergent rewriting system $\mathcal{R}_{\mathcal{DY}}$ representing the equational properties of standard cryptographic primitives. This rewriting system is a standard representation of the capabilities of a symbolic attacker in an environment employing the most commonly used cryptographic primitives, and we use it frequently for our examples and as a case study.

Similarly, orienting from left to right the equations that define $=_{\mathsf{E},d}$ in Example 2 yield a subterm convergent rewriting system $\mathcal{R}_{\mathcal{DY},d}$.

2.5 Central Problems

Deducibility. We say that a term t is *deducible from* ϕ *under* $\mathcal{R}$, and write $\phi \vdash_{\mathcal{R}} t$, if there is a $t' \in \phi[T(\phi)]$ such that $t' \approx_{\mathcal{R}} t$.

Static equivalence. Given two frames $\phi = \upsilon\tilde{n}.\sigma$ and $\phi' = \upsilon\tilde{n}'.\sigma'$, we say that ϕ and ϕ' are *statically equivalent under* $\mathcal{R}$, and write $\phi \approx^s_{\mathcal{R}} \phi'$, if $T(\phi) = T(\phi')$ (i.e., $\tilde{n} = \tilde{n}'$ and $dom(\sigma) = dom(\sigma')$) and, for all $t, t' \in T(\phi)$, $t\sigma \approx_{\mathcal{R}} t'\sigma$ if and only if $t\sigma' \approx_{\mathcal{R}} t'\sigma'$.

2.6 Off-line Guessing

Equivalence notions are well-suited for modeling off-line guessing attacks [4, 36, 79]. Suppose that a nonce g has low entropy. This may occur, for example, if g is a human-picked password. In this case, an attacker may choose a small set of bitstrings with a high probability of containing the bitstring represented by g. The attacker can then use each of these bitstrings as his guess for the password. The attack is successful if he can verify which of these guesses is correct.

In our study of security against off-line guessing attacks we consider the security of *protocols* as well as the security of particular *protocol executions*. Intuitively, a particular protocol execution is secure against off-line guessing attacks if, after observing (or possibly participating in) that execution, the attacker cannot perform (off-line) computations to validate or refute each of his guesses of the weak secret considered. As expected, a protocol is secure against off-line guessing if all executions of the protocol (in which the attacker may participate) are secure against off-line guessing attacks. We refer to off-line guessing given a particular protocol execution as *static off-line guessing* and to off-line guessing given a protocol as *dynamic off-line guessing*.

Off-line guessing is one of the simplest examples of equivalence properties used for security protocol analysis. We apply our models to the analysis of security

against off-line guessing attacks in both static (Chapters 3 and 5) and dynamic (Chapter 4) contexts.

We now present each of these problems in more detail.

2.6.1 Static Off-line Guessing

The following definition captures the intuition of (static) off-line guessing described above in the context of the applied-pi calculus and is standard in the literature, *e.g.* [4, 36].

Definition 1. *Let $\mathcal{R}$ be a convergent rewriting system, $\phi = \upsilon\tilde{n}.\sigma$ be a frame, and $g \in \mathcal{N}$ be a name. Fix some fresh name $w \in \mathcal{N} \setminus (sub(ran(\sigma)) \cup \{g\})$ and define ϕ_g and ϕ_w by*

$$\phi_g = \upsilon(\tilde{n} \cup \{w\}).(\sigma \cup \{x_{n+1} \mapsto g\}),$$
$$\phi_w = \upsilon(\tilde{n} \cup \{w\}).(\sigma \cup \{x_{n+1} \mapsto w\}).$$

We say that ϕ allows an *off-line guessing attack* of g under $\mathcal{R}$ *if $\phi_g \not\approx^s_{\mathcal{R}} \phi_w$.*

Note that this definition is independent of the particular choice of the name w.

Intuitively, the attacker's guess can be seen as a message in the network. The attacker does not know beforehand if his guess is correct, but he can check this if he can distinguish a frame in which x_{n+1} stands for a random name w from a frame in which x_{n+1} stands for the guessed name g.

This is the notion of off-line guessing we use in Chapter 3. In Chapter 5 we present a symbolic framework for the analysis of protocol security against a more powerful attacker who can use (some forms of) cryptanalysis and apply it to the study of static security against off-line guessing. The framework used to express the capabilities of such an attacker, and the corresponding notion of security against off-line guessing, are defined there.

2.6.2 Active Off-line Guessing

Protocol security against dynamic off-line guessing can be modeled as trace equivalence between processes representing an execution of the protocol (or protocols) considered followed by the leakage on a public channel of either the weak secret considered (representing a successful guess) or some fresh name (representing an unsuccessful guess). The standard technique is to admit a bounded number of sessions and consider all possible message interleavings that may occur within that number of sessions. The only remaining source of infinity in the analysis is then the attacker inputs to the network. To overcome this limitation, attacker inputs are represented symbolically: That is, whenever a protocol participant expects a message from the network, that message is represented as a variable which must be instantiated with a term that the attacker may deduce at that point of the execution of the protocol. [57, 58, 70, 125]

In the case of the example process P_E introduced in Example 3 we will also consider that the protocols are now executed in the presence of an adversary C,

and we assume fixed a particular interleaving of the messages exchanged over the network that we will use in a running example throughout Chapter 4.

Example 5. Recall the extended process P_E described in Example 3. Suppose that $K\mathsf{AB}$ is a weak key and we want to test whether an attacker can successfully mount an off-line guessing attack of it. We assume that the attacker C is a legitimate user of the network: i.e., C has a public key $K\mathsf{C}_{\mathsf{pub}}$ available to all agents and access to the corresponding key seed $K\mathsf{C}$, and shares symmetric keys $K\mathsf{AC}$ and $K\mathsf{BC}$ with the remaining agents.

Since our procedure only works for bounded processes, we must consider particular protocol runs, each specifying all protocol interleavings and branching, in the presence of an active adversary. Because we consider input from the environment (i.e., the attacker), the tree of possible executions is infinitely branching. Nevertheless, the problem can be decided by considering *symbolic traces*, as is usual in constraint solving approaches to protocol security, as explained above. We consider that, at the end of each such execution, the attacker "guesses" a value for the key $K\mathsf{AB}$. The execution admits an off-line guessing attack if the attacker may use the information he acquired during protocol execution to validate his guess, i.e., to distinguish a correct guess from an incorrect one. Thus, for each such interleaving, we consider two processes: one leaks the secret key, while the other one leaks a fresh random value. The particular protocol execution considered is secure against off-line guessing if the two resulting processes are trace equivalent, meaning that the information available to the attacker is not sufficient to distinguish a correct guess from an incorrect one.

For the purpose of this example, which we will use as the basis for our running example throughout Chapter 4, we will focus on a protocol run specified by the process P_E^* described in Figure 3. In this particular run, agent B starts executing an instance of protocol P_E^2 with agent A as the intended responder. The attacker C intercepts this message on channel c_2, encrypts it using A's public key, and redirects it to channel c_1, effectively using it as the first message of an execution of protocol P_E^1. Agent A reads this message on channel c_1 and responds by sending the appropriate message (as specified by protocol P_E^1) on channel c_1. The attacker again intercepts this message, storing it as the variable h (for *handle*). He then replaces the message sent by A and intended for B in the execution of protocol P_E^2 by his own message, represented here by the variable ρ. The execution of protocol P_E^2 then continues as normal, until in the last step A either accepts B's message or aborts. To avoid cluttering the notation, we only bind the names that are used in this particular run.

$$
\begin{array}{l}
P_E^* = \nu K\mathsf{A}.\ \nu K\mathsf{B}.\ \nu K\mathsf{AB}.\ \nu K.\ \nu r.\ \nu r'.\ \nu r_1.\ \nu M.\ \nu N. \\
\quad \left\{ {}^{K\mathsf{A}_{\mathsf{pub}}}/_{x\mathsf{A}}\ {}^{K\mathsf{B}_{\mathsf{pub}}}/_{x\mathsf{B}}, {}^{K\mathsf{C}_{\mathsf{pub}}}/_{x\mathsf{C}} \right\}. \\
\quad \left\{ \left\{ \left| \{K\}_{x\mathsf{A}}^{r_1} \right| \right\}_{K\mathsf{AB}}/_{x_1}, {}^{\{|M|\}_{\gamma_K}}/_{x_2} \right\}. \\
\quad \left\{ \left\{ \left| \langle \{|x_2|\}_K^{-1}, N \rangle \right| \right\}_K/_{x_3}, \left\{ \left| \pi_2(\{|x_3|\}_{\gamma_K}^{-1}) \right| \right\}_{\gamma_K}/_{x_4} \right\}. \\
\quad (\mathtt{out}(c_2, x_1).\mathtt{in}(c_2, x_1). \\
\qquad \mathtt{out}\,(c_1, \{x_1\}_{x\mathsf{A}}^{r})\,. \\
\qquad \mathtt{in}(c_1, y). \\
\qquad \mathtt{out}\left(c_1, \left\{ \{y\}_{K\mathsf{A}_{\mathsf{priv}}}^{-1} \right\}_{x\mathsf{C}}^{r'}\right). \\
\qquad \mathtt{in}(c_1, h) \\
\quad \mathtt{out}(c_2, \rho). \\
\quad \mathtt{in}(c_2, y).\mathtt{out}(c_2, x_2). \\
\quad \mathtt{in}(c_2, y).\mathtt{out}(c_2, x_3). \\
\quad \mathtt{in}(c_2, y).\mathtt{if}\ \ \pi_1(\{|x_3|\}_{\gamma_K}^{-1}) = M \\
\qquad\quad \mathtt{then\ out}(c_2, x_4)\ \mathtt{else}\ \mathbf{0})
\end{array}
$$

Figure 3: A particular execution P_E^* of P_E.

According to the reasoning above, we will study the equivalence of the processes

$$\nu w.\,(P_E^*.\mathtt{out}(c_1, K\mathsf{AB}))$$

and

$$\nu w.\,(P_E^*.\mathtt{out}(c_1, w))\,.$$

Chapter 3

The FAST Algorithm and Tool

In this chapter we describe efficient algorithms for deciding message deducibility and static equivalence under subterm theories and discuss our implementation thereof as the FAST tool.

We assume fixed a subterm convergent rewriting system $\mathcal{R}$ and a frame $\phi = \upsilon\tilde{n}.\sigma$ and $\phi' = \upsilon\tilde{n}'.\sigma'$ such that $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ and t_i is in normal form for all $i \in \{1, \ldots, n\}$.

3.1 DAG-Representation of Terms

In order to obtain polynomial complexity bounds for our algorithms, we will work with DAG (directed acyclic graph) representations of terms, as in [5].

Definition 2. *Let $t \in T_\Sigma(X)$ be a term. Let V be a set of vertices, $\lambda\colon V \to \Sigma \cup X$ a labelling function, $E \subseteq V \times V \times \mathbb{N}$ a set of edges, and $v \in V$ a vertex.*

If $t \in X$, then (V, λ, E, v) is a DAG-representation of t if $\lambda(v) = t$ and $(v, v', n) \notin E$ for all $v' \in V$ and all $n \in \mathbb{N}$.

If $t = f(t_1, \ldots, t_n)$, then (V, λ, E, v) is a DAG-representation of t if $\lambda(v) = f$, there are edges $(v, v_1, 1), \ldots, (v, v_n, n) \in E$ such that, for each $i \in \{1, \ldots, n\}$, (V, λ, E, v_i) is a DAG-representation of t_i, and E does not contain any other edges (v, v', m).

A tuple $\mathcal{T} = (V, \lambda, E)$ is a DAG-forest if, for all $v \in V$, (V, λ, E, v) represents some term $t \in T_\Sigma(X)$. If $\mathcal{T} = (V, \lambda, E)$ is a DAG-forest and $v \in V$, we use the following notions:

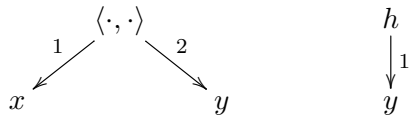
- $\mathsf{term}_\mathcal{T}(v)$ is the (unique) term represented by (V, λ, E, v);
- for each $i \in \mathbb{N}$, there exists at most one $v' \in V$ such that $(v, v', i) \in E$; if one such v' exists, we denote it by $\mathsf{e}_{i,\mathcal{T}}(v)$;
- $\mathsf{in}_\mathcal{T}(v) = \{w \in V \mid (w, v, i) \in E, \text{ for some } i\}$;
- $\mathsf{out}_\mathcal{T}(v) = \{w \in V \mid (v, w, i) \in E, \text{ for some } i\}$;

- $\mathsf{leaves}(\mathcal{T}) = \{v \in V \mid \mathsf{out}_{\mathcal{T}}(v) = \emptyset\}$;
- $\mathsf{roots}(\mathcal{T}) = \{v \in V \mid \mathsf{in}_{\mathcal{T}}(v) = \emptyset\}$.

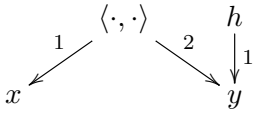
If $\mathcal{T}$ has only one root, we may refer to it as a DAG-tree.

Let $\mathcal{T} = (V, \lambda, E)$ be a DAG-forest. If $\mathsf{roots}(\mathcal{T}) = \{v\}$, we say that $\mathcal{T}$ is a DAG-representation of the term $\mathsf{term}_{\mathcal{T}}(v)$. When no confusion can arise, we may abuse notation and use the same symbol for such a DAG-forest and the term represented by it. $\mathcal{T}$ is *minimal* if there are no distinct vertices $v, v' \in V$ such that $\mathsf{term}_{\mathcal{T}}(v) = \mathsf{term}_{\mathcal{T}}(v')$. $\mathcal{T}$ is in normal form if, for each vertex v in $\mathcal{T}$, $\mathsf{term}_{\mathcal{T}}(v)$ is in normal form. A *normal form* of $\mathcal{T}$ is a DAG-forest $\mathcal{T}_{\mathsf{nf}}$ such that, for all $v \in \mathsf{roots}(\mathcal{T})$, there is a vertex v_{nf} in $\mathcal{T}_{\mathsf{nf}}$ such that $\mathsf{term}_{\mathcal{T}}(v)\downarrow = \mathsf{term}_{\mathcal{T}_{\mathsf{nf}}}(v_{\mathsf{nf}})$, and for each $v_{\mathsf{nf}} \in \mathsf{roots}(\mathcal{T}_{\mathsf{nf}})$, there is a $v \in \mathsf{roots}(\mathcal{T})$ such that $\mathsf{term}_{\mathcal{T}}(v)\downarrow = \mathsf{term}_{\mathcal{T}_{\mathsf{nf}}}(v_{\mathsf{nf}})$. The minimal normal form of a DAG-forest is unique up to renaming of vertices.

Example 6. The diagram



depicts a DAG-forest containing DAG-representations of the terms $\langle x, y\rangle$ and $h(y)$. Its minimal normal form is shown below.



Our complexity results depend on the rewriting system and are stated in terms of the size of terms and frames. If $t \in T_{\Sigma}(\mathcal{N})$ is a term, we define the *size* $|t|$ of t by

$$|t| = \begin{cases} 1 & \text{if } t \in X \\ 1 + \sum_{i=1}^{n} |t_i| & \text{if } t = f(t_1, \ldots, t_n) \end{cases}.$$

If ϕ is a frame, then the *size* $|\phi|$ of ϕ is given by

$$|\phi| = \sum_{x \in dom(\phi)} |x\phi|.$$

If $\mathcal{T} = (V, \lambda, E)$ is a DAG-forest, we define $|\mathcal{T}| = |V|$.

Indexing tree nodes. Each vertex v in a DAG-forest $\mathcal{T} = (V, \lambda, E)$ is associated with a *key*

$$(\lambda(v), \mathsf{e}_{1,\mathcal{T}}(v), \ldots, \mathsf{e}_{ar(\lambda(v)),\mathcal{T}}(v)).$$

If $\mathcal{T}$ is minimal, then for each key k there exists at most one vertex $v \in V$ associated with the key k. Given a DAG-forest $\mathcal{T}$ and a vertex v occurring in $\mathcal{T}$, we denote v's key by $\mathsf{key}(v)$.

Our complexity results assume that, given a key k, adding a vertex v with key k and accessing a (or determining that there is no) vertex with key k can be done in $\mathcal{O}(\log|\mathcal{T}|)$ time. This can be achieved, for example, by keeping a (self-balancing) binary search tree in which each key is $\mathsf{key}(v)$ for some vertex v occurring in $\mathcal{T}$, and the corresponding node contains (a pointer to) the vertex v. Additionally, we require that the sets $\mathsf{in}_{\mathcal{T}}(v)$ and $\mathsf{out}_{\mathcal{T}}(v)$ of incoming and outgoing edges of a vertex v be computed in constant time given a vertex v.

In the implementation of our algorithm as the FAST tool, each vertex in $\mathcal{T}$ is represented as a C++ object containing as data members a vector of incoming edges and a vector of outgoing edges. The tree is kept minimal by storing the set of vertices in a C++ `map` container, where each vertex's key is a C++ structure containing the vertex's label and a vector of pointers to its children vertices.

This implementation ensures that accessing the (or determining that there is no) vertex in $\mathcal{T}$ with a given key can be done in $\mathcal{O}(\log|\mathcal{T}|)$ and that the sets of incoming and outgoing edges of a vertex v can be accessed in constant time. Note that the size of the set of outgoing edges from a vertex v is not necessarily $\mathcal{O}(1)$; however, we have that

$$\sum_{v\in V} |\mathsf{out}_{\mathcal{T}}(v)| = |E|$$

is in $\mathcal{O}(|\mathcal{T}|)$.

3.2 Frame Saturation

In this section we present our frame saturation algorithm for subterm convergent equational theories. Frame saturation is an established technique for deciding deducibility and static equivalence [5,37,69]. Our procedure is less general than those implemented in [37, 69], but it is more efficient for subterm convergent equational theories.

Throughout this section we assume fixed a set $\Upsilon = \{\tau_1, \ldots, \tau_{nvars(\mathcal{R})}\}$ of fresh names, i.e.,

$$\Upsilon \cap (\tilde{n} \cup sub(ran(\phi)) \cup sub(ran(\phi'))) = \emptyset.$$

Definition 3. *We say that a frame $\phi_s = \upsilon\tilde{n}.\sigma_s$ is a* saturation of ϕ (under $\mathcal{R}$) *if the following conditions hold:*

(1) there exists a function $[\![\cdot]\!]_\Phi \colon dom(\phi_s) \to T(\phi)$ such that $[\![x]\!]_\Phi\phi \approx_{\mathcal{R}} x\phi_s$ for all $x \in dom(\phi_s)$;

(2) $(\zeta\phi)\!\downarrow \in \phi_s[T(\phi_s)]$ for all $\zeta \in T(\phi)$.

We say that the function $[\![\cdot]\!]_\Phi$ required by the definition of saturation of ϕ is a *ϕ-translation for ϕ_s*. As is standard for substitutions, we extend $[\![\cdot]\!]_\Phi$ to $T(\phi_s)$ homomorphically. It is straightforward to check that this extension satisfies $[\![\zeta]\!]_\Phi\phi \approx_\mathcal{R} \zeta\phi_s$ for all $\zeta \in T(\phi_s)$.

The following simple Lemma justifies the usefulness of saturations.

Lemma 1. *Let ϕ_s be a saturation of ϕ and $t \in T_\Sigma(\mathcal{N})$ be a term. Then, $\phi \vdash_\mathcal{R} t$ if and only if $t\!\downarrow \in \phi_s[T(\phi_s)]$.*

Proof. If $\phi \vdash_\mathcal{R} t$, then there exists $\zeta \in T(\phi)$ such that $\zeta\phi\!\downarrow = t\!\downarrow$; by (2), $\zeta\phi\!\downarrow \in \phi_s[T(\phi_s)]$, and thus also $t\!\downarrow \in \phi_s[T(\phi_s)]$.

On the other hand, if $t \in \phi_s[T(\phi_s)]$, then there is $\zeta \in T(\phi_s)$ such that $\zeta\phi_s = t\!\downarrow$, and we have $[\![\zeta]\!]_\Phi\phi \approx_\mathcal{R} \zeta\phi_s \approx_\mathcal{R} t$. Since $[\![\zeta]\!]_\Phi \in T(\phi)$, we conclude that $\phi \vdash_\mathcal{R} t$.

□

The DAG-Forest $\mathcal{T}_{\phi,\mathcal{R}}$

The first step in our saturation algorithm is to build a forest

$$\mathcal{T}_{\phi,\mathcal{R}} = (V_{\phi,\mathcal{R}}, \lambda_{\phi,\mathcal{R}}, E_{\phi,\mathcal{R}}),$$

as well as functions $[\![\cdot]\!]^{DAG}_\Phi$ and rw (line 1 in Algorithm 3). In this section we define these objects, show how to compute them, and present their relevant properties.

$\mathcal{T}_{\phi,\mathcal{R}}$ is a minimal DAG-forest containing DAG representations of all terms $l\sigma_l$, where l is the left-hand side of some rewrite rule $(l \to r) \in \mathcal{R}$ and $\sigma_l \colon vars(l) \to sub(ran(\phi)) \cup \Upsilon$ is a substitution.

The function $[\![\cdot]\!]^{DAG}_\Phi$ can be seen as an extended DAG-representation of the ϕ-translation function required by the definition of saturation: For each vertex $v \in V_{\phi,\mathcal{R}}$ representing a term $t \in ran(\sigma) \cup \Upsilon$, $[\![v]\!]^{DAG}_\Phi$ is a DAG-representation of a ϕ-recipe for t. Note that, if $t = x\sigma$ for some $x \in dom(\phi)$, we can choose $[\![v]\!]^{DAG}_\Phi$ to be a DAG-representation of x, and, if $t \in \Upsilon$, $[\![v]\!]^{DAG}_\Phi$ can be chosen to be a DAG-representation of t.

Finally, the function rw is such that, for each vertex v representing a term $l\sigma_l$ for some left-hand side l of a rewrite rule $l \to r$ and some substitution σ_l as described above, $\mathsf{rw}(v)$ is the (unique) vertex representing $r\sigma_l$.

We begin by presenting Algorithm 1, used for computing minimal normal forms of DAG-forests. Its correctness and complexity are stated in Lemma 2, proved in Appendix A.

Lemma 2. *Given a DAG-forest $\mathcal{T}$, Algorithm 1 computes a minimal normal form $\mathcal{T}_\mathsf{min} = (V_\mathsf{min}, \lambda_\mathsf{min}, E_\mathsf{min})$ of $\mathcal{T}$ in time $\mathcal{O}(|\mathcal{T}| \log |\mathcal{T}|)$, and the output function $\mathsf{min} \colon \mathsf{roots}(\mathcal{T}) \to V_\mathsf{min}$ is such that, for all $v \in \mathsf{roots}(\mathcal{T})$,*

$$(\mathsf{term}_\mathcal{T}(v))\!\downarrow = \mathsf{term}_{\mathcal{T}_\mathsf{min}}(\mathsf{min}(v)).$$

Algorithm 1 Algorithm for computing minimal normal forms of DAG-forests.

Input: a DAG-forest $\mathcal{T} = (V, \lambda, E)$
Output: a minimal normal form $\mathcal{T}_{\mathsf{min}} = (V_{\mathsf{min}}, \lambda_{\mathsf{min}}, E_{\mathsf{min}})$ of $\mathcal{T}$ and a function $\mathsf{min}\colon \mathsf{roots}(\mathcal{T}) \to V_{\mathsf{min}}$

$\mathsf{visitnow} \leftarrow \mathsf{leaves}(\mathcal{T})$, $\mathsf{visitnext} \leftarrow \emptyset$, $\mathsf{min} \leftarrow \emptyset$
$\mathcal{T}_{\mathsf{min}} = (V_{\mathsf{min}}, \lambda_{\mathsf{min}}, E_{\mathsf{min}}) \leftarrow (\emptyset, \emptyset, \emptyset)$
while $\mathsf{visitnow} \neq \emptyset$
  for all $v \in \mathsf{visitnow}$
    if $\mathsf{out}_{\mathcal{T}}(v) \subseteq dom(\mathsf{min})$ **then**
      $\mathsf{visitnext} = \mathsf{visitnext} \cup \mathsf{in}_{\mathcal{T}}(v)$
      if there is $v_{\mathsf{min}} \in V_{\mathsf{min}}$ **s.t.**
        $\mathsf{key}(v_{\mathsf{min}}) = (\lambda(v), \mathsf{min}(\mathsf{e}_{1,\mathcal{T}}(v)), \ldots, \mathsf{min}(\mathsf{e}_{ar\lambda(v),\mathcal{T}}(v)))$ **then**
        $\mathsf{min} \leftarrow \mathsf{min} \cup \{v \mapsto v_{\mathsf{min}}\}$
      else
        if there are $(l \to r) \in \mathcal{R}$, $\sigma_l\colon vars(l) \to T_\Sigma(X)$ **s.t.**
          $(\lambda(v))(\mathsf{term}_{\mathcal{T}_{\mathsf{min}}}(\mathsf{min}(\mathsf{e}_{1,\mathcal{T}_{\mathsf{min}}}(v))), \ldots, \mathsf{term}_{\mathcal{T}_{\mathsf{min}}}(\mathsf{min}(\mathsf{e}_{ar(\lambda(v)),\mathcal{T}_{\mathsf{min}}}(v)))) = l\sigma_l$ **then**
          $\mathsf{min} = \mathsf{min} \cup \{v \mapsto v_r\}$, **where** $\mathsf{term}_{\mathcal{T}_{\mathsf{min}}}(v_r) = r\sigma_l$
        else
          choose $v_{\mathsf{min}} \notin V_{\mathsf{min}}$
          $\mathsf{min} \leftarrow \mathsf{min} \cup \{v \mapsto v_{\mathsf{min}}\}$
          $V_{\mathsf{min}} \leftarrow V_{\mathsf{min}} \cup \{v_{\mathsf{min}}\}$
          $\lambda_{\mathsf{min}} \leftarrow l_{\mathsf{min}} \cup \{v_{\mathsf{min}} \mapsto \lambda(v)\}$
          $E_{\mathsf{min}} \leftarrow E_{\mathsf{min}} \cup \{(v_{\mathsf{min}}, \mathsf{min}(\mathsf{e}_{i,\mathcal{T}}(v)), i) \mid 1 \leq i \leq ar(\lambda(v))\}$
  $\mathsf{visitnow} \leftarrow \mathsf{visitnext}$
  $\mathsf{visitnext} \leftarrow \emptyset$
prune $\mathcal{T}_{\mathsf{min}}$ leaving only the subtrees of vertices in $\{\mathsf{min}(v) \mid v \in \mathsf{roots}(\mathcal{T})\}$
$\mathsf{min} = \mathsf{min}\,|_{\mathsf{min}[\mathsf{roots}(\mathcal{T})]}$
return $\mathcal{T}_{\mathsf{min}}, \mathsf{min}$

Algorithm 2 uses the union of disjoint DAG-forests, as defined below.

Definition 4. *Let $\mathcal{T}_1 = (V_1, \lambda_1, E_1)$ and $\mathcal{T}_2 = (V_2, \lambda_2, E_2)$ be DAG-forests such that the $V_1 \cap V_2 = \emptyset$. We define the union of $\mathcal{T}_1$ and $\mathcal{T}_2$ as*

$$\mathcal{T}_1 \cup \mathcal{T}_2 = (V_1 \cup V_2, \lambda_1 \cup \lambda_2, E_1 \cup E_2).$$

We extend this definition as expected to any arbitrary number of DAG-forests.

We now present Algorithm 2, used to obtain the DAG-forest $\mathcal{T}_{\phi,\mathcal{R}}$ and the functions $[\![\cdot]\!]^{DAG}_{\Phi}$ and rw described in the introduction to this section. The algorithm is straightforward. However, its efficiency requires representing DAG-trees with a data structure which allows nodes to be retrieved with $\mathcal{O}(\log |\mathcal{T}_{\phi,\mathcal{R}}|)$ complexity, such as the one described in Section 3.1.

Algorithm 2 Algorithm for computing $\mathcal{T}_{\phi,\mathcal{R}}$.

`Input:` a frame $\phi = \upsilon\tilde{n}.\phi$, with $\phi = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$
`Output:` a DAG-forest $\mathcal{T}_{\phi,\mathcal{R}}$ and the functions $[\![\cdot]\!]^{DAG}_{\Phi}$, rw

1: let $\mathcal{T}_1, \ldots, \mathcal{T}_{n+nvars(\mathcal{R})}$ be DAG-representations of $t_1, \ldots, t_n, \tau_1, \ldots, \tau_{nvars(\mathcal{R})}$ with roots $v_1, \ldots, v_{n+nvars(\mathcal{R})}$ and disjoint sets of vertices
2: let $\mathcal{T}(\phi) = (V_\phi, \lambda_\phi, E_\phi)$ and min be the output of Algorithm 1 on input $\bigcup_{i=1}^{n+nvars(\mathcal{R})} \mathcal{T}_i$
3: $\mathcal{T}_{\phi,\mathcal{R}} = (V_{\phi,\mathcal{R}}, \lambda_{\phi,\mathcal{R}}, E_{\phi,\mathcal{R}}) \leftarrow \mathcal{T}(\phi)$
4: $[\![\cdot]\!]^{DAG}_{\Phi} \leftarrow \{\mathsf{min}(v_{n+i}) \mapsto (\{v_{n+i}\}, \{v_{n+i} \mapsto \tau_i\}, \emptyset) \mid i \in \{1, \ldots, nvars(\mathcal{R})\}\}$
5: `for` $i = 1$ `to` n
6: `if` $\mathsf{min}(v_i) \notin dom([\![\cdot]\!]^{DAG}_{\Phi})$
7: `then` $[\![\cdot]\!]^{DAG}_{\Phi} \leftarrow ([\![\cdot]\!]^{DAG}_{\Phi} \cup \{v \mapsto (\{v_i\}, \{v_i \mapsto x_i\}, \emptyset)\})$
8: $\mathsf{rw} = \emptyset$
9: `for all` $(l \to r) \in \mathcal{R}, \sigma_l \colon vars(l) \to sub(ran(\phi)) \cup \Upsilon$
10: add to $\mathcal{T}_{\phi,\mathcal{R}}$ vertices to represent $l\sigma_l$, keeping $\mathcal{T}_{\phi,\mathcal{R}}$ minimal
11: $\mathsf{rw} \leftarrow \mathsf{rw} \cup \{v_l \mapsto v_r\}$, where $\mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(v_l) = l\sigma_l$ and $\mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(v_r) = r\sigma_l$

The relevant properties of $\mathcal{T}_{\phi,\mathcal{R}}$, $[\![\cdot]\!]^{DAG}_{\Phi}$ and rw, as well as the complexity of Algorithm 2, are stated in the following Lemma. The complexity of computing rw and $[\![\cdot]\!]^{DAG}_{\Phi}$ can be made logarithmic, e.g., by using a map data structure, i.e., imposing some order on the inputs and using them as keys in a balanced binary search tree whose nodes also contain the output corresponding to their key.

Lemma 3. *The forest $\mathcal{T}_{\phi,\mathcal{R}} = (V_{\phi,\mathcal{R}}, \lambda_{\phi,\mathcal{R}}, E_{\phi,\mathcal{R}})$ and the functions $[\![\cdot]\!]^{DAG}_{\Phi}$ and* rw *are such that:*

(1) *$\mathcal{T}_{\phi,\mathcal{R}}$ is minimal, can be obtained in time $\mathcal{O}(|\phi|^{nvars(\mathcal{R})} \log |\phi|)$, and $|\mathcal{T}_{\phi,\mathcal{R}}| \in \mathcal{O}(|\phi|^{nvars(\mathcal{R})})$;*

(2) rw *can be computed in time* $\mathcal{O}(\log|\phi|)$*;*

(3) $[\![\cdot]\!]^{DAG}_{\Phi}$ *can be computed in time* $\mathcal{O}(\log|\phi|)$ *and, for each* $v \in dom(\phi)$*,* $|[\![v]\!]^{DAG}_{\Phi}| = 1$*;*

(4) *for each* $s \in sub(ran(\phi)) \cup \Upsilon$*, there is an unique* v *such that* $\mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(v) = s$*;*

(5) *for each* $v \in dom(\mathsf{rw})$*,* $\mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(v) \to_{\mathcal{R}} \mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(\mathsf{rw}(v))$*;*

(6) *for each* $t \in ran(\phi) \cup \Upsilon$*, there is a* v *such that* $\mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(v) = t$ *and* $v \in dom([\![\cdot]\!]^{DAG}_{\Phi})$*;*

(7) *for each* $v \in dom([\![\cdot]\!]^{DAG}_{\Phi})$*,* $\mathsf{term}_{[\![v]\!]^{DAG}_{\Phi}}(v)$ *is a* ϕ*-recipe for* $\mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(v)$*;*

(8) *for each rule* $(l \to r) \in \mathcal{R}$ *and each substitution*

$$\sigma_l \colon vars(l) \to sub(ran(\phi)) \cup \Upsilon,$$

there is a unique $v \in V_{\phi,\mathcal{R}} \cap dom(\mathsf{rw})$ *such that* $\mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(v) = l\sigma_l$ *and* $\mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(\mathsf{rw}(v)) = r\sigma_l$*.*

Frame Saturation Algorithm

Our frame saturation algorithm is given in Algorithm 3. It performs a bottom-up traversal of the forest $\mathcal{T}_{\phi,\mathcal{R}}$ (described in the previous section). At each vertex v, whenever a recipe for the term $\mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(v)$ represented by that vertex is found, v is added to the range of $[\![\cdot]\!]^{DAG}_{\Phi}$, and $[\![v]\!]^{DAG}_{\Phi}$ is a DAG-representation of a ϕ-recipe for that term. A recipe is found if one has recipes $[\![v_1]\!]^{DAG}_{\Phi}, \ldots, [\![v_n]\!]^{DAG}_{\Phi}$ for all vertices v_i that have an incoming edge (v, v_i, i) from v. If the term represented by v is an instance of the left-hand side of a rule, then this recipe is also stored under $[\![\mathsf{rw}(v)]\!]^{DAG}_{\Phi}$ (note that $\mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(v) \to_{\mathcal{R}} \mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(\mathsf{rw}(v))$). Thus, throughout the saturation process, the function $[\![\cdot]\!]^{DAG}_{\Phi}$ associates each vertex v in its domain to a DAG-representation of a ϕ-recipe for $\mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(v)$. Whenever we add a vertex v to the domain of $[\![\cdot]\!]^{DAG}_{\Phi}$, we add all vertices v' with an outgoing edge (v', v, i) to v to the list of vertices to visit in the next iteration of the visiting loop, since it is possible that having a recipe for the term represented by v provides a recipe for the term represented by v', even if such a recipe could not be obtained before. At the end of the process, a term $t \in sub(ran(\phi))$ is deducible from ϕ if and only if the (unique) vertex representing that term is in the domain of $[\![\cdot]\!]^{DAG}_{\Phi}$.

The algorithm also stores the functions σ_s and $[\![\cdot]\!]_{\Phi}$. Only subterms of the range of ϕ whose corresponding vertices are added to the domain of $[\![\cdot]\!]^{DAG}_{\Phi}$ by using rewrite rules are added to the range of these functions. The function σ_s is such that $\phi_s = \upsilon\tilde{n}.\sigma_s$ is a saturation of ϕ, and $[\![\cdot]\!]_{\Phi}$ is such that $dom([\![\cdot]\!]_{\Phi}) = dom(\phi_s)$ and, for each $x \in dom(\phi_s)$, $[\![x]\!]_{\Phi}$ is a DAG-representation of a ϕ-recipe for $x\phi_s$.

Furthering our abuse of notation of using the same symbol for a term and its DAG-representation, we use the symbol $[\![\cdot]\!]_\Phi$ as the substitution that assigns, to each $x \in dom([\![\cdot]\!]_\Phi)$, the term represented by (the DAG-forest) $[\![x]\!]_\Phi$.

The tree $\mathcal{T}_{\phi,\mathcal{R}}$ has at most $\mathcal{O}(|\phi|^{nvars(\mathcal{R})})$ vertices, and each vertex $v \in V_{\phi,\mathcal{R}}$ is visited at most $|\mathsf{in}_{\mathcal{T}_{\phi,\mathcal{R}}}(v)|$ times. Thus, the total number of visits to vertices is at most $\mathcal{O}(|\phi|^{nvars(\mathcal{R})})$. By using suitable data structures, we can ensure that each visit takes at most time $\mathcal{O}(\log|\phi|)$. We thus obtain an asymptotic complexity of $\mathcal{O}(|\phi|^{nvars(\mathcal{R})}\log|\phi|)$.

Algorithm 3 Frame Saturation Algorithm

`Input:` a frame $\phi = v\tilde{n}.\sigma$, with $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$
`Output:` a saturation $\phi_s = v\tilde{n}.\sigma_s$ of ϕ and a function $[\![\cdot]\!]_\Phi$

1: compute $\mathcal{T}_{\phi,\mathcal{R}} = (V_{\phi,\mathcal{R}}, \lambda_{\phi,\mathcal{R}}, E_{\phi,\mathcal{R}})$, rw, $[\![\cdot]\!]_\Phi^{DAG}$
2: $[\![\cdot]\!]_\Phi \leftarrow \{x \mapsto (\{v_x\}, \{v_x \mapsto x\}, \emptyset) \mid x \in dom(\sigma)\}$,
where the v_x are such that $\mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(v_x) = x\phi$
3: $\phi_s \leftarrow \phi$
4: $\mathsf{visitnow} \leftarrow \mathsf{leaves}(\mathcal{T}_{\phi,\mathcal{R}}) \cup (\bigcup_{v \in dom([\![\cdot]\!]_\Phi^{DAG})} \mathsf{in}_{\mathcal{T}_{\phi,\mathcal{R}}}(v))$, $\mathsf{visitnext} \leftarrow \emptyset$
5: **while** $\mathsf{visitnow} \neq \emptyset$
6: **for all** $v \in \mathsf{visitnow}$
7: **if** $\lambda(v) \in X \setminus \tilde{n}$ **and** $v \notin dom([\![\cdot]\!]_\Phi^{DAG})$
8: $[\![\cdot]\!]_\Phi^{DAG} \leftarrow [\![\cdot]\!]_\Phi^{DAG} \cup \{v \mapsto (v, \{v \mapsto \lambda(v)\}, \emptyset)\}$
9: $\mathsf{visitnext} \leftarrow \mathsf{visitnext} \cup \mathsf{in}_{\mathcal{T}_{\phi,\mathcal{R}}}(v)$
10: **if** $\mathsf{out}_{\mathcal{T}_{\phi,\mathcal{R}}}(v) \subseteq dom([\![\cdot]\!]_\Phi^{DAG})$ **and** $v \notin dom([\![\cdot]\!]_\Phi^{DAG})$
11: **for** $i \in \{1, \ldots, ar(\lambda_{\phi,\mathcal{R}}(v))\}$
12: $(V_i, \lambda_i, E_i) \leftarrow [\![\mathsf{e}_{i,\mathcal{T}_{\phi,\mathcal{R}}}(v)]\!]_\Phi^{DAG}$
$[\![\cdot]\!]_\Phi^{DAG} \leftarrow [\![\cdot]\!]_\Phi^{DAG} \cup$
13: $\{v \mapsto (v \cup \left(\bigcup_{i=1}^{ar(\lambda_{\phi,\mathcal{R}}(v))} V_i\right), \{v \mapsto \lambda(v)\} \cup \bigcup_{i=1}^{ar(\lambda_{\phi,\mathcal{R}}(v))} \lambda_i, \bigcup_{i=1}^{v} \{(v, \mathsf{e}_{i,\mathcal{T}_{\phi,\mathcal{R}}}(v), i)\} \cup \bigcup_{i=1}^{ar(\lambda_{\phi,\mathcal{R}}(v))} E_i)\}$
14: **if** $v \in dom(\mathsf{rw})$ **and** $\mathsf{rw}(v) \notin dom([\![\cdot]\!]_\Phi^{DAG})$
15: $[\![\cdot]\!]_\Phi^{DAG} \leftarrow [\![\cdot]\!]_\Phi^{DAG} \cup \{\mathsf{rw}(v) \mapsto [\![v]\!]_\Phi^{DAG}\}$
16: **if** $\mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(\mathsf{rw}(v)) \in sub(ran(\phi))$
17: **choose** $x \in \mathcal{X} \setminus dom(\phi_s)$
18: $\phi_s \leftarrow \phi_s \cup \{x \mapsto \mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(\mathsf{rw}(v))\}$
19: $[\![\cdot]\!]_\Phi \leftarrow [\![\cdot]\!]_\Phi \cup \{x \mapsto [\![\mathsf{rw}(v)]\!]_\Phi^{DAG}\}$
20: $\mathsf{visitnext} \leftarrow \mathsf{visitnext} \cup \mathsf{in}_{\mathcal{T}_{\phi,\mathcal{R}}}(\mathsf{rw}(v))$
21: **else** $\mathsf{visitnext} \leftarrow \mathsf{visitnext} \cup \mathsf{in}_{\mathcal{T}_{\phi,\mathcal{R}}}(v)$
22: **return** $[\![\cdot]\!]_\Phi$, $\phi_s = v\tilde{n}.\sigma_s$

This complexity bound, as well as the correctness of the algorithm, are established by Lemma 4. Its proof is given in Appendix A.

Lemma 4. *Algorithm 3 terminates in time $\mathcal{O}(|\phi|^{nvars(\mathcal{R})}\log|\phi|)$, and:*

- ϕ_s *is a saturation of* ϕ *(under* $\mathcal{R}$*);*
- $dom(\llbracket\cdot\rrbracket_\Phi) = dom(\phi_s)$*;*
- *for each* $x \in dom(\phi_s)$*:*
  - $\llbracket x\rrbracket_\Phi \in T(\phi)$*;*
  - $\llbracket x\rrbracket_\Phi$ *is a DAG-representation of a* ϕ*-recipe for* $x\phi_s$*;*
  - $|\llbracket x\rrbracket_\Phi| \in \mathcal{O}(|\phi|)$*;*
- *for each* $v \in dom(\llbracket\cdot\rrbracket_\Phi^{DAG})$*:*
  - *there is a* ϕ_s*-recipe* ζ *for* $\mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(v)$ *such that* $\llbracket v\rrbracket_\Phi^{DAG} = \llbracket\zeta\rrbracket_\Phi$ *is a* ϕ*-recipe for* $\mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(v)$*;*
  - *if* $\mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(v) \in \phi_s[T(\phi_s)]$*, then* $v \in dom(\llbracket\cdot\rrbracket_\Phi^{DAG})$

3.3 Deciding Deducibility and Static Equivalence

Throughout this section, $\mathcal{T}_{\phi,\mathcal{R}}$ and Υ are as in the previous sections, ϕ_s and $\llbracket\cdot\rrbracket_\Phi$ are as output by Algorithm 3, and $\phi' = \nu\tilde{n}.\sigma'$ is a frame such that

$$dom(\phi') = dom(\phi) = \{x_1, \ldots, x_n\}.$$

We assume that $dom(\phi_s) = \{x_1, \ldots, x_{n_s}\}$ and that σ_s is an extension of σ.

Deducibility. In light of Lemma 1, to solve the deducibility problem under $\mathcal{R}$ for a frame ϕ and a term t, it suffices to compute $t\downarrow$ and the saturated frame $\phi_s = \nu\tilde{n}.\sigma_s$ (using Algorithm 3) and then decide whether $t\downarrow \in \phi_s[T(\phi_s)]$. In Appendix A we show that the time complexities of these three computations are, respectively, $\mathcal{O}(|t|\log|t|)$, $\mathcal{O}(|\phi|^{nvars(\mathcal{R})}\log|\phi|)$, and $\mathcal{O}((|t|+|\phi|^2)\log(|t|+|\phi|^2))$, yielding the complexity bound established by Theorem 1.

Theorem 1. *Given a frame* ϕ *and a term* t*, the time complexity of deciding whether* $\phi \vdash_{\mathcal{R}} t$ *is at most*

$$\mathcal{O}((|t|+|\phi|^2)\log(|t|+|\phi|^2) + |\phi|^{nvars(\mathcal{R})}\log|\phi|).$$

Static Equivalence. Algorithm 4 summarizes our procedure for deciding static equivalence. Note that some of the operations performed by this algorithm must use the DAG-representation of terms to ensure polynomial-time complexity. To keep the presentation concise and clean, we refer to Appendix A for the exposition of such details.

The first loop (lines 5–8) tests whether syntactical equality between terms yielded by two distinct ϕ-recipes under ϕ implies that these two recipes yield equationally equal terms under ϕ'. The condition in lines 8–10 tests whether all pairs

Algorithm 4 Decision procedure for static equivalence.

Input: two frames $\phi = \upsilon\tilde{n}.\sigma$ and $\phi' = \upsilon\tilde{n}.\sigma'$
such that $dom(\phi) = dom(\phi') = \{x_1, \ldots, x_n\}$
Output: **true** if $\phi \approx^s_{\mathcal{R}} \phi'$ and **false** otherwise

1: **compute** $\mathcal{T}_{\phi,\mathcal{R}}$, $[\![\cdot]\!]^{DAG}_{\Phi}$, rw, $[\![\cdot]\!]_{\Phi}$ and ϕ_s
2: **choose** $\pi : \{1, \ldots, n_s\} \rightarrowtail \{1, \ldots, n_s\}$
s.t. $1 \leq i < j \leq n_s \Rightarrow |x_{\pi(i)}\phi_s| \leq |x_{\pi(j)}\phi_s|$
3: **for all** $k \in \{1, \ldots, n_s\}$
4: $\phi_{s,k} \leftarrow \upsilon\tilde{n}.\left\{x_{\pi(1)} \mapsto x_{\pi(1)}\phi_s, \ldots, x_{\pi(k)} \mapsto x_{\pi(k)}\phi_s\right\}$
5: **for all** $k \in \{1, \ldots, n_s\}$
6: **if** $x_{\pi(k)}\phi_s \in \phi_s[T(\phi_{s,k-1})]$ **then**
7: **choose** $\zeta \in T(\phi_{s,k-1})$ **s.t.** $x_{\pi(k)}\phi_s = \zeta\phi_s$
8: **if** $[\![x_{\pi(k)}]\!]_{\Phi}\phi' \not\approx_{\mathcal{R}} [\![\zeta]\!]_{\Phi}\phi'$ **then return false**
9: **for all** $v \in dom(\mathsf{rw})$
10: **if** $([\![v]\!]^{DAG}_{\Phi})\phi' \not\approx_{\mathcal{R}} ([\![\mathsf{rw}(v)]\!]^{DAG}_{\Phi})\phi'$
11: **then return false**
12: Repeat once lines 1–11, exchanging ϕ and ϕ'
13: **return true**

of recipes representing instances of the left and right-hand sides of a rule under ϕ represent equal terms (modulo $\mathcal{R}$) also under ϕ'. If either of the two loops outputs **false** then the two frames are not statically equivalent. Otherwise, we conclude that all equalities (between recipes, modulo $\mathcal{R}$) satisfied by ϕ are also satisfied by ϕ'. Repeating the procedure exchanging the roles of ϕ and ϕ', allows one to decide whether $\phi \approx^s_{\mathcal{R}} \phi'$. The correctness of this procedure and its time complexity analysis are provided by Theorem 2.

Theorem 2. *Algorithm 4 decides whether $\phi \approx^s_{\mathcal{R}} \phi'$ in time*

$$\mathcal{O}((|\phi| + |\phi'|)^{\max(nvars(\mathcal{R}),2)} \log(|\phi| + |\phi'|)).$$

Here, $\mathcal{O}((|\phi|+|\phi'|)^2 \log(|\phi|+|\phi'|))$ is an upper bound for the time complexity of the first loop (lines 5–8) of Algorithm 4, while $\mathcal{O}((|\phi| + |\phi'|)^{nvars(\mathcal{R})} \log(|\phi| + |\phi'|))$ is an upper bound for the time complexity of the second (lines 9–11).

3.4 Comparison With Existing Algorithms

Our algorithms compare favorably to other existing algoritmhs. [5] presents the first proof that deduction and static equivalence under subterm convergent equational theories can be decided in polynomial-time. However, efficiency is not their main concern and it is not surprising that our algorithms have much better asymptotic time complexity. For example, for the theory $\approx_{\mathcal{DY},d}$, the time complexities of our algorithms are

$$\mathcal{O}((|\phi|^2 + |t|) \log(|\phi|^2 + |t|)$$

for deducibility, and

$$\mathcal{O}((|\phi| + |\phi'|)^2 \log(|\phi| + |\phi'|))$$

for static equivalence. In contrast, our best estimates for the time complexity of the algorithms in [5] are $\mathcal{O}(|\phi|^{10} + |t|^4)$ and $\mathcal{O}((|\phi| + |\phi'|)^{15})$ for the same problems.

The time complexity of the YAPA tool [37] is not polynomial, as it uses a straightforward representation of terms, as opposed to DAGs. Furthermore, our saturation technique is also more efficient: in YAPA, for each (n, p, q)-decomposition of the left-hand side of a rewrite rule and each assignment of the first $n + p$ parameters to recipes in the frame, it may be necessary to compute the normal form of a term t. We are not aware of any general algorithms for this task that have a better time complexity than $\mathcal{O}(|t|^4)$ (see discussion below). If we denote by $Y(\mathcal{R})$ the greatest value of $n + p$ for all (n, p, q)-decompositions of rewriting rules in $\mathcal{R}$, we obtain a time complexity of $\mathcal{O}(|\phi|^{Y(\mathcal{R})+4})$ for YAPA's saturation procedure; this is significantly worse than the time complexity of $\mathcal{O}(|\phi|^{nvars(\mathcal{R})} \log |\phi|)$ achieved by our algorithm (note that we always have $nvars(\mathcal{R}) \leq Y(\mathcal{R})$). For the rewriting system $\mathcal{R}_{\mathcal{DY},d}$ we obtain an estimated time complexity of $\mathcal{O}(|\phi|^7)$ for the saturation procedure in YAPA and $\mathcal{O}(|\phi|^2 \log |\phi|)$ for ours. Note that this estimate assumes that DAGs are implemented; the exact implementation of DAGs and the algorithms to compute normal forms may affect the time complexity of the procedure. It may also be possible to provide better bounds on the number of recipes for which we need to perform this reduction to a normal form.

Our saturation procedure is also more efficient than that implemented by the KISS tool. In this tool, the rule Narrowing generates a number of deduction facts for each rewriting rule in $\mathcal{R}$. If we denote by $K(\mathcal{R})$ the maximum number of side conditions in deduction facts generated by any rewriting rule in $\mathcal{R}$, we again have that $nvars(\mathcal{R}) \leq K(\mathcal{R})$: for example, $K(\mathcal{R}_{\mathcal{DY}}) = 3$. The terms in these side-conditions must then be instantiated (via the rule $\mathsf{F} - \mathsf{Solving}$) with terms that are heads of some deduction fact. There are at least $\mathcal{O}(|\phi|)$ such terms, which implies that the saturated frame may have $\mathcal{O}(|\phi|^{K(\mathcal{R})})$ deduction facts. Testing the premise of the rules $\mathsf{F} - \mathsf{Solving}$, $\mathsf{E} - \mathsf{Solving}$, and Unifying requires selecting two deduction facts from the frame and performing an operation with linear-time complexity. Since there are $\mathcal{O}(|\phi|^{2K(\mathcal{R})})$ such pairs, we obtain a time complexity of at least $\mathcal{O}(|\phi|^{2K(\mathcal{R})+1})$. For the rewriting system $\mathcal{R}_{\mathcal{DY},d}$, this amounts to a time complexity of $\mathcal{O}(|\phi|^7)$ for KISS, in contrast to the time complexity of $\mathcal{O}(|\phi|^2 \log |\phi|)$ for our algorithms. Here it may also be possible to improve this time complexity bound, for example by limiting the number of pairs of rules that must be tested.

Finally, we note that all the algorithms discussed here require deciding the equality of two terms t and t' under the equational theory. Our algorithms rely on the subterm convergence of the rewriting system to perform this task with time complexity $\mathcal{O}((|t| + |t'|) \log(|t| + |t'|))$. This constitutes a marked advantage over algorithms for more general rewriting systems, for which we are not aware of any algorithm improving the time complexity of $\mathcal{O}((|t| + |t'|)^4)$ achieved in [5].

A very efficient algorithm for the deducibility problem is given in [75]. For the

rewriting systems $\mathcal{R}$ and $\mathcal{R}_{\mathcal{DY},d}$, this algorithm achieves linear-time complexity.

Table 3.1 presents a summary of (our estimations of) the theoretical time complexities of these algorithms under a general subterm convergent rewriting system $\mathcal{R}$. Table 3.2 considers instead the rewriting system $\mathcal{R}_{\mathcal{DY},d}$. For conciseness, we use $nvars^*(\mathcal{R})$ to denote $\max\{2, nvars(\mathcal{R})\}$. In the case of YAPA we note that the worst-case theoretical time complexity is exponential, and provide also our best estimates for its time complexity if DAG-representation of terms is implemented.

Table 3.1: Theoretical time complexities of FAST, YAPA, KISS under $\mathcal{R}$.

	$\phi \vdash_{\mathcal{R}} t$	$\phi \approx^s_{\mathcal{R}} \phi'$
FAST	$(\lvert t\rvert + \lvert\phi\rvert^{nvars^*(\mathcal{R})}) \log(\lvert t\rvert + \lvert\phi\rvert)$	$(\lvert\phi\rvert + \lvert\phi'\rvert)^{nvars^*(\mathcal{R})} \log(\lvert\phi\rvert + \lvert\phi'\rvert)$
YAPA	exponential / $\lvert\phi\rvert^{Y(\mathcal{R})+4} + \lvert t\rvert^4$	exponential / $(\lvert\phi\rvert + \lvert\phi'\rvert)^{Y(\mathcal{R})+8}$
KISS	$\lvert\phi\rvert^{2\cdot K(\mathcal{R})+1} + \lvert t\rvert^4$	$(\lvert\phi\rvert + \lvert\phi'\rvert)^{2\cdot K(\mathcal{R})+5}$

Table 3.2: Theoretical time complexities of FAST, YAPA, KISS and [5] under $\mathcal{R}_{\mathcal{DY},d}$.

	$\phi \vdash_{\mathcal{R}_{\mathcal{DY},d}} t$	$\phi \approx^s_{\mathcal{R}_{\mathcal{DY},d}} \phi'$
FAST	$(\lvert\phi\rvert^2 + \lvert t\rvert) \log(\lvert\phi\rvert^2 + \lvert t\rvert)$	$(\lvert\phi\rvert + \lvert\phi'\rvert)^3 \log(\lvert\phi\rvert + \lvert\phi'\rvert)$
YAPA	exponential / $\lvert\phi\rvert^7 + \lvert t\rvert^4$	exponential / $(\lvert\phi\rvert + \lvert\phi'\rvert)^{11}$
KISS	$\lvert\phi\rvert^7 + \lvert t\rvert^4$	$(\lvert\phi\rvert + \lvert\phi'\rvert)^{11}$
[5]	$\lvert\phi\rvert^{10} + \lvert t\rvert^4$	$(\lvert\phi\rvert + \lvert\phi'\rvert)^{15}$

3.5 Algorithm Performance

Our algorithm for message deducibility and static equivalence is implemented in the FAST tool.

We have considered several families of interesting and practically relevant examples to compare the performance of our algorithm with YAPA and KISS. The results show great disparities in the performance of the three algorithms. Neither KISS nor YAPA show a clear advantage over the other: depending on the example, either algorithm may perform significantly faster than the other. As expected from the time complexity results discussed in the previous section, FAST generally performs much better than either of these algorithms, particularly for static equivalence. Even for artificial examples designed to degrade its performance, FAST still compares favorably to other algorithms: for the message deducibility problem, it is either faster, or slower by only a small constant, and it remains the most efficient algorithm for static equivalence by a significant margin. This constitutes a significant advantage since the problematic cases for YAPA and KISS degrade these algorithms' performances dramatically.

All our tests were performed using a computer with an Intel Core 2 Duo processor running at 2.53GHz and with 4GB memory. In all our static equivalence

tests, we consider two equal frames. Similarly, in all our deduction tests, the input term is a secret that does not occur in the range of the substitution of the input frame. Therefore, the result is positive in all static equivalence tests and negative in all deducibility tests. This does not affect the algorithm's performance significantly, as both frames still have to be saturated in all implementations — that is, deducible subterms must still be added to the saturation, and the sets of equations which must be tested to check for static equivalence must still be generated. Static equivalence takes a slightly longer time in this case because all equations must be checked rather than stopping as soon as a counter-example is found. However, this does not change the asymptotic behavior of the algorithms.

The implementation and our benchmarks are available for download at [3].

3.5.1 Chained Keys

This family of tests uses the standard Dolev-Yao signature $\Sigma^{\mathcal{DY}}$ and corresponding rewriting system $\mathcal{R}_{\mathcal{DY}}$ introduced in Chapter 2. For $n \in \mathbb{N}$, we define the frame

$$\phi_n^{ck} = \upsilon \tilde{n}_n^{ck}.\sigma_n^{ck},$$

where

$$\tilde{n}_n^{ck} = \{k, k_0, \ldots, k_n\}$$

and

$$\sigma = \left\{x_1 \mapsto \{|k_0|\}_{k_1}, \ldots, x_n \mapsto \{|k_{n-1}|\}_{k_n}, x_{n+1} \mapsto k_n\right\}.$$

For each parameter n, the deduction problem is to decide whether $\phi_n^{ck} \vdash_{\mathcal{DY}} k$, and the static equivalence problem is to decide whether $\phi_n^{ck} \approx^s_{\mathcal{R},\mathcal{DY}} \phi_n^{ck}$.

FAST has a much better performance than both YAPA and KISS for these examples. YAPA also performs much better than KISS. Tables 3.3 and 3.4 illustrate these relationships.

Table 3.3: Performance on *chained keys* for deduction (time in ms)

Parameter	50	100	200	500	1000	2000	5000
FAST	11	20	40	143	224	474	1526
KISS	259	1730	12655	288606	$>$ 300000	$>$ 300000	$>$ 300000
YAPA	31	108	415	4624	11297	62457	$>$ 300000

Table 3.4: Performance on *chained keys* for static equivalence (time in ms)

Parameter	50	100	200	500	750	1500	2500
FAST	20	41	88	247	424	1020	1546
KISS	1341	12185	127828	$>$ 300000	$>$ 300000	$>$ 300000	$>$ 300000
YAPA	143	744	5516	18467	44451	197648	$>$ 300000

3.5.2 Chained Encryptions

This family of examples uses the same standard Dolev-Yao signature $\Sigma^{\mathcal{DY}}$ and rewriting system $\mathcal{R}_{\mathcal{DY}}$ as Section 3.5.1. It has been previously studied in [37]: it exploits the fact that YAPA does not implement DAG-representations of terms to cause the runtime of the algorithm to explode exponentially.

For each $n > 0$, we define t_n recursively by

$$t_i = \begin{cases} \{|k_0|\}_{k_1} & \text{if } i = 1 \\ \{|\langle t_{i-1}, k_{i-1}\rangle|\}_{k_i} & \text{if } i > 1 \end{cases}.$$

We also define

$$\phi^{ce_n} = \tilde{n}^{ce_n}.\sigma^{ce_n},$$

where

$$\tilde{n}^{ce_n} = \{k, k_0, \ldots, k_n\}$$

and

$$\sigma^{ce_n} = \{x_1 \mapsto t_n, x_2 \mapsto k_n\}.$$

The deduction problem (for parameter n) is to decide whether $\phi^{ce_n} \vdash_{\mathcal{DY}} k$; the static equivalence problem is deciding whether $\phi^{ce_n} \approx^s_{\mathcal{R},\mathcal{DY}} \phi^{ce_n}$.

Since KISS implements a DAG representation of terms, its performance is significantly better than that of YAPA. FAST is much faster than both these algorithms, as shown in Tables 3.5 and 3.6.

Table 3.5: Performance on *Chained Encryption* for deduction (time in ms)

Parameter	15	20	22	25	40	60	80
FAST	10	15	12	20	24	40	65
KISS	473	1560	2698	3592	29881	198720	$>$ 300000
YAPA	1447	66824	304027	$>$ 300000	$>$ 300000	$>$ 300000	$>$ 300000

Table 3.6: Performance on *Chained Encryption* for static equivalence (time in ms)

Parameter	12	15	17	18	20	50	75
FAST	11	20	23	23	20	52	90
KISS	656	1284	2083	2900	4300	156797	$>$ 300000
YAPA	3456	43707	211627	$>$ 300000	$>$ 300000	$>$ 300000	$>$ 300000

3.5.3 Composed Keys

We continue using the standard Dolev-Yao signature $\Sigma^{\mathcal{DY}}$ and rewriting system $\mathcal{DY}$. For $n, s, i \in \mathbb{N}$, define $t^i_{n,s}$ recursively by

$$t^0_{n,s} = \{|\langle k_{2s-1}, k_{2s-2}\rangle|\}_{\langle k_{2s}, k_{s2+1}\rangle}$$

and

$$t^i_{n,s} = \left\{ \left| \langle t^{i-1}_{n,s} \langle k_{2s+1+2i(n-1)}, k_{2s+2i(n-1)} \rangle, \right\rangle | \right\}_{\langle k_{2s+2+2i(n-1)}, k_{2s+3+2i(n-1)} \rangle}.$$

For $k > 0$, we consider the frame $\phi^c_k = \upsilon \tilde{n}^c_n.\sigma^c_n$, where

$$\tilde{n}^c_n = \{k, k_0, \ldots, k_{2n^2+1}\}$$

and

$$\sigma^c_n = \left\{ x_1 \mapsto t^{n-1}_{n,1}, \ldots, x_n \mapsto t^{n-1}_{n,n}, x_{n+1} \mapsto k_{2n^2}, x_{n+2} \mapsto k_{2n^2+1} \right\}.$$

The deduction problem corresponding to parameter n considered in our tests is to decide whether $\phi^c_n \vdash_{\mathcal{DY}} k$. The static equivalence problem corresponding to parameter n is to decide whether $\phi^c_n \approx^s_{\mathcal{R},\mathcal{DY}} \phi^c_n$.

This family of examples is particularly challenging because the decryption keys are pairs of secrets. At each point of the algorithm's execution, decrypting the right message yields a pair of previously unknown secrets. This pair may then be used to compose the next decryption key by exchanging the order of the terms in the pair. As illustrated in Tables 3.7 and 3.8, the difference in FAST's performance is particularly marked in this example. KISS also performs much better than YAPA.

Table 3.7: Performance on *composed* for deduction (time in ms)

Parameter	3	4	5	7	9	10	20
FAST	7	11	17	34	61	126	945
KISS	138	867	3760	46369	245207	>300000	>300000
YAPA	158	34118	>300000	>300000	>300000	>300000	>300000

Table 3.8: Performance on *composed* for static equivalence (time in ms)

Parameter	3	4	5	6	8	10	20
FAST	12	21	28	48	92	148	1635
KISS	469	2625	10428	252000	>300000	>300000	>300000
YAPA	936	157358	>300000	>300000	>300000	>300000	>300000

3.5.4 Denning-Sacco Shared Key Protocol

The Denning-Sacco symmetric key protocol [95] is used to establish session keys in a network with a single server and multiple agents. Each agent shares a (secret) symmetric key with the server, but there are no shared keys between agents. In Alice&Bob notation, the protocol is as follows.

1. $\mathsf{A} \to \mathsf{S}$: A, B
2. $\mathsf{S} \to \mathsf{A}$: $\{\mathsf{A}, K_{\mathsf{A},\mathsf{B}}, T, \{K_{\mathsf{A},\mathsf{B}}, \mathsf{A}, T\}_{K_{\mathsf{S},\mathsf{B}}}\}_{K_{\mathsf{S},\mathsf{A}}}$
3. $\mathsf{A} \to \mathsf{B}$: $\{K_{\mathsf{A},\mathsf{B}}, \mathsf{A}, T\}_{K_{\mathsf{S},\mathsf{B}}}$

Here, A and B are two participants, and S is the server. A requests from the server a session key to communicate with B. The server generates a new session key, $K_{\mathsf{A},\mathsf{B}}$, and sends it to A, encrypted with the (symmetric) key shared between A and S. This message also contains a timestamp T, used to determine the validity of the new session key, and the ticket $\{K_{\mathsf{A},\mathsf{B}}, \mathsf{A}, T\}_{K_{\mathsf{S},\mathsf{B}}}$. A then forwards this ticket to B, who can decrypt it using the key $K_{\mathsf{S},\mathsf{B}}$ shared between B and S, to obtain the new session key $K_{\mathsf{A},\mathsf{B}}$, the name A of the intended communication partner, and the time T of the request.

This example uses the result of executing multiple sessions of the Denning-Sacco protocol. For the parameter n we assume a network with $3n$ participants, each of which initiates one session with each other participant. We assume that one third of the shared keys between the server and the agents are compromised, i.e., available to the attacker.

We will once again use the signature $\Sigma^{\mathcal{DY}}$ and the rewriting system $\mathcal{DY}$ from Section 3.5.1. For each parameter n and each integers $i, j \in \{1, \ldots, 3n\}$ such that $i \neq j$, we define:

- $\sigma_n^1 = \left\{x_i^1 \mapsto K_{\mathsf{S},i} \mid i \in \{1, \ldots, n\}\right\}$;
- $\sigma_{n,i,j}^2 = \left\{x_{i,j}^2 \mapsto \langle \mathsf{A}_i, \mathsf{A}_j \rangle\right\}$;
- $\sigma_{n,i,j}^3 = \left\{x_{i,j}^3 \mapsto \left\{\left|\langle \mathsf{A}_j, \langle K_{i,j}, \langle T_{i,j}, \{|\langle K_{i,j}, \langle \mathsf{A}_i, T_{i,j}\rangle\rangle|\}_{K_{\mathsf{S},j}}\rangle\rangle\right|\right\}_{K_{\mathsf{S},i}}\right\}$;
- $\sigma_n^4 = \left\{x_{i,j}^4 \mapsto \{|\langle K_{i,j}, \langle \mathsf{A}_i, T_{i,j}\rangle\rangle|\}_{K_{\mathsf{S},j}}\right\}$.

We also define

$$\tilde{n}_n^{ds} = \{K_{i,j}, T_{i,j}, K_{\mathsf{S},i} \mid i, j \in \{1, \ldots, 3n\}\}$$

and

$$\sigma_n^{ds} = \sigma_n^1 \cup \left(\bigcup_{\substack{i,j=1 \\ i \neq j}}^{3n} \left(\sigma_{n,i,j}^2 \cup \sigma_{n,i,j}^3 \cup \sigma_{n,i,j}^4\right) \right).$$

Finally, the frame we consider is then given by $\phi_n^{ds} = \tilde{n}_n^{ds}.\sigma_n^{ds}$.

Here, σ_n^1 represents the keys compromised by the attacker and σ_n^2, σ_n^3, and σ_n^4 represent the messages exchanged as part of the execution of the first, second, and third steps of the protocol, respectively. The deduction problem is to decide whether $\phi_n^{ds} \vdash_{\mathcal{DY}} K_{\mathsf{S},3n}$ and the static equivalence problem is to decide whether $\phi_n^{ds} \approx^s_{\mathcal{R},\mathcal{DY}} \phi_n^{ds}$.

YAPA performs noticeably better than KISS in this example. FAST, as before, is significantly faster than both. The results are shown in Tables 3.9 and 3.10.

Table 3.9: Performance on *Denning-Sacco* for deduction (time in ms)

Parameter	7	9	11	12	14	24
FAST	768	1336	2588	2239	5083	20073
KISS	30195	91743	232093	> 300000	> 300000	> 300000
YAPA	10320	30732	68845	74298	172249	> 300000

Table 3.10: Performance on *Denning-Sacco* for static equivalence (time in ms)

Parameter	3	5	7	9	11	13	20
FAST	181	585	1281	2300	6598	7507	24614
KISS	1219	8543	34726	158717	> 300000	> 300000	> 300000
YAPA	446	2836	12300	52506	134391	269781	> 300000

3.5.5 Projections

In this family of examples, for parameter $n \in \mathbb{N}$, we use the signature $\Sigma^{proj,n}$, where $\Sigma_1^{proj,n} = \{h, \pi_1, \ldots, \pi_n\}$, and $\Sigma_n^{proj,n} = \{\langle\cdot, \ldots, \cdot\rangle\}$. The rewriting system $proj_n$ is given by

$$proj_n = \{\pi_1(\langle h(x_1), \ldots, h(x_n)\rangle) \to x_1, \ldots, \pi_n(\langle h(x_1), \ldots, h(x_n)\rangle) \to x_n\}.$$

For each parameter $n \in \mathbb{N}$, we will consider the frame $\phi_n^{proj} = \tilde{n}_n^{proj}.\sigma_n^{proj}$, where

$$\tilde{n}_n^{proj} = \{k, k_1, \ldots, k_n\}$$

and

$$\sigma_n^{proj} = \{x_1 \mapsto \langle h(k_1), \ldots, h(k_n)\rangle\}.$$

The deduction problem is to decide whether $\phi_n^{proj} \vdash_{proj_n} k$ and the static equivalence problem is to decide whether $\phi_n^{proj} \approx^s_{\mathcal{R},proj_n} \phi_n^{proj}$.

In this example, FAST is again much faster than both KISS and YAPA; KISS performs much better than YAPA. The results are shown in Tables 3.11 and 3.12.

Table 3.11: Performance on *Projections* for deduction (time in ms)

Parameter	10	13	17	50	200	500
FAST	7	11	37	159	8657	234912
KISS	17	55	89	1493	171994	> 300000
YAPA	989	20976	> 300000	> 300000	> 300000	> 300000

3.5.6 FAST Worst Case

In this family of examples, for the parameter n, we use the signature Σ^{wc}, where $\Sigma_1^{wc} = \{f\}$ and $\Sigma_n^{wc} = \{h\}$. The rewriting system is given by the set

$$wc_n = \{h(f(x_1), \ldots, f(x_n)) \to x_1\}.$$

Table 3.12: Performance on *Projections* for static equivalence (time in ms)

Parameter	5	6	50	100	150	200
FAST	6	14	323	3202	10498	27426
KISS	13	41	8379	85700	>300000	>300000
YAPA	9477	319911	>300000	>300000	>300000	>300000

We consider the frame $\phi_n^{wc} = \tilde{n}_n^{wc}.\sigma_n^{wc}$, where

$$\tilde{n}_n^{wc} = \{k, k_1, \ldots, k_n\}$$

and

$$\sigma_n^{wc} = \{x_1 \mapsto f(k_1), \ldots, x_n \mapsto f(k_n)\}.$$

The deduction problem is to decide whether $\phi_n^{wc} \vdash_{wc_n} k$ and the static equivalence problem is to decide whether $\phi_n^{wc} \approx^s_{\mathcal{R},wc_n} \phi_n^{wc}$.

This example is challenging because, to saturate this frame, FAST must instantiate each element of the tuple with each of the secret names. Therefore, the asymptotic time complexity of FAST for this family is $\mathcal{O}(n^n)$. Note that this does not contradict the fact that, for a given rewriting system, FAST has polynomial-time complexity; the exponential-time complexity results from the fact that the size of the rewriting system itself increases with the parameter n.

Table 3.13: Performance on *Worst Case* for deduction (time in ms)

Parameter	3	4	5	6
FAST	9	72	1192	32487
KISS	10	47	866	21446
YAPA	11	161	6607	>300000

Table 3.14: Performance on *Worst Case* for static equivalence (time in ms)

Parameter	3	4	5	6
FAST	15	142	2199	56312
KISS	16	146	2125	69533
YAPA	16	297	8862	>300000

None of the existing algorithms perform well on this example: FAST's performance is comparable to that of KISS and YAPA performs significantly worse. This is illustrated in Tables 3.13 and 3.14.

3.5.7 Non-linear Terms

It is interesting to note that FAST's complexity depends chiefly on the number of different variables in the rewriting system. Therefore, it's performance is not significantly affected if the left-hand sides of rewrite rules are non-linear. This is not the case for the other algorithms, whose performance degrades when the

complexity of the terms in the rewriting system increases, even when the number of variables remains the same. Tables 3.15 and 3.16 illustrate this point. Here, the rewriting system considered is

$$wc2_n = \{h(f(x_1), f(x_1), \ldots, f(x_n), f(x_n)) \to x_1\}\,.$$

The frames and problems considered here are the same as in the previous section.

Table 3.15: Performance on *Worst Case 2* for deduction (time in ms)

Parameter	2	3	4	5	6
FAST	7	9	148	1567	43282
KISS	9	99	4381	183236	> 300000
YAPA	45	> 300000	> 300000	> 300000	> 300000

Table 3.16: Performance on *Worst Case 2* for static equivalence (time in ms)

Parameter	2	3	4	5	6
FAST	4	17	396	6197	47937
KISS	10	292	16135	> 300000	> 300000
YAPA	56	> 300000	> 300000	> 300000	> 300000

Chapter 4

Deciding Trace Equivalence

In this chapter we describe our procedure for deciding the equivalence of constraint systems, which can be used to decide the trace equivalence of bounded simple processes under equational theories generated by convergent rewriting systems for which a finitary unification algorithm exists.

4.1 Basic Definitions

In this section we introduce the notions used throughout this chapter.

4.1.1 Generalized Term Algebra

Our algorithm uses a notion of frame and recipe more general than the standard notion used in the applied-pi calculus and described in Chapter 2.

To define our notion of frame and recipe, we assume fixed two countably infinite, disjoint sets $\mathcal{H}$ and $\mathcal{X}_R$ of *handles* and *recipe variables*, respectively. We assume fixed a total order $\prec_{\mathcal{X}_R}$ on recipe variables.

Note that $\mathcal{X}_R \cap \mathcal{X} = \emptyset$; that is, the set $\mathcal{X}_R$ of recipe variables is also disjoint from the set $\mathcal{X}$ of variables. To make this distinction clearer throughout this chapter, we refer to variables as *term variables*, and write $\mathcal{X}_T$ for the set of term variables.

A *handle substitution* is a substitution $\sigma \colon \mathcal{H} \nrightarrow T_\Sigma(\mathcal{N} \cup \mathcal{X}_R)$. If $t \in T_\Sigma(X)$, we define the set $rvars(t)$ of recipe variables occurring in t analogously to the definition of $names(t)$. In this chapter we write $tvars(t)$ instead of $vars(t)$ to refer to the set of term variables occurring in t.

A *generalized frame* is a pair $\Phi = (\tilde{n}, \sigma)$, written $\nu\tilde{n}.\sigma$, where $\tilde{n} \subseteq \mathcal{N}$ is a finite set of names (the restricted names) and σ is a handle substitution. We associate a frame with its corresponding substitution. Thus, if $\Phi = \nu\tilde{n}.\sigma$, we define the domain of Φ as $dom(\Phi) = dom(\sigma)$ and, if $h \in dom(\sigma)$, we define $h\Phi = h\sigma$. We assume that $ran(\Phi) \cap \mathcal{X}_R = \emptyset$, i.e., no handle is mapped to a single recipe variable.

In this chapter we will always use generalized frames. Thus, for conciseness, we refer to generalized frames simply as frames.

Generalized recipes. Since we are interested in modelling public functions, we assume that, for each $n \in \mathbb{N}$ and each function symbol $f \in \Sigma_n$, there exists a handle $h_f \in \mathcal{H}$ such that

$$h_f \in dom(\sigma) \qquad \text{and} \qquad h_f\sigma = f(\rho_1, \ldots, \rho_n)$$

for some recipe variables $\rho_1 \prec_{\mathcal{X}_R} \ldots \prec_{\mathcal{X}_R} \rho_n$. We will write Φ_Σ for the frame obtained in this way from the signature Σ.

If Φ is a generalized frame, we define the set $T(\Phi)$ of Φ-*recipes* as the smallest set such that:

- $\mathcal{X}_R \cup (\mathcal{N} \setminus \tilde{n}) \subset T(\Phi)$;
- if $h \in dom(\Phi)$, $|rvars(h\Phi)| = n$, and $\zeta_1, \ldots, \zeta_n \in T(\Phi)$, then

$$h(\zeta_1, \ldots, \zeta_n) \in T(\Phi).$$

If $rvars(h\Phi) = \emptyset$, we write simply h instead of $h()$.

If $\zeta_1, \ldots, \zeta_n$ are Φ-recipes, we also make no distinction between $f(\zeta_1, \ldots, \zeta_n)$ and $h_f(\zeta_1, \ldots, \zeta_n)$. Therefore, we have $T_\Sigma(T(\Phi)) = T(\Phi)$. Note that $T(\Phi)$ could be equivalently defined by

$$T(\Phi) = T_{\Sigma^\Phi}(\mathcal{X}_R \cup (\mathcal{N} \setminus \tilde{n})),$$

where

$$\Sigma^\Phi = \biguplus_{i \in \mathbb{N}} \Sigma_i^\Phi$$

and, for each $i \in \mathbb{N}$,

$$\Sigma_i^\Phi = \{h \in dom(\sigma) \mid |rvars(h\sigma)| = i\}.$$

Handles are used simply as the variables in active substitutions in the applied-pi calculus. Recipes represent the ways that an attacker can build terms from other terms he observes (represented by the σ) without using secret names (represented by $\tilde{n}$). Recipe variables represent *holes* that can be filled with recipes. Thus, replacing recipe variables occurring in a recipe by other recipes yields a new recipe. This implements the idea of representing deducible terms by nesting a finite set of contexts, that is used, e.g., in the KISS algorithm [69].

Recipe substitutions. A substitution $\gamma\colon \mathcal{X}_R \nrightarrow T(\Phi)$, is called a Φ-*recipe substitution*. If $h \in dom(\Phi)$ and $\rho_1 \prec_{\mathcal{X}_R} \ldots \prec_{\mathcal{X}_R} \rho_n$ are the recipe variables occurring in $h\Phi$, we often write (h, γ) (with $\gamma\colon rvars(h\Phi) \to T(\Phi)$) to denote the Φ-recipe $h(\rho_1\gamma, \ldots, \rho_n\gamma)$. Furthermore, we need not require that $dom(\gamma) = rvars(h\Phi)$, by setting

$$(h, \gamma) = (h, \gamma\,|_{rvars(h\Phi)} \circ id(rvars(h\Phi))).$$

A recipe substitution γ is *ground* if no recipe variables occur in the range of γ, i.e., $sub[ran(\gamma)] \cap \mathcal{X}_R = \emptyset$.

Recipe variables can be seen as variables that are instantiated by recipes rather than by terms: indeed, term substitutions are functions $\alpha\colon \mathcal{X}_T \nrightarrow T_\Sigma(\mathcal{N} \cup \mathcal{X}_R)$, while Φ-recipe substitutions are functions $\gamma\colon \mathcal{X}_R \nrightarrow T(\Phi)$.

Extension of handle substitution. If $\Phi = \nu\tilde{n}.\sigma$ is a frame, we define the extension $\hat{\sigma}$ of σ to $T(\Phi)$ by

$$\zeta\hat{\sigma} = \begin{cases} \zeta & \text{if } \zeta \in \mathcal{N} \cup \mathcal{X}_R \\ (h, \sigma)(\hat{\sigma} \circ \gamma) & \text{if } \zeta = (h, \gamma) \end{cases}.$$

We will also abuse notation by using the symbol σ to refer to σ's extension $\hat{\sigma}$ to $T(\Phi)$ and, when ζ is a Φ-recipe, we will often write $\zeta\Phi$ instead of $\zeta\sigma$.

As before, we say that a term t is *(syntactically) constructible from* Φ if $t \in \sigma[T(\Phi)]$. If γ is a Φ-recipe substitution, we define the extension

$$\hat{\gamma}\colon T(\Phi) \to T(\Phi)$$

of γ to $T(\Phi)$ by

$$\zeta\hat{\gamma} = \begin{cases} \zeta\gamma & \text{if } \zeta \in \mathcal{X}_R \\ \zeta & \text{if } \zeta \in \mathcal{N} \\ (h, \hat{\gamma} \circ \delta) & \text{if } \zeta = (h, \delta) \end{cases}.$$

As before, we abuse notation by writing γ to refer to γ's extension $\hat{\gamma}$.

Note that $\Phi \circ \gamma\colon \mathcal{X}_R \to T_\Sigma(\mathcal{N} \cup \mathcal{X}_R)$.

Equational unification. Equational unification is a well-known problem [17] that has been used extensively in security protocol analysis [102]. We now introduce it in the context of our framework.

A *unification problem* is a set

$$\mathcal{U} = \{t_1^1 \stackrel{?}{=} t_1^2, \ldots, t_n^1 \stackrel{?}{=} t_n^2\}$$

where, for each $i \in \{1, \ldots, n\}$,

$$t_i^1, t_i^2 \in T_\Sigma(\mathcal{N} \cup \mathcal{X}_T \cup \mathcal{X}_R)$$

are terms with term variables and recipe variables. A *classical substitution* is a substitution

$$\varrho\colon \mathcal{X}_T \cup \mathcal{X}_R \nrightarrow T_\Sigma(\mathcal{N} \cup \mathcal{X}_T \cup \mathcal{X}_R).$$

A *finitary unification algorithm for* $\approx_{\mathcal{R}}$ is an algorithm EUnif with the following properties:

(1) If $\mathcal{U}$ is a unification problem, then $\mathsf{EUnif}(\mathcal{U})$ is a finite set of classical substitutions ϱ such that $t\varrho \approx_{\mathcal{R}} t'\varrho$ for any $(t \stackrel{?}{=} t') \in \mathcal{U}$ (i.e., ϱ solves $\mathcal{U}$ modulo $\approx_{\mathcal{R}}$).

(2) For any other classical solution ϱ' of $\mathcal{U}$ modulo $\approx_{\mathcal{R}}$, there is $\varrho \in \mathsf{EUnif}(\mathcal{U})$ and a classical substitution ϱ'' such that $\varrho' \approx_{\mathcal{R}} \varrho'' \circ \varrho$.

Our procedure assumes that such a finitary unification algorithm EUnif exists for $\approx_{\mathcal{R}}$. Finitary unification algorithms always exist for subterm convergent theories, since such theories have the finite variant property (see [103] and the boundedness theorem of [74]).

Recipe variable instantiations. Fixed a frame Φ, a recipe variable ρ, and a Φ-recipe ζ, we define the set $\zeta[\rho]$ of *instantiations of* ρ *in* ζ as the smallest set such that:

- if $\zeta = \rho$, then $\zeta[\rho] = \{\rho\}$;
- if $\zeta = h(\zeta_1, \ldots, \zeta_n)$, $rvars(h\Phi) = \{\rho_1, \ldots, \rho_n\}$, $\rho_1 \prec_{\mathcal{X}_R} \ldots \prec_{\mathcal{X}_R} \rho_n$, and $\rho_i = \rho$, then $\zeta_i \in \zeta[\rho]$;
- if $\zeta = h(\zeta_1, \ldots, \zeta_n)$, then $\zeta_i[\rho] \subseteq \zeta[\rho]$ for all $i \in \{1, \ldots, n\}$.

If γ is a Φ-recipe substitution, we define the set $\gamma[\rho]$ of instantiations of ρ in γ by

$$\gamma[\rho] = \bigcup_{\zeta \in ran(\gamma)} \zeta[\rho].$$

Example 7. Continuing our running example, the attacker's knowledge corresponding to the protocol run described in Example 5 is described by the frame

$$\begin{aligned}\Phi = \quad & \nu\tilde{n}'.\Phi_{\Sigma^{\mathcal{PY}}} \cup \left\{h_1 \mapsto \left\{\left|\{K\}^{r'}_{K\mathsf{A}_{\mathsf{pub}}}\right|\right\}_{K\mathsf{AB}}\right\} \\ & \cup \left\{h_2 \mapsto \left\{\{\rho\}^{-1}_{K\mathsf{A}_{\mathsf{priv}}}\right\}^{r}_{K\mathsf{C}_{\mathsf{pub}}}, h_3 \mapsto \{|M|\}_K\right\} \\ & \cup \{h_4 \mapsto \{|\langle M, N\rangle|\}_K, h_5 \mapsto \{|N|\}_K\},\end{aligned}$$

where $\tilde{n}' = \{K\mathsf{A}, K\mathsf{B}, K\mathsf{AB}, r, r', K, M, N\}$. Here, ρ represents the message sent by the attacker on the channel c_1^1 during the execution of the first protocol. The handle h_2 in essence gives the attacker the possibility of obtaining an asymmetric encryption of a given secret with the public key of C given an asymmetric encryption of that secret with the public key of A. To decide security against offline guessing, we must thus consider the frames $\Phi_w = \nu\tilde{n}.\Phi \cup \{h_6 \mapsto w\}$ and $\Phi_s = \nu\tilde{n}.\Phi \cup \{h_6 \mapsto K\mathsf{AB}\}$, where $\tilde{n} = \tilde{n}' \cup \{w\}$.

Consider the recipes

$$\zeta_1 = \{|h_3|\}^{-1}_{\left\{h_2(\{|h_1|\}^{-1}_{h_6})\right\}^{-1}_{K\mathsf{C}_{\mathsf{priv}}}}$$

and

$$\zeta_2 = \pi_1(\{|h_4|\}^{-1}_{h_2(\{|h_1|\}^{-1}_{h_6})}).$$

We have $\zeta_1, \zeta_2 \in T(\Phi_s)$. This is not an accurate description of the attacker's capabilities: namely, the ρ occurring in h_2 is instantiated with $\{|h_1|\}^{-1}_{h_6}$. However, the handles $h_2, \ldots, h_6$ may not be used to instantiate the recipe ρ, as the attacker must send ρ before all the other messages are exchanged and the symmetric key $K\mathsf{AB}$ is revealed to him. This is modelled by means of *deducibility constraints*, introduced below.

4.1.2 Constraint Systems

Deducibility constraints. Let $\Phi = \nu\tilde{n}.\sigma$ be a frame. A *deducibility constraint sequence* (DCS) for Φ is a finite sequence $(\rho_1, K_1), \ldots, (\rho_n, K_n)$ such that, whenever $1 \leq i < j \leq n$:

- $\rho_i, \rho_j \in \mathcal{X}_R$ and $\rho_i \neq \rho_j$;
- $K_i \subseteq K_j \subseteq dom(\Phi)$;
- if $h \in K_j$, then $\rho_j \notin rvars(h\Phi)$.

The first condition implies that a DCS may be interpreted as a function

$$D\colon \mathcal{X}_R \nrightarrow \mathcal{P}(dom(\Phi))$$

with finite domain. In light of this, we denote by $dom(D)$ the set of recipe variables ρ for which there exists K such that (ρ, K) occurs in D. If $\rho \in dom(D)$, we also write $D(\rho)$ for the (unique) set K such that (ρ, K) occurs in D. If $i \in \{1, \ldots, n\}$, we denote by D_i the DCS $(\rho_1, K_1), \ldots, (\rho_i, K_i)$, and denote by Φ_i the frame $\nu\tilde{n}.\sigma\,|_{K_i}$.

Intuitively, a DCS models an active attacker's deduction capabilities after the execution of a given symbolic trace in which all interleavings and branchings have been determined. For each $i \in \{1, \ldots, n\}$, the recipe variable ρ_i represents a message sent by the attacker over the network at time i, and K_i is a set of handles that describe the attacker's deduction capabilities at that point. The message ρ_i may then be used by other agents to generate other messages, which they also send over the network. The attacker is able to eavesdrop on such messages and add them to his knowledge; therefore, it is possible that $\rho_i \in rvars(\Phi[K_j])$, where $j > i$ represents some later point in time. Note that $K_i \subseteq K_j$, since the attacker's deduction capabilities increase as messages are sent over the network.

Example 8. As noted in Example 7 above, the frames Φ_s and Φ_w are not an accurate description of the attacker's knowledge in the process P_E^*: we must make explicit the fact that handles $h_2, \ldots, h_6$ cannot be used to instantiate the recipe variable ρ. This is done by using the deducibility constraint $D = (\rho, dom(\Phi_{\Sigma^{\mathcal{DY}}}) \cup \{h_1\})$.

Constraint systems. We have seen in Example 5 that a symbolic trace specifies which branch is taken in each `if-then-else` instruction in a process. This branching depends on the equality or inequality of terms built using the messages sent over the network by the attacker. This restricts the capabilities of an attacker since, for each such symbolic trace, the messages he sends must satisfy these equalities and inequalities. Therefore, deducibility constraint sequences are not sufficient to represent the attacker's capabilities for each symbolic trace of a given process. The additional restrictions are represented by equality and inequality constraints, usually referred to as positive and negative constraints, respectively.

Definition 5. *A* constraint system C *is a tuple* (Φ, D, P, N)*, where:*

- $\Phi = \nu\tilde{n}.\sigma$ *is a frame;*
- D *is a DCS for* Φ*;*
- P *and* N *are sets of pairs of terms in* $T_\Sigma(\mathcal{N} \cup dom(D))$*.*

Instead of (t, t')*, we write* $(t \overset{?}{\approx}_{\mathcal{R}} t')$ *if* $(t, t') \in P$ *and* $(t \overset{?}{\not\approx}_{\mathcal{R}} t')$ *if* $(t, t') \in N$*.*

We say that P *is the set of* positive *constraints associated with* C*, while* N *is the set of* negative *constraints.*

We will often write C^Φ and C^D for the frame and DCS associated to a constraint system C.

Example 9. In the process P_E^* described in Example 3, the third step of the second protocol involves agent A checking the equality of two terms. This is expressed through the positive constraint set P containing the single constraint

$$\pi_1\left(\left\{\left|\left\{\left|\langle\left\{\left|\{|M|\}_{\gamma_K}\right|\right\}_K^{-1}, N\rangle\right|\right\}_K\right|\right\}_{\gamma_K}^{-1}\right) \overset{?}{\approx}_{\mathcal{R}} M,$$

where γ_K is as in Example 3. It is simple to check that this does indeed hold.

The resulting deducibility constraints that we use to study the problem in our running example are thus $C_w = (\Phi_w, D, P, \emptyset)$ and $C_s = (\Phi_s, D, P, \emptyset)$.

D-binding. Let $D = (\rho_1, K_1), \ldots, (\rho_n, K_n)$ be a deducibility constraint sequence for a frame Φ. We define the notion of *D-binding of* Φ as a Φ-recipe substitution $\theta\colon dom(D) \to T(\Phi)$ such that, for all $i \in \{1, \ldots, n\}$, we have:

- $\rho_i\theta \in T(\Phi\,|_{K_i})$, and

- $\theta[\rho_i] = \rho_i\theta$.

We say that a D-binding is ground if it is ground as a Φ-recipe substitution.

Given a D-binding θ for a frame $\Phi = \nu\tilde{n}.\sigma$, the frame Φ_θ is defined by

$$\Phi_\theta = \nu\tilde{n}.(\Phi \circ \theta) \circ \Phi.$$

Note that $dom(\Phi) = dom(\Phi_\theta)$.

If $i \in \{1, \ldots, n\}$, then $\theta \mid_{D_i}$ is a D_i-binding of Φ. We write θ_i instead of $\theta \mid_{D_i}$. For each $j \in \{0, 1, \ldots, n\}$, we define the DCS D_{θ_j} inductively as follows:

- $D_{\theta_0} = \epsilon$;
- if $1 \leq j \leq n$, then

$$D_{\theta_j} = D_{\theta_{j-1}}, (\rho_1^j, K_j), \ldots, (\rho_{n_j}^j, K_j),$$

where

$$\left\{\rho_1^j, \ldots, \rho_{n_j}^j\right\} = rvars(\rho_j\theta) \setminus dom(D_{\theta_{j-1}})$$

are the recipes variables which occur in $\rho_j\theta$ and do not occur in $\rho_k\theta$ for any $k < j$.

We write D_θ instead of D_{θ_n}.

Note that, for all $i \in \{1, \ldots, n\}$, D_{θ_i} is a DCS for Φ_{θ_i} and, if θ' is a D_θ-binding for Φ_θ, then $\theta' \circ \theta$ is a D-binding for Φ. This property is used in several of our completeness proofs.

Recall that recipe variables in $dom(D)$ represent recipes for messages that the attacker must send over the network during some protocol execution. This means that such recipe variables cannot be instantiated with different recipes in different contexts, as other recipes do; instead, the attacker must commit to some recipe for each recipe variable in $dom(D)$, and this recipe must be valid at the point in time when the attacker sends the corresponding message over the network. This is captured by the definition of D-binding of Φ: such a D-binding is simply an assignment of recipes to recipe variables in $dom(D)$ such that each recipe in the domain of D is always instantiated to the same recipe, which must only use handles representing messages that have been sent over the network before the attacker must commit to a recipe for that recipe variable.

The fact that these recipes may themselves contain recipe variables will be useful later because it allows us to represent, with a single recipe, a family of possible (ground) recipes that the attacker may use to produce a message at any given point. The deducibility constraint D_θ keeps track of which handles may be used to instantiate each such recipe variable.

(Φ, D)-, (Φ, θ)-recipes. Given a deducibility constraint D for Φ, a (Φ, D)-recipe is a Φ-recipe ζ such that, for each $\rho \in dom(D)$, $\zeta[\rho] = \{\zeta_\rho\}$ for some $\zeta_\rho \in T(\Phi\mid_{D(\rho)})$. This corresponds to requiring that each recipe variable ρ in $dom(D)$ is only instantiated with a single recipe that uses only the handles available to the attacker at the point in time at which the message containing recipe variable ρ must be sent over the network. We denote by $T(\Phi, D)$ the set of (Φ, D)-recipes.

A (Φ, D)-recipe substitution γ is a Φ-recipe substitution such that, for each $\rho \in dom(D)$, there exists a recipe ζ_ρ such that $\gamma[\rho] = \{\zeta_\rho\}$ and $\zeta_\rho \in T(\Phi)$. Note that our definition of (Φ, D)-recipe substitution γ is stronger than merely requiring that all recipes in $ran(\gamma)$ be in $T(\Phi, D)$: if $\zeta, \zeta' \in ran(\gamma)$ and $\rho \in dom(D)$, we do not merely require that $\zeta[\rho] = \zeta_\rho$ and $\zeta'[\rho] = \zeta'_\rho$ for some ζ_ρ and some ζ'_ρ, but also that $\zeta'_\rho = \zeta_\rho$.

If $\zeta \in T(\Phi, D)$, we define the substitution $\theta_{D,\zeta}$ by

$$\theta_{D,\zeta} = \{\rho \mapsto \zeta_\rho \mid \rho \in dom(D), \zeta[\rho] = \zeta_\rho\}\,.$$

Analogously, if γ is a (Φ, D)-recipe substitution, we define the substitution $\theta_{D,\gamma}$ by

$$\theta_{D,\gamma} = \{\rho \mapsto \zeta_\rho \mid \rho \in dom(D), \gamma[\rho] = \zeta_\rho\}\,.$$

It is simple to check that $\theta_{D,\gamma}$ is a D-binding of Φ.

Example 10. In our running example, we have $T(\Phi_s) = T(\Phi_w)$ and $T(\Phi_s, D) = T(\Phi_w, D)$, but not $T(\Phi_s) = T(\Phi_s, D)$: for example, we have seen in Example 7 that $\zeta_1, \zeta_2 \in T(\Phi_s)$. However, neither ζ_1 nor ζ_2 is in $T(\Phi_s, D)$, since $\zeta_1[\rho] = \zeta_2[\rho] = \left\{\{|h_1|\}_{h_6}^{-1}\right\} \not\subseteq T(\Phi\mid_{D(\rho)})$.

If θ is a D-binding of Φ, we say that ζ is a (Φ, θ)-recipe if, for all $\rho \in dom(D)$, $\zeta[\rho] = \{\rho\theta\}$. We denote the set of (Φ, θ)-recipes by $T(\Phi, \theta)$. We say that a Φ-recipe substitution γ is a (Φ, θ)-recipe substitution if, for all $\rho \in dom(D)$, $\gamma[\rho] = \{\rho\theta\}$. This requirement is equivalent to requiring that $ran(\gamma) \subseteq T(\Phi, \theta)$.

A *solution of a constraint system* $C = (\Phi, D, P, N)$ is a D-binding θ of Φ such that, for all $(t \stackrel{?}{\approx}_{\mathcal{R}} t') \in P$, $t\Phi_\theta \approx_R t'\Phi_\theta$, and, for all $(t \stackrel{?}{\not\approx}_{\mathcal{R}} t') \in N$, $t\Phi_\theta \not\approx_R t'\Phi_\theta$. A solution of C is ground if it is ground as a D-binding of Φ. We denote by $\mathsf{sol}(C)$ the set of solutions of C.

(D-)Static equivalence. Static equivalence in our framework is a straightforward generalization of static equivalence for the applied-pi calculus: two frames Φ and Φ' are statically equivalent (under $\mathcal{R}$), written $\Phi \approx^s_{\mathcal{R}} \Phi'$, if $T(\Phi) = T(\Phi')$ and, for all $\zeta, \zeta' \in T(\Phi)$, $\zeta\Phi \approx_{\mathcal{R}} \zeta'\Phi$ iff $\zeta\Phi' \approx_{\mathcal{R}} \zeta'\Phi'$.

We generalize this notion further by introducing the notion of *D-static equivalence*, to account for the additional limitations imposed to the attacker by deducibility constraints. If two frames Φ and Φ' are such that $T(\Phi) = T(\Phi')$ and if D is a deducibility constraint for Φ (or, equivalently, for Φ'), we say that Φ and Φ'

are *D-statically equivalent*, and write $\Phi \approx^s_{\mathcal{R},D} \Phi'$ if, for any D-binding θ of Φ and any recipes $\zeta, \zeta' \in T(\Phi, \theta)$, we have

$$\zeta\Phi \approx_{\mathcal{R}} \zeta'\Phi \qquad \text{iff} \qquad \zeta\Phi' \approx_{\mathcal{R}} \zeta'\Phi'.$$

Equivalently, $\Phi^1 \approx^s_{\mathcal{R},D} \Phi^2$ if and only if, for all $i \in \{1, 2\}$ and all D-bindings θ of Φ^i, $\Phi^i_\theta \approx^s_{\mathcal{R}} \Phi^{3-i}_\theta$. Note that the standard notion of static equivalence corresponds to ϵ-static equivalence, with ϵ representing the empty DCS.

The notion of static equivalence is central in defining the equivalence of constraint systems.

Definition 6 (Equivalence of constraint systems)**..** *Two constraint systems*

$$C^1 = (\Phi^1, D^1, P^1, N^1) \qquad \textit{and} \qquad C^2 = (\Phi^2, D^2, P^2, N^2)$$

are equivalent*, written $C^1 \sim C^2$, if:*

- $T(\Phi^1, D^1) = T(\Phi^2, D^2)$*,*
- *the set of D^1-bindings of Φ^1 and the set of D^2-bindings of Φ^2 are the same, and*
- *whenever $i \in \{1, 2\}$ and θ is a ground solution of C^i, then θ is a solution of C^{3-i} such that $\Phi_\theta \approx^s_{\mathcal{R}} \Phi'_\theta$.*

We denote by $\mathsf{sol}(C)$ the set of solutions of a constraint system C.

Example 11. Continuing our running example, we have $\Phi_w \not\approx^s_{\mathcal{R}} \Phi_s$: The recipes introduced in Example 7 witness this fact, since $\zeta_1\Phi_s \approx_{\mathcal{R}} M \approx_{\mathcal{R}} \zeta_2\Phi_s$ and this equation does not hold in Φ_w because $h_2(\{|h_1|\}^{-1}_{h_6})\Phi_w \not\approx_{\mathcal{R}} K$. However, we have $\Phi_w \approx^s_{\mathcal{R},D} \Phi_s$, and $C_w \sim C_s$.

4.1.3 Application to Trace Equivalence

In [65], Cheval et al. show that deciding trace equivalence of bounded processes can be reduced to deciding the symbolic equivalence of sets of constraint systems. We rely on their results to apply our constraint solving algorithm to the decision of trace equivalence. However, our formal definition of constraint system is designed to fit our technique and ease the presentation, and is slightly different from the one given there. The following theorem states that our notion of constraint system may nevertheless be used to decide trace equivalence. It relies on the translation of bounded processes into constraint systems given in [65] and on associating a constraint system in the sense of the definition given there to a constraint system according to our definition with the same set of solutions.

Theorem 3. *Deciding trace equivalence of simple bounded processes can be reduced to deciding the equivalence of constraint systems (according to Definition 6).*

Proof. Theorem 6 in [65] shows that, given a procedure for deciding the symbolic equivalence of sets of constraint systems, we can derive an algorithm for deciding the trace equivalence of bounded processes (i.e., processes without replication). The proof of Proposition 7 in the same paper shows that, if we restrict ourselves to considering simple processes, then it is sufficient to consider the equivalence of constraint systems (i.e., we may assume that the sets of constraint systems considered are singletons). Therefore, it is sufficient to show that, deciding the equivalence of constraint systems according to the definition in [65] can be reduced to deciding the equivalence of constraint systems according to our definition.

Translation of constraint systems. Suppose that $\mathcal{C} = (\mathcal{E}, \Phi, \mathcal{D})$ is a constraint system as defined in [65], using the (implicit) signature $\mathcal{F}$. We assume that:

- $\mathcal{X}$ is the set of *second-order variables* used in $\mathcal{C}$, with $ar(X)$ denoting the arity of X for each $X \in \mathcal{X}$,
- $\Phi = \{w_1 \rhd t_1, \ldots, w_n \rhd t_n\}$, and
- $\mathcal{D} = D \cup P \cup N$, where D is a set of constraints of the form $X \rhd x$, P is a set of constraints of the form $s =^?_{\mathcal{R}} s'$ and N is a set of constraints of the form $s \neq^?_{\mathcal{R}} s'$.

Moreover, for each $i \in \{1, \ldots, n\}$, let X_i be the (unique) second-order variable such that $(X_i \rhd w_i) \in \mathcal{D}$. We assume that, if $i < j$, then $ar(X_i) \leq ar(X_j)$. This is not strictly required by the definition in [65], but it may be trivially enforced by simply reordering Φ.

We define the corresponding constraint system according to our definition as follows. Let

$$\sigma_{\mathcal{F}} = \left\{h_f \mapsto f(\rho_1, \ldots, \rho_{ar(f)}) \mid f \in \mathcal{F}\right\}$$

and

$$H_{\mathcal{F}} = \{h_f \mid f \in \mathcal{F}\} = dom(\sigma_{\mathcal{F}}).$$

Let $\tau\colon \{w_1, \ldots, w_n\} \rightarrowtail \mathcal{X}_R$ be an injective function and $\hat{\Phi} = \nu\mathcal{E}.\sigma$, where

$$\sigma = \{h_1 \mapsto t_1\tau, \ldots, h_n\tau \mapsto t_n\} \cup \sigma_{\mathcal{F}}$$

for some handles

$$h_1, \ldots, h_n \in \mathcal{H} \setminus H_{\mathcal{F}}.$$

The set D of deducibility constraints in $\mathcal{C}$ is translated as the deducibility constraint sequence $\hat{D}$ given by

$$\hat{D} = (w_1\tau, K_1), \ldots, (w_n\tau, K_n)$$

where

$$K_i = H_{\mathcal{F}} \cup \left\{h_1, \ldots, h_{ar(X_i)}\right\}$$

for each $i \in \{1, \ldots, n\}$. The sets of positive and negative constraints are naturally translated into our model by the sets

$$\hat{P} = \left\{ (s\tau, s'\tau) \mid (s =^?_{\mathcal{R}} s') \in P \right\},$$

and

$$\hat{N} = \left\{ (s\tau, s'\tau) \mid (s \neq^?_{\mathcal{R}} s') \in N \right\}.$$

The constraint system (according to definition 5) $\hat{C}$, given by

$$\hat{C} = (\hat{\Phi}, \hat{D}, \hat{N}, \hat{P}),$$

is the translation of the constraint system $\mathcal{C}$ into our definition.

Translation of substitutions θ. We consider the set of substitutions θ (in the context of [65]) such that

(1) $dom(\mathcal{C}) = \{X_1, \ldots, X_2\} = \mathsf{var}^2(\mathcal{C})$, and

(2) $X_i\theta \in \mathcal{T}((\mathcal{N} \setminus \mathcal{E}) \cup \{w_1, \ldots, w_{ar(X_i)}\})$.

Terms in $\mathcal{T}(\mathcal{N} \setminus \mathcal{E}$ and terms in $T(\Phi)$ are in direct correspondence, as explained in Section 4.1.1: The set of atoms is the same and the application of function symbols is represented in our framework by the handles in $H_{\mathcal{F}}$. These functions θ do not directly coincide with the set of $\hat{D}$-bindings $\hat{\theta}$ of $\hat{C}$ according to our definition for the reason that the range of θ may contain the variables $w_1, \ldots, w_n$; by contrast, in the range of $\hat{\theta}$, the recipe variables $w_1\tau, \ldots, w_n\tau$ are replaced by $w_1\tau\hat{\theta}$, $\ldots$, $w_n\tau\hat{\theta}$. More precisely, such a function θ as in [65] corresponds to the $\hat{D}$-binding $\hat{\theta}$ of $\hat{\Phi}$ defined as follows:

- $\hat{\theta}_0 = \emptyset$;
- for each $i \in \{1, \ldots, n\}$,

$$\hat{\theta}_i = \hat{\theta}_{i-1} \cup \left\{ w_i\tau \mapsto X_i\theta\tau\hat{\theta}_{i-1} \right\};$$

- $\hat{\theta} = \hat{\theta}_n$.

We show by induction that on i that, for all $i \in \{1, \ldots, n\}$:

- $\hat{\theta}_i$ is a $D_{\leq i}$-binding of $\hat{\Phi}$, where $\hat{D}_{\leq i}$ is the restriction of $\hat{D}$ to its first i elements, and
- $w_i\tau\hat{\theta}_i\hat{\Phi} = w_i\lambda$.

The result is trivially true for $i = 0$. If it is true for the $i - 1$, then, for all variables w_j occurring in $X_i\theta$, we have $j < i$ and

$$w_j\tau\hat{\theta}_{i-1} \in T(\hat{\Phi} \mid_{K_j}).$$

Since $X_i\theta$ is built from $\mathcal{N} \setminus \mathcal{E}$ and $w_1, \ldots, w_{i-1}$ by using the function symbols in $\mathcal{F}$, and $K_j \subseteq K_{i-1}$ for all $j \in \{1, \ldots, i-1\}$, it follows that

$$X_i\theta\tau\hat{\theta}_{i-1} \in T(\Phi\,|_{K_i}).$$

Moreover, for all $j \in \{1, \ldots, i-1\}$,

$$w_j\tau\hat{\theta}_i = w_j\tau\hat{\theta}_{i-1} \in T(\Phi\,|_{K_j});$$

thus, $\hat{\theta}$ is a $\hat{D}_{\leq i}$-binding of $\hat{\Phi}$. To prove the second property, we note that, by the induction hypothesis, the result holds for $i-1$; this implies that, for all $j \in \{1, \ldots, i-1\}$,

$$\begin{aligned} w_j\lambda &= (X_j\theta)(\Phi\lambda) \\ &= w_j\tau\hat{\theta}\hat{\Phi}. \end{aligned}$$

Moreover, since

$$\begin{aligned} \mathsf{var}^1(X_i\theta) &\subseteq \left\{w_1, \ldots, w_{ar(X_i)}\right\} \\ &\subseteq \left\{w_1, \ldots, w_{i-1}\right\}, \end{aligned}$$

we obtain

$$\begin{aligned} w_i\lambda &= (X_i\theta)(\Phi\theta) \\ &= (X_i\theta)\tau\hat{\theta}_{i-1}\hat{\Phi} \\ &= w_i\tau\hat{\theta}_i\hat{\Phi}. \end{aligned}$$

Inverse translation of $\hat{D}$-bindings. If $\hat{\theta}$ is a $\hat{D}$-binding of $\hat{\Phi}$ then, for each $i \in \{1, \ldots, n\}$, there exists

$$\zeta_i \in T(\hat{\Phi}\,|_{K_i}, \emptyset)$$

such that

$$\zeta_i\hat{\theta} = w_i\tau\hat{\theta}.$$

We define the term

$$T(\zeta_i) \in \mathcal{T}((\mathcal{N} \setminus \mathcal{E}) \cup \{w_1, \ldots, w_{i-1}\})$$

from ζ_i inductively as follows:

- if $\zeta_i \in \mathcal{N} \setminus \mathcal{E}$, then $T(\zeta_i) = \zeta_i$;
- if $head(\zeta_i) = h_j$ for some $j \in \{1, \ldots, i-1\}$, then we have $\zeta_i = (h_j, \emptyset)$ (since $\zeta_i \in T(\hat{\Phi}\,|_{K_i}, \emptyset)$), and we define $T(\zeta_i) = w_j$;
- if $\zeta_i = h_f(\zeta'_1, \ldots, \zeta'_n)$ for some $f \in \mathcal{F}$, then $T(\zeta_i) = f(T(\zeta'_1), \ldots, T(\zeta'_n))$.

We consider the substitution θ such that, for each $i \in \{1, \ldots, n\}$, $X_i\theta = T(\zeta_i)$. It is straightforward to check that the substitution θ satisfies requirements (1) and (2) and that the translation of θ as defined in the above paragraph is precisely $\hat{\theta}$.

Translation of recipes. We note that, if θ is a substitution satisfying requirements (1) and (2) and λ is the first-order solution of $\mathcal{C}$ associated with θ, the properties of the translation proved above imply that, for all $i \in \{1, \ldots, n\}$,

$$w_i(\Phi\lambda) = h_i\hat{\Phi}_{\hat{\theta}}.$$

Now, we note that, since $\hat{\theta}$ is a ground $\hat{D}$-binding, we have

$$T(\hat{\Phi}_{\hat{\theta}}) = \mathcal{T}((\mathcal{N} \setminus \mathcal{E}) \cup \{h_1, \ldots, h_n\}),$$

using the identification between the application of function symbols and the application of the corresponding handles described for our method in Section 4.1.1. Thus, there exists a natural bijective translation between $\hat{\Phi}_{\hat{\theta}}$-recipes in our notion of constraint system and $(\Phi\lambda)$-recipes in the definition of [65]: More precisely, if ι is the function

$$\iota = \{w_i \mapsto h_i \mid i \in \{1, \ldots, n\}\},$$

then, for all

$$t \in \mathcal{T}((\mathcal{N} \setminus \mathcal{E}) \cup \{w_1, \ldots, w_n\}),$$

we have

$$t(\Phi\lambda) = t\iota(\hat{\Phi}_{\hat{\theta}});$$

and similarly, for all $\zeta \in T(\hat{\Phi}_{\hat{\theta}})$, we have

$$\zeta(\hat{\Phi}_{\hat{\theta}}) = \zeta\iota^{-1}(\Phi\lambda).$$

Proof of main result. Suppose that $\mathcal{C}_1$ and $\mathcal{C}_2$ are constraint systems according to [65], and consider their translations $\hat{\mathcal{C}}_1$ and $\hat{\mathcal{C}}_2$ into constraint systems according to our definition as defined in the paragraphs above. The definition of [65] requires that $\mathcal{C}_1$ and $\mathcal{C}_2$ have the same structure; this can be equivalently stated as saying that the functions θ satisfying requirements (1) and (2) are the same for $\mathcal{C}_1$ and $\mathcal{C}_2$. This corresponds to our requirement that $T(\hat{\Phi}_1, \hat{D}_1) = T(\hat{\Phi}_2, \hat{D}_2)$ and the set of $\hat{D}_1$-bindings of $\hat{\Phi}_1$ coincides with the set of $\hat{D}_2$-bindings of $\hat{\Phi}_2$. Moreover, we have seen above that substitutions θ satisfying requirements (1) and (2) can be associated bijectively with $\hat{D}_1$-bindings $\hat{\theta}$ of $\hat{\Phi}_1$ (equivalently, with $\hat{D}_2$-bindings $\hat{\theta}$ of $\hat{\Phi}_2$).

Suppose that $\mathcal{C}_1$ and $\mathcal{C}_2$ are not equivalent. Then, there exists a substitution θ satisfying requirements (1) and (2) such that either (a) θ is a solution of one of the constraint systems and not of the other, or (b) it is a solution of both constraint systems and

$$(\Phi\lambda_1) \not\approx^s_{\mathcal{R}} (\Phi\lambda_2),$$

where λ_1 and λ_2 are the first-order solutions of $\mathcal{C}_1$ and $\mathcal{C}_2$ (respectively) associated with θ.

In case (a), assume (without loss of generality) that θ is a solution of $\mathcal{C}_1$ and not of $\mathcal{C}_2$. Then, we have

- for all $(s =^?_{\mathcal{R}} s') \in P_1$, $s\lambda_1 \approx_{\mathcal{R}} s'\lambda_1$,
- for all $(s \neq^?_{\mathcal{R}} s') \in N_1$, $s\lambda_1 \not\approx_{\mathcal{R}} s'\lambda_1$,

and there is either

- $(s =^?_{\mathcal{R}} s') \in P_2$ such that $s\lambda_2 \not\approx_{\mathcal{R}} s'\lambda_2$, or
- $(s \neq^?_{\mathcal{R}} s') \in N_2$ such that $s\lambda_2 \approx_{\mathcal{R}} s'\lambda_2$.

Now, the property that

$$\begin{aligned} w_i\tau\hat{\Phi}_{1,\hat{\theta}} &= w_i\tau\hat{\theta}_i\hat{\Phi}_1 \\ &= w_i\lambda, \end{aligned}$$

proved above, implies that an equality is satisfied after applying the substitution λ_1 if and only if it is satisfied after applying the substitution $\hat{\Phi}_{1,\hat{\theta}} \circ \tau$; thus, $\hat{\theta}$ is a solution of $\hat{C}_1$ and not of $\hat{C}_2$, showing that the constraint systems $\hat{C}_1$ and $\hat{C}_2$ are not equivalent.

In case (b), there exist recipes

$$t, t' \in \mathcal{T}((\mathcal{N} \setminus \mathcal{E}) \cup \{w_1, \ldots, w_n\})$$

which are equal (modulo $\approx_{\mathcal{R}}$) in one of the constraint systems and not under the other. For concreteness, suppose without loss of generality that

$$t(\Phi_1\lambda_1) \approx_{\mathcal{R}} t'(\Phi_1\lambda_1) \qquad \text{and} \qquad t(\Phi_2\lambda_2) \not\approx_{\mathcal{R}} t'(\Phi_2\lambda_2).$$

In this case, the result proved in the above paragraph shows that

$$t\iota(\hat{\Phi}_{1,\hat{\theta}}) \approx_{\mathcal{R}} t'\iota(\hat{\Phi}_{1,\hat{\theta}}) \qquad \text{and} \qquad t\iota(\hat{\Phi}_{2,\hat{\theta}}) \approx_{\mathcal{R}} t'\iota(\hat{\Phi}_{2,\hat{\theta}}),$$

and again $\hat{C}_1 \not\sim \hat{C}_2$.

We conclude that, if $\mathcal{C}_1 \not\sim \mathcal{C}_2$, then we can use our translations of constraint systems, substitutions and recipes to transform a witness of that fact into a witness that $\hat{C}_1 \not\sim \hat{C}_2$. The converse can be proven completely analogously. This shows that the problem of deciding equivalence of constraint systems as defined in [65] is equivalent (in terms of decidability) to the problem of deciding the equivalence of constraint systems according to our definition, concluding the proof.

□

4.2 (Φ, D)-Unification

In this section we describe (Φ, D)-unification, which is of central importance in our decision procedure. Intuitively, (Φ, D)-unification generalizes syntactic unification because the terms to be unified may contain *recipe variables* as well as term variables and constants. Recipe variables differ from term variables because

they are mapped to recipes rather than to terms. Φ-solutions to unification problems are pairs (α, γ), where α is a term substitution that maps term variables to terms as usual. Note, however, that terms in our framework may contain recipe variables. In contrast, γ is a Φ-recipe substitution that maps recipe variables to Φ-recipes. Such a pair (α, γ) is a solution of the Φ-unification problem if the substitution $\alpha \cup (\Phi \circ \gamma)$ is a solution of the (syntactic) unification problem in the standard sense. (Φ, D)-unification imposes the stronger constraint that γ must be a (Φ, D)-recipe substitution, i.e., each recipe variable in $dom(D)$ must always be instantiated with the same Φ-recipe using only the handles associated with it by D. If D is the empty sequence, then the set of (Φ, D)-solutions of a unification problem coincides with its set of Φ-solutions.

Recipe variables are also reminiscent of second-order term variables in second order unification [17]. Φ-unification is, however, a different problem with a more restricted set of solutions, since recipe variables cannot be mapped to any term and instead must be mapped to terms in the range of Φ.

Throughout this section we will use a simple running example to illustrate the notions introduced, encompassing Examples 12—14.

4.2.1 Unification

If

$$\mathcal{U} = \left\{ (t_i^1 \stackrel{?}{=} t_i^2) \mid i \in \{1, \ldots, n\} \right\}$$

is a unification problem, we define the set $tvars(\mathcal{U})$ of term variables occurring in $\mathcal{U}$ by $tvars(\mathcal{U}) = \bigcup_{i=1}^{n} \left(\bigcup_{j=1}^{2} tvars(t_i^j) \right)$, and define the set $rvars(\mathcal{U})$ of recipe variables occurring in $\mathcal{U}$ analogously. If ξ is any substitution, we write $\xi \mid_{\mathcal{U}}$ to denote the restriction of ξ to $tvars(\mathcal{U}) \cup rvars(\mathcal{U})$.

We define the partial order relation $\preceq^{\mathcal{U}}$ on classical substitutions as follows: if ϱ and ϱ' are classical substitutions, then $\varrho \preceq^{\mathcal{U}} \varrho'$ if there exists a classical substitution ϱ_* such that $\varrho' \mid_{\mathcal{U}} = \varrho_* \circ \varrho \mid_{\mathcal{U}}$.

A *classical solution* of $\mathcal{U}$ is a classical substitution ϱ such that $t_i^1 \varrho = t_i^2 \varrho$ for all $i \in \{1, \ldots, n\}$. Classical solutions of a Φ-unification problem correspond to the usual notion of *unifier* when recipe variables are interpreted as term variables. It is well-known that if a unification problem $\mathcal{U}$ admits a unifier, then there is a *most general unifier* ϱ such that, if ϱ' is any unifier, then $\varrho \preceq^{\mathcal{U}} \varrho'$.

In the following, we let Unif be a syntactic unification algorithm which, given a unification problem $\mathcal{U}$, returns a most general unifier $\mathsf{Unif}(\mathcal{U})$ of $\mathcal{U}$ if one exists, and $\bot$ otherwise. Furthermore, we assume that the solution ϱ output by $\mathsf{Unif}(\mathcal{U})$ is such that $dom(\varrho) = rvars(\mathcal{U}) \cup tvars(\mathcal{U})$, $rvars(ran(\varrho)) = \emptyset$, and all term variables occurring in the range of ϱ are fresh. The algorithm Unif runs in linear time [17].

We will use a separate, simple running example to illustrate some aspects of Φ- and (Φ, D)-unification. This spans Examples 12—14.

Example 12. Consider a signature $\Sigma_f = \Sigma_{f,2} = \{f\}$ that contains a single binary function symbol f, and the unification problem $\mathcal{U} = \left\{\rho_* \stackrel{?}{=} f(s, f(s, \rho'_*))\right\}$, where $\rho_*, \rho'_* \in \mathcal{X}_R$. The standard syntactic unifier of $\mathcal{U}$ is simply

$$\left\{\rho_* \mapsto f(s, f(s, \rho''_*)), \rho'_* \mapsto \rho''_*\right\}$$

for some $\rho''_* \in \mathcal{X}_R$.

4.2.2 Φ-Unification

$(\Phi, \mathcal{X}_R, \mathcal{X}_T)$-substitutions. If $\Phi = \nu\tilde{n}.\sigma$ is a frame, a $(\Phi, \mathcal{X}_R, \mathcal{X}_T)$-substitution is a pair (α, γ), where

$$\alpha \colon \mathcal{X}_T \nrightarrow T_\Sigma(\mathcal{N} \cup \mathcal{X}_R \cup \mathcal{X}_T)$$

is a term substitution, and

$$\gamma \colon \mathcal{X}_R \nrightarrow T(\Phi)$$

is a Φ-recipe substitution. If (α, γ) is a $(\Phi, \mathcal{X}_R, \mathcal{X}_T)$-substitution, we define the (classical) substitution

$$(\alpha, \gamma)_\Phi \colon dom(\alpha) \cup dom(\gamma) \to T_\Sigma(\mathcal{N} \cup \mathcal{X}_R \cup \mathcal{X}_T)$$

by $(\alpha, \gamma)_\Phi = \alpha \cup (\Phi \circ \gamma)$.

We extend $(\alpha, \gamma)_\Phi$ homomorphically to $T_\Sigma(\mathcal{N} \cup \mathcal{X}_R \cup \mathcal{X}_T)$. If $\mathcal{U}$ contains no term variables, we say that a Φ-recipe substitution γ is a Φ-solution if $(\emptyset, \gamma)$ is a Φ-solution.

(Complete sets of) Φ-solutions. A Φ-*solution of* $\mathcal{U}$ is a $(\Phi, \mathcal{X}_R, \mathcal{X}_T)$-substitution (α, γ) such that $(\alpha, \gamma)_\Phi$ is a classical solution of $\mathcal{U}$.

We define the partial order relation $\preceq^{\mathcal{U},\Phi}$ on $(\Phi, \mathcal{X}_R, \mathcal{X}_T)$-substitutions as follows: $(\alpha, \gamma) \preceq^{\mathcal{U},\Phi} (\alpha', \gamma')$ if and only if there exist a term substitution

$$\alpha_* \colon \mathcal{X}_T \nrightarrow T_\Sigma(\mathcal{N} \cup \mathcal{X}_R \cup \mathcal{X}_T)$$

and a replacement function

$$\gamma_* \colon \mathcal{X}_R \nrightarrow T_\Sigma(\mathcal{N} \cup \mathcal{X}_R)$$

such that:

- $\alpha' = \alpha_* \circ \alpha$;
- for all $\rho \in rvars(\mathcal{U})$ and all $p \in pos(\rho\gamma)$, we have $p \in pos(\rho\gamma')$ and either:
  - $head(\rho\gamma\,|_p) = head(\rho\gamma'\,|_p)$;
  - or $\rho\gamma\,|_p \in dom(\gamma_*)$ and $\rho\gamma'\,|_p\ \Phi = \rho\gamma_*$.

The first of these conditions is as in the definition of most general syntactic unifier. The second condition intuitively means that γ' may be obtained from γ by taking each position p of a recipe in the range of γ in which a recipe variable ρ' occurs and replacing it by some Φ-recipe for $\rho'\gamma_*$.

A set $\Delta^{\Phi}_{\mathcal{U}}$ is a *complete set of* Φ*-solutions of* $\mathcal{U}$ if all elements of $\Delta^{\Phi}_{\mathcal{U}}$ are Φ-solutions of $\mathcal{U}$ and, whenever (α', γ') is a Φ-solution of $\mathcal{U}$, there is $(\alpha, \gamma) \in \Delta^{\Phi}_{\mathcal{U}}$ such that $(\alpha, \gamma) \preceq^{\mathcal{U},\Phi} (\alpha', \gamma')$.

Finite complete sets of Φ-solutions do not always exist, as illustrated by the following example. We conjecture that it is decidable whether a unification problem has a finite complete set of Φ-solutions.

Example 13. We continue our running example for unification introduced in Example 12. Consider the unification problem $\mathcal{U}$ introduced in that example, and the frame

$$\Phi = \left\{h_1 \mapsto f(s, \rho), h_2 \mapsto f(s, f(s, \rho'))\right\}$$

over Σ_f. The functions

$$\gamma_{11} = \left\{\rho_* \mapsto h_1(h_1(\rho'_*)), \rho'_* \mapsto \rho'_*\right\} \qquad \text{and} \qquad \gamma_2 = \left\{\rho_* \mapsto h_2(\rho'_*)\right\}$$

are Φ-unifiers of $\mathcal{U}$.

There is no finite complete set of Φ-unifiers of $\mathcal{U}$: The smallest complete set of Φ-solutions of $\mathcal{U}$ contains γ_{11}, γ_2, and all elements of the set

$$\Delta^{N}_{\mathcal{U}} = \left\{\rho_* \mapsto h_1(h_2^n(h_1(\rho'_*))), \rho'_* \mapsto h_2^n(\rho'_*) \mid n \in \mathbb{N}\right\},$$

where $h_i^n(x)$ represents n consecutive applications of the handle h_i to x: that is, $h_i^0(x) = x$, and, for each $k \in \mathbb{N} \setminus \{0\}$, $h_i^k = h_i(h_i^{k-1}(x))$.

4.2.3 (Φ, D)-Unification

(Complete sets of) (Φ, D)**-solutions.** If $D = (\rho_1, K_1), \ldots, (\rho_n, K_n)$ is a DCS for Φ, a (Φ, D)*-solution of* $\mathcal{U}$ is a Φ-solution (α, γ) of $\mathcal{U}$ such that γ is a (Φ, D)-recipe substitution. As before, a set $\Delta^{\Phi,D}_{\mathcal{U}}$ is a *complete set of* (Φ, D)*-solutions of* $\mathcal{U}$ if all elements of $\Delta^{\Phi,D}_{\mathcal{U}}$ are (Φ, D)-solutions of $\mathcal{U}$ and, whenever (α', γ') is a (Φ, D)-solution of $\mathcal{U}$, there is $(\alpha, \gamma) \in \Delta^{\Phi,D}_{\mathcal{U}}$ such that $(\alpha, \gamma) \preceq^{\mathcal{U},\Phi} (\alpha', \gamma')$.

If D is the empty sequence, then (α, γ) is a Φ-solution of $\mathcal{U}$ if and only if it is a (Φ, D)-solution of $\mathcal{U}$. Therefore, our algorithm for finding (Φ, D)-solutions of a unification problem can also be used to find Φ-solutions.

Example 14. Continuing our unification example, consider now the DCS D for Φ given by $D = (\rho', \{h_1\})$. Then, γ_{11} and γ_2 are (Φ, D)-solutions of $\mathcal{U}$; however, no element of $\Delta^{N}_{\mathcal{U}} \setminus \{\gamma_{11}\}$ is, since they instantiate ρ' with more than one different recipe. $\{\gamma_{11}, \gamma_2\}$ is a complete set of (Φ, D)-solutions of $\mathcal{U}$.

If $D' = (\rho', \{h_1\}), (\rho, \{h_1, h_2\})$ then, for the same reason, γ_{11} is not a (Φ, D')-solution. $\{\gamma_2\}$ is a complete set of (Φ, D')-solutions of $\mathcal{U}$.

Example 15. Let us now return to our running example in Examples 3—11, and consider the unification problem

$$\mathcal{U} = \left\{ \{|\rho_1|\}^{-1}_{\rho_2} \stackrel{?}{=} \left\{ \left| \{|x|\}_y \right| \right\}^{-1}_{y} \right\}.$$

A Φ-solution of $\mathcal{U}$ is given by (α, γ), where

$$\alpha = \left\{ x \mapsto \{K\}^{r'}_{KA_{\mathsf{pub}}}, y \mapsto KAB \right\} \qquad \text{and} \qquad \gamma = \{\rho_1 \mapsto h_1, \rho_2 \mapsto h_6\};$$

indeed, this is also a (Φ, D)-solution. This shows that the attacker represented by Φ_s can deduce $\left\{ \left| \{|x|\}_y \right| \right\}^{-1}_{y} \downarrow = \{K\}^{r'}_{KA_{\mathsf{pub}}}$. $\mathcal{U}$ does not have Φ_w-solutions, and therefore also no (Φ_w, D)-solution.

Consider now the frame $\Phi'_s = \Phi_s \cup \left\{ h_7 \mapsto \{K\}^{r'}_{KA_{\mathsf{pub}}} \right\}$, which extends Φ_s with the term that we have just found to be deducible, and the unification problem

$$\mathcal{U} = \left\{ \{\rho\}^{-1}_{KA_{\mathsf{priv}}} \stackrel{?}{=} \left\{ \{x\}^{z}_{y_{\mathsf{pub}}} \right\}^{-1}_{y_{\mathsf{priv}}} \right\}.$$

Note that

$$\{\rho\}^{-1}_{KA_{\mathsf{priv}}} \in sub\left(\left\{ \{\rho\}^{-1}_{KA_{\mathsf{priv}}} \right\}^{r}_{KC_{\mathsf{pub}}} \right) = sub(h_2\Phi).$$

We find that the set $\{(\alpha, \gamma), (\alpha_7, \gamma_7)\}$, where

$$\alpha = \alpha_7 = \left\{ x \mapsto K, y \mapsto KA, z \mapsto r' \right\}$$

and

$$\gamma = \left\{ \rho \mapsto \{|h_1|\}^{-1}_{h_6} \right\}, \qquad \gamma_7 = \{\rho \mapsto h_7\}$$

is a finite complete set of Φ'_s-solutions of $\mathcal{U}$.

There are no (Φ'_s, D)-solutions, as D forbids instantiating ρ using h_6. These unifiers will be relevant in the next section to illustrate our saturation algorithm; intuitively, they represent ways in which an attacker can use his knowledge to instantiate left-hand sides of rewrite rules and thereby deduce new terms.

4.2.4 (Φ, D)-Unification Algorithm

Our (Φ, D)-unification algorithm solves a slightly more general problem: given a frame Φ, a DCS D, a unification problem $\mathcal{U}$, and a D-binding θ of Φ, it outputs a finite complete set $\mathsf{genUnif}(\Phi, D, \mathcal{U}, \theta)$ of (Φ_θ, D_θ)-solutions of $\mathcal{U}$.

Note that, taking $\theta = id(dom(D))$, we have $\Phi_\theta = \Phi$ and $D_\theta = D$. In this case, if $\mathsf{genUnif}(\Phi, D, \mathcal{U}, \theta)$ is a complete set of (Φ, D, θ)-solutions of $\mathcal{U}$, then $\mathsf{genUnif}(\Phi, D, \mathcal{U}, \theta)$ is also a complete set of (Φ, D)-solutions of $\mathcal{U}$.

Linear-left form. The set of terms that occur on the left-hand side of $\mathcal{U}$ is $\mathcal{U}_L = \{t_1^1, \ldots, t_n^1\}$; analogously, $\mathcal{U}_R$ is the set of terms occurring on $\mathcal{U}$'s right-hand side. If ϱ is a classical substitution, we define the application of ϱ to $\mathcal{U}$ to be the unification problem

$$\mathcal{U}\varrho = \left\{ t_i^1 \varrho \stackrel{?}{=} t_i^2 \varrho \mid i \in \{1, \ldots, n\} \right\}.$$

We say that a unification problem $\mathcal{U}$ is in *linear-left form* if:

- $\mathcal{U}_L \subset \mathcal{X}_T \cup \mathcal{X}_R$,
- $tvars(\mathcal{U}_R) \cap tvars(\mathcal{U}_L) = \emptyset$,
- $rvars(\mathcal{U}_R) = \emptyset$,
- and $t_i^1 \neq t_j^1$ whenever i, j are distinct integers in $\{1, \ldots, n\}$.

The next Lemma shows that it is sufficient to solve unification problems in linear-left form.

Lemma 5. *Let $\mathcal{U}$ be a unification problem and $\varrho = \mathsf{Unif}(\mathcal{U})$. Let $\mathcal{U}_\varrho$ be the unification problem given by*

$$\mathcal{U}_\varrho = \left\{ (x \stackrel{?}{=} x\varrho) \mid x \in dom(\varrho) \right\}.$$

Then, $\mathcal{U}_\varrho$ is in linear-left form. If D is a DCS for Φ, θ is a D-binding of Φ, and Δ_ϱ is a complete set of (Φ, D, θ)-solutions of $\mathcal{U}_\varrho$, then Δ_ϱ is also a complete set of (Φ, D, θ)-solutions of $\mathcal{U}$.

Solved forms. We say that $\mathcal{U}$ is in *solved form* if it is in linear-left form and, for all $(t^1, t^2) \in \mathcal{U}$, if $t^1 \in \mathcal{X}_R$, then $t^2 \in \mathcal{X}_T$.

Suppose that $\mathcal{U}$ is in solved form, and let

$$\mathcal{U}_{\mathcal{X}_R} = \left\{ (t_i^1 \stackrel{?}{=} t_i^2) \in \mathcal{U} \mid t_i^1 \in \mathcal{X}_R \right\}$$

and

$$\mathcal{U}_{\mathcal{X}_T} = \left\{ (t_i^1 \stackrel{?}{=} t_i^2) \in \mathcal{U} \mid t_i^2 \in \mathcal{X}_T \right\}.$$

Let $\iota \colon tvars(\mathcal{U}_{\mathcal{X}_R}) \to \mathcal{X}_R$ be some injective substitution. Define

$$\gamma \colon rvars(\mathcal{U}_{\mathcal{X}_R}) \to T(\Phi_\theta, D_\theta)$$

by

$$\gamma = \left\{ \rho \mapsto x\iota \mid (\rho \stackrel{?}{=} x) \in \mathcal{U}_{\mathcal{X}_R} \right\}.$$

Finally, let $\alpha = \alpha_L \cup \alpha_R \cup \alpha_P$, where

$$\alpha_L = \left\{ x \mapsto t \mid (x \stackrel{?}{=} t) \in \mathcal{U}_{\mathcal{X}_T} \right\},$$

$$\alpha_P = \left\{ x \mapsto \rho\iota \mid (\rho \stackrel{?}{=} x) \in \mathcal{U}_{\mathcal{X}_R} \right\},$$

and

$$\alpha_R = \{ x \mapsto x\alpha_P \mid x \in tvars(\mathcal{U}_{\mathcal{X}_{T,R}}) \}.$$

Note that α_R could be equivalently defined by

$$\alpha_R = \alpha_P \cup id(tvars(\mathcal{U}_{\mathcal{X}_{T,R}}) \setminus tvars(\mathcal{U}_{\mathcal{X}_R})).$$

We define the set $\mathsf{sol}(\Phi, D, \mathcal{U}, \theta)$ by

$$\mathsf{sol}(\Phi, D, \mathcal{U}, \theta) = \{(\alpha, \gamma)\}.$$

This is a (finite) complete set of (Φ, D, θ)-solutions of $\mathcal{U}$, as stated by the next lemma.

Lemma 6. *Let $\mathcal{U}$ be a unification problem in solved form. Then, $\mathsf{sol}(\Phi, D, \mathcal{U}, \theta)$ is a complete set of (Φ, D, θ)-solutions of $\mathcal{U}$.*

Algorithm 5 is our (Φ, D, θ)-unification algorithm. Theorem 5 establishes its correction.

Theorem 4. *If Algorithm 5 terminates on input $(\Phi, D, \mathcal{U}, \theta)$, then it outputs a finite complete set of (Φ_θ, D_θ)-solutions of $\mathcal{U}$.*

4.3 D-Saturations

In this section we introduce *D-saturation*, another key technique of our approach. Saturation techniques have been widely used to decide static equivalence [37, 69, 76]. In these approaches (and in Chapter 3), a saturation of a frame Φ is an extension Φ_s of Φ such that every term *deducible* in Φ (using equational reasoning) is *constructible* in Φ_s (syntactically), and such that every Φ_s-recipe ζ_s has a *translation* into a Φ-recipe ζ such that $\zeta_s\Phi_s \approx_{\mathcal{R}} \zeta\Phi$. D-saturation generalizes this notion.

Intuitively, if Φ is a frame and D is a DCS for Φ, a D-saturation is a set of D-bindings for Φ such that any ground D-binding θ' for Φ is an instance of a D-binding θ in the saturation that describes all the possible rewriting steps in $\Phi_{\theta'}$. In this sense, a D-saturation may also be thought of as a complete set of D-bindings of Φ, together with the corresponding saturations (in the classical sense) of the resulting frames. Thus, each D-binding θ in the saturation is associated with a frame $\Theta_F(\theta)$ (the F stands for *frame*) which saturates Φ_θ in the usual sense (described above), and a Φ-translation function $\Theta_T(\theta)$ (the T stands for *translation*) that associates each $(\Theta_F(\theta), D_\theta)$-recipe ζ to a (Φ, D)-recipe ζ' such that $\zeta\Theta_F(\theta) \approx_{\mathcal{R}} \zeta'\Phi$.

In the rest of this section we give precise definitions of these notions and present our D-saturation algorithm.

Algorithm 5 (Φ, D)-unification algorithm.

`Input:` a frame Φ, a DCS D for Φ, a D-binding θ of Φ,
and a unification problem $\mathcal{U} = \left\{t_i^1 \stackrel{?}{=} t_i^2 \mid i \in \{1, \ldots, n\}\right\}$

`Output:` a complete set $\mathsf{genUnif}(\Phi, D, \theta, \mathcal{U})$ of (Φ_θ, D_θ)-solutions of $\mathcal{U}$

`if` $\mathcal{U}$ is in solved form
  `output` $\mathsf{sol}(\mathcal{U})$
`else if` $\mathcal{U}$ is not in linear-left form
  $\varrho \leftarrow \mathsf{Unif}(\mathcal{U})$
  $\mathcal{U}_\varrho \leftarrow \left\{(x \stackrel{?}{=} x\varrho) \mid x \in dom(\varrho)\right\}$
  `output` $\mathsf{genUnif}(\Phi, D, \mathcal{U}_\varrho, \theta)$
`else choose` $(\rho \stackrel{?}{=} t) \in \mathcal{U}$ `s.t.` $\rho \in \mathcal{X}_R$ `and` $t \notin \mathcal{X}_T$
  `for` $h \in dom(\Phi_\theta)$ `s.t.` $\rho \notin dom(D_\theta)$ `or` $h \in D_\theta(\rho)$
    `choose` ι: $rvars(h\Phi_\theta) \setminus dom(D_\theta) \rightarrowtail \mathcal{X}_R$ fresh
    $\mathcal{U}^h \leftarrow \mathcal{U}(\{\rho \mapsto h\Phi_\theta\iota\})$
    $\theta' \leftarrow \theta$
    `if` $\rho \in dom(D_\theta)$
      $\theta' \leftarrow \{\rho \mapsto (h, \emptyset)\} \circ \theta$
    $\Delta^h \leftarrow \mathsf{genUnif}(\Phi, D, \mathcal{U}^h, \theta')$
    $\Delta^h_{\mathcal{U}} \leftarrow \left\{(\alpha, \gamma \circ \{\rho \mapsto (h, \iota)\}) \mid (\alpha, \gamma) \in \Delta^h\right\}$
  $\Delta^a_{\mathcal{U}} \leftarrow \emptyset$
  `if` $t \in \mathcal{N} \setminus \tilde{n}$
    $\mathcal{U}^a \leftarrow \mathcal{U} \setminus \left\{\rho \stackrel{?}{=} t\right\}$
    $\theta' \leftarrow \theta$
    `if` $\rho \in dom(D)$
      $\theta' \leftarrow \{\rho \mapsto t\} \circ \theta$
    $\Delta^a \leftarrow \mathsf{genUnif}(\Phi, D, \mathcal{U}^a, \theta')$
    $\Delta^a_{\mathcal{U}} \leftarrow \{(\alpha, \gamma \circ \{\rho \mapsto t\}) \mid (\alpha, \gamma) \in \Delta^a\}$
  `output` $\Delta^a_{\mathcal{U}} \cup \left(\bigcup_{h \in dom(\Phi)} \Delta^h_{\mathcal{U}}\right)$

Φ-translation. A Φ-translation is a substitution $[\![\cdot]\!]_\Phi \colon \mathcal{H} \nrightarrow T(\Phi)$. If $\Phi' = \nu\tilde{n}.\sigma'$ is a frame such that $dom(\sigma') \subseteq dom([\![\cdot]\!]_\Phi)$, we extend the substitution $[\![\cdot]\!]_\Phi$ to $T(\Phi')$ as follows:

- if $t \in \mathcal{N} \cup \mathcal{X}_R$, then $[\![t]\!]_\Phi = t$;
- if $h \in dom(\sigma')$, then $[\![(h,\gamma)]\!]_\Phi = [\![h]\!]_\Phi([\![\cdot]\!]_\Phi \circ \gamma)$.

In this case, we also say that $[\![\cdot]\!]_\Phi$ is a Φ-*translation for* Φ'.

If γ is a Φ'-recipe substitution, we write $[\![\gamma]\!]_\Phi$ for the Φ-recipe substitution

$$[\![\cdot]\!]_\Phi \circ \gamma \colon dom(\gamma) \to T(\Phi),$$

so that $\rho[\![\gamma]\!]_\Phi = [\![\rho\gamma]\!]_\Phi$ for all $\rho \in \mathcal{X}_R$.

We will be interested in Φ-translations $[\![\cdot]\!]_\Phi$ for frames Φ' with the additional property that $[\![h]\!]_\Phi\Phi \approx_\mathcal{R} h\Phi'$ for all $h \in dom(\Phi')$. In this case, it is straightforward to prove that this property is preserved for all $T(\Phi')$: that is, if $\zeta \in T(\Phi')$, then $[\![\zeta]\!]_\Phi \in T(\Phi)$ and $[\![\zeta]\!]_\Phi\Phi \approx_\mathcal{R} \zeta\Phi'$.

DCS translation. Suppose that Φ and Φ' are frames,

$$D = (\rho_1, K_1), \ldots, (\rho_n, K_n)$$

is a DCS for Φ, and $[\![\cdot]\!]_\Phi \colon dom(\Phi') \to T(\Phi)$ is a Φ-translation function for Φ'. We define the $[\![\cdot]\!]_\Phi$-*translation* of D to be

$$[\![D]\!]_\Phi = (\rho_1, \overline{K_1}), \ldots, (\rho_n, \overline{K_n}),$$

where $\overline{K_1}, \ldots, \overline{K_n}$ are defined, for each $i \in \{1, \ldots, n\}$, by

$$\overline{K_i} = \left\{ h \in dom(\Phi') \mid [\![h]\!]_\Phi \in T(\Phi \mid_{K_1}) \right\}.$$

It is straightforward to check that, if θ is a $[\![D]\!]_\Phi$-binding for Φ', then $[\![\theta]\!]_\Phi$ is a D-binding for Φ.

D-saturation. A D-saturation of Φ is a function $\Theta = (\Theta_F, \Theta_T)$ (i.e., there exist functions Θ_F, Θ_T such that $dom(\Theta) = dom(\Theta_F) = dom(\Theta_T)$ and $\Theta(x) = (\Theta_F(x), \Theta_T(x))$ for all $x \in dom(\Theta)$) such that

- for all $\theta \in dom(\Theta)$:
  - θ is a D-binding of Φ,
  - $\Theta_F(\theta)$ is a frame such that $(\Phi_\theta)\!\downarrow \subseteq \Theta_F(\theta)$,
  - and $\Theta_T(\theta) \colon dom(\Theta_F(\theta)) \to T(\Phi, D_\theta)$ is a Φ-translation for $\Theta_T(\theta)$;
- whenever θ' is a ground D-binding of Φ, there exists $\theta \in dom(\Theta)$ and a $(\Theta_F(\theta), \Theta_T(\theta)(D_\theta))$-recipe substitution γ such that:

- $\theta\gamma\Theta_F(\theta) \approx_{\mathcal{R}} \Phi\downarrow$;
- for all $\zeta' \in T(\Phi_\theta)$, there exists $\zeta \in T(\Theta_F(\theta), id(dom(D_\theta)))$ such that

$$\zeta\gamma\Theta_F(\theta) = (\zeta'\Phi_{\theta'})\downarrow .$$

Note that, in the second requirement, we use the fact that

$$dom(\Theta_T(\theta)(D_\theta)) = dom(D_\theta),$$

implied by the requirement that $\Phi_\theta\downarrow \subseteq \Theta_F(\theta)$.

Example 16. We continue our running example. From the unification problem and its solution described in Example 15, we see that to obtain an ϵ-saturation we must add to the frame Φ_s the deducible terms $\{K\}^{r'}_{K\mathsf{A}_{\mathsf{pub}}}$ and $\{K\}^{r}_{K\mathsf{C}_{\mathsf{pub}}}$, picking new handles h_7 and h_8. This frame is still associated with the empty D-binding $\theta = \emptyset$, since $dom(\epsilon) = \emptyset$. Thus, we get

$$\Theta_F(\emptyset) = \Phi_s \cup \left\{ h_7 \mapsto \{K\}^{r'}_{K\mathsf{A}_{\mathsf{pub}}}, h_8 \mapsto \{K\}^{r}_{K\mathsf{C}_{\mathsf{pub}}} \right\},$$

with

$$\Theta_T(\emptyset)(h_7) = \{|h_1|\}^{-1}_{h_6} \qquad \text{and} \qquad \Theta_T(\emptyset)(h_8) = h_2(\{|h_1|\}^{-1}_{h_6}).$$

Continuing the saturation of the frame, the unification problem

$$\left\{ \{\rho_1\}^{-1}_{\rho_2} \stackrel{?}{=} \left\{ \{x\}^{z}_{y_{\mathsf{pub}}} \right\}^{-1}_{y_{\mathsf{priv}}} \right\}$$

admits a $(\Theta_F(\emptyset), \epsilon)$-solution (α, γ), with

- $x\alpha = K$;
- $y\alpha = K\mathsf{C}$;
- $z\alpha = r$;
- $\rho_1\gamma = h_8$;
- $\rho_2\gamma = K\mathsf{C}_{\mathsf{priv}}$

(recall that $\{\rho_1\}^{-1}_{\rho_2}, \rho_{1\mathsf{priv}} \in \Phi_{\Sigma^{\mathcal{D}\mathcal{Y}}}$). Since

$$\left(\{\rho_1\}^{-1}_{\rho_2}\gamma\right)\downarrow \;= \left(\left\{\{x\}^{z}_{y_{\mathsf{pub}}}\right\}^{-1}_{y_{\mathsf{priv}}}\alpha\right)\downarrow \;= K,$$

we must update $\Theta_F(\emptyset)$ by adding

$$\{h_9 \mapsto K\},$$

and must update $\Theta_T(\emptyset)$ by adding

$$\left\{ h_9 \mapsto \left\{ h_2(\{|h_1|\}_{h_6}^{-1}) \right\}_{K\mathsf{C}_{\mathsf{priv}}}^{-1} \right\}.$$

Iterating this process, we will find that in order to saturate $\Theta_F(\emptyset)$ we must still have M and N in the range of $\Theta(\emptyset)$, e.g.

$$h_{10}\Theta_F(\emptyset) = M \qquad \text{and} \qquad h_{11}\Theta_F(\emptyset) = N,$$

and

$$h_{10}\Theta_T(\emptyset) = \{|h_3|\}^{-1}_{\left\{h_2(\{|h_1|\}_{h_6}^{-1})\right\}_{K\mathsf{C}_{\mathsf{priv}}}^{-1}} = \zeta_1,$$

$$h_{11}\Theta_T(\emptyset) = \pi_2 \left(\{|h_4|\}^{-1}_{\left\{h_2(\{|h_1|\}_{h_6}^{-1})\right\}_{K\mathsf{C}_{\mathsf{priv}}}^{-1}} \right).$$

However, Φ_s is saturated after updating it only once to include

$$\left\{ h_7 \mapsto \{|h_1|\}_{h_6}^{-1} \right\}.$$

Because the attacker cannot use h_6 to instantiate ρ, the attacker cannot instantiate any further relevant rewrite rules, and all terms that can be deduced from Φ_s using (Φ, D)-recipes can also be syntactically constructed from Φ_s.

Algorithm 6 is a procedure for computing D-saturations of Φ.

The next theorem states the correctness of Algorithm 6. Its termination is guaranteed when $\mathcal{R}$ is subterm convergent and Φ is grounded by D.

Theorem 5. *If Algorithm 6 terminates, then its output Θ is a D-saturation of Φ.*

4.4 D-Static Equivalence

Our D-static equivalence procedure uses the D-saturation technique introduced in Section 4.3. Given frames Φ and Φ', we obtain a finite representation of all D-bindings of each frame by means of their D-saturations. From a D-saturation of Φ, we can obtain a finite set of tests which are sufficient to decide whether Φ' satisfies all equalities satisfied by Φ or not. This finite set of tests is described by Theorem 6.

This algorithm is a fundamental component of our procedure for deciding the equivalence of constraint systems, detailed in Section 4.5.2.

Remark 1. Let Θ be a saturation of a frame Φ computed by Algorithm 6. Let $\theta \in dom(\Theta)$, $h \in dom(\Theta_F(\theta))$, $h_s \in sub(h\Theta_F(\theta))$, $l \in \mathcal{R}_L$. Then, there exists a finite complete set $\Delta(l, h, h_s, \theta)$ of $(\Theta_F(\theta), \Theta_T(\theta)(D_\theta))$-solutions of

$$\left\{ l \stackrel{?}{=} h_s\Theta_F(\theta) \right\}.$$

Algorithm 6 D-saturation algorithm.

`Input:` a frame Φ and a DCS D for Φ

`Output:` a D-saturation of Φ

$\theta \leftarrow id(dom(D))$
$\Theta_F \leftarrow \{\theta \mapsto \Phi\downarrow\}$
$\Theta_T \leftarrow \{\theta \mapsto \{h \mapsto (h, \emptyset) \mid h \in dom(\Phi)\}\}$
saturated $\leftarrow$ **false**
while saturated $=$ **false**
  saturated $\leftarrow$ **true**
  for $\theta \in dom(\Theta), l \in \mathcal{R}_L, h \in dom(\Theta_F(\theta))$,
  and $h_s \in sub(h\Theta_F(\theta)) \setminus \mathcal{X}_R$
    $\mathcal{U} \leftarrow \left\{l \stackrel{?}{=} h_s\right\}$
    $\Delta \leftarrow \mathsf{genUnif}(\Theta_F(\theta), \Theta_T(\theta)(D_\theta), \mathcal{U})$
    for $(\alpha, \gamma) \in \Delta$
      $t \leftarrow ((h, \gamma)\Theta_F(\theta))\downarrow$
      $\theta' \leftarrow (\Theta_T(\theta) \circ \theta_{\Theta_T(\theta)(D_\theta),\gamma}) \circ \theta$
      if $\theta' \in dom(\Theta_F)$ **and** $t\downarrow \in \Theta_F(\theta')[T(\Theta_F(\theta'), id(dom(D)))]$
        continue
      saturated $\leftarrow$ **false**
      choose $h_* \notin dom(\Theta_F(\theta))$
        $\Theta_F(\theta') \leftarrow \Theta_F(\theta') \cup \{h_* \mapsto t\}$
        $\Theta_T(\theta') \leftarrow \Theta_T(\theta') \cup \{h_* \mapsto \Theta_T(\theta')((h, \gamma))\}$
      else
        $F' \leftarrow ((\Theta_F(\theta) \circ \theta_{\Theta_F(\theta)(D_\theta),\gamma}) \circ \Theta_F(\theta))\downarrow \cup \{h_* \mapsto t\}$
        $T' \leftarrow (\Theta_T(\theta) \circ \theta_{\Theta_T(\theta)(D_\theta),\gamma}) \circ \Theta_T(\theta)$
        $T' \leftarrow T' \cup \{h_* \mapsto (h, \gamma)T'\}$
        $\Theta_F \leftarrow \Theta_F \cup \{\theta' \mapsto F'\}$
        $\Theta_T \leftarrow \Theta_T \cup \{\theta' \mapsto T'\}$
return $\Theta = (\Theta_F, \Theta_T)$

If $(\alpha, \gamma) \in \Delta(l, h, h_s, \theta)$, then there exists

$$\zeta \in T(\Theta_F(\theta'), \Theta_T(\theta')(D_{\theta'}))$$

such that

$$\zeta\Theta_F(\theta') = ((h, \gamma)\Theta_F(\theta))\downarrow,$$

where

$$\theta' = (\Theta_T(\theta) \circ \theta_{\Theta_T(\theta)(D_\theta),\gamma}) \circ \theta.$$

This is a consequence of the instructions in lines 17—18 and 23—24 of Algorithm 6. Both θ' and ζ can be trivially computed from h, h_s, and γ; this is useful to decide static equivalence, as described by Theorem 6.

The following theorem is our main result on static equivalence. It provides a finite set of recipes that are sufficient to test D-static equivalence between two frames. Its termination depends on the termination of the saturation procedure (for both frames) and the termination of several calls to our (Φ, D)-unification algorithm.

Theorem 6. *Let Φ^1 and Φ^2 be frames such that $T(\Phi^1) = T(\Phi^2)$, D be a DCS for Φ^1 (or equivalently, for Φ^2), and Θ^1, Θ^2 be D-saturations of Φ^1 and Φ^2, respectively. Suppose that, for all $i \in \{1, 2\}$:*

- *for all $\theta \in dom(\Theta^i)$ and all $h \in dom(\Theta_F(\theta))$, $\Delta^i(h, \theta)$ is a finite complete set of $(\Theta^i_F(\theta), \Theta_T(\theta)(D_\theta))$-solutions of $\left\{\rho \stackrel{?}{=} h\Theta^i_F(\theta)\right\}$, where ρ is a fresh recipe variable;*

- *for all $\theta \in dom(\Theta^i)$, all $h \in dom(\Theta^i_F(\theta))$, all $h_s \in sub(h\Theta^i_F(\theta))$, and all $l \in \mathcal{R}_L$, $\Delta^i(l, h, h_s, \theta)$ is a finite complete set of $(\Theta^i_F(\theta), \Theta_T(\theta)(D_\theta))$-solutions of $\left\{l \stackrel{?}{=} h_s\right\}$;*

Then, $\Phi^1 \approx^s_{\mathcal{R},D} \Phi^2$ if and only if, for all $i \in \{1, 2\}$ and all $\theta \in dom(\Theta^i)$:

- *for all $\gamma \in \Delta(h, \theta)$,*

$$\rho\gamma\Theta_T(\theta)\Phi^{3-i} \approx_{\mathcal{R}} (h, \gamma)\Theta_T(\theta)\Phi^{3-i};$$

- *for all $(\alpha, \gamma) \in \Delta(l, h, h_s, \theta)$, we have*

$$(h, \gamma)\Theta_T(\theta)\Phi^{3-i} \approx_{\mathcal{R}} \zeta\Theta_T(\theta)\Phi^{3-i},$$

where θ' and ζ are as in Remark 1.

4.5 Constraint Systems

In this section we describe our procedure for deciding the equivalence of constraint systems. The main idea is to obtain finite descriptions (using terms with recipe variables) of all the possible solutions to the deducibility constraints, the positive constraints, and the negative constraints.

In Section 4.5.1 describes our notion of complete set of solutions of a positive constraint system and a procedure for computing such complete sets. This procedure is used as a subroutine of our procedure for deciding the equivalence of constraint systems, detailed in Section 4.5.2.

Throughout this section, we assume that Φ is a frame and D is a DCS for Φ.

4.5.1 Positive Constraints

We have shown in Section 4.3 that, if Θ is a D-saturation of Φ, then any D-binding of Φ can be seen as an instance of some D-binding $\theta \in dom(\Theta)$. If there exists a finitary unification algorithm for $\approx_R$, then such a set of D-bindings may be restricted to a complete set of D-bindings of Φ that satisfy the set P of positive constraints. In this section we define complete sets of solutions of a positive constraint system, and describe our procedure for computing such finite sets.

Complete sets of solutions of positive constraints. We say that a set Θ^+ is a *complete set of D-bindings of Φ satisfying P* if, for all $\theta \in \Theta^+$, θ is a D-binding of Φ that satisfies P and, whenever θ' is a ground D-binding of Φ that satisfies P, there exists $\theta \in \Theta^+$ and a (Φ_θ, D_θ)-recipe substitution γ such that

- $\theta\gamma\Phi \approx_{\mathcal{R}} \theta'\Phi$,
- and, for all $\zeta' \in T(\Phi, \theta')$, there exists $\zeta \in T(\Phi, \theta)$ such that

$$(\zeta'\Phi)\downarrow = (\zeta\Phi)\downarrow\,(\Phi \circ \gamma)\downarrow\,.$$

Theorem 7 provides a simple algorithm for obtaining such finite complete sets.

Theorem 7. *Let Φ be a frame, D be a DCS, and Θ be a saturation of Φ. Let P be a set of positive constraints, i.e., a set of pairs $(t \stackrel{?}{\approx}_{\mathcal{R}} t')$, where $t, t' \in T_\Sigma(\mathcal{N} \cup dom(D))$. For each $\theta \in dom(\Theta)$, let Υ_θ be a finite complete set of $\approx_{\mathcal{R}}$-unifiers for the set*

$$\bigcup_{(t \stackrel{?}{\approx}_{\mathcal{R}} t') \in P} \left\{ t\theta\Phi \stackrel{?}{\approx}_{\mathcal{R}} t'\theta\Phi \right\}.$$

Suppose that, for each $\theta \in dom(\Theta)$ and each $\varrho \in \Upsilon_\theta$, $\Delta_{\theta,\varrho}$ is a finite complete set of $(\Theta_F(\theta), \Theta_T(\theta)(D_\theta))$-solutions of the unification problem

$$\bigcup_{\rho \in dom(D_\theta)} \left\{ \rho \stackrel{?}{=} (\rho\varrho)\downarrow \right\}.$$

Then,

$$\Theta^{+} = \bigcup_{\theta\in\Theta}\left(\bigcup_{\varrho\in\Upsilon_\theta}\left(\bigcup_{(\alpha,\gamma)\in\Delta_{\theta,\varrho}}\{(\gamma\Theta_T(\theta))\circ\theta\}\right)\right)$$

is a finite complete set of D-bindings of Φ that satisfy P.

Theorem 7 yields a (possibly non-terminating) algorithm for computing finite complete sets of D-bindings of Φ which satisfy a set P of positive constraints. We denote by genUnifP this algorithm and we write $\mathsf{genUnifP}(\Phi, D, P)$ for its output on input (Φ, D, P) (assuming that it terminates).

4.5.2 Equivalence of Constraints Systems

Given procedures for (Φ, D)-unification, D-saturation, finitary unification under $\approx_{\mathcal{R}}$, and static equivalence, we can decide the equivalence of sets of constraint systems by using Algorithm 6.

Intuitively, consider two constraint systems $C = (\Phi, D, P, N)$ and $C' = (\Phi', D', P', N')$ such that $T(\Phi) = T(\Phi')$ and the set of D-bindings of Φ coincides with the set of D'-bindings of Φ'. We begin by computing finite complete sets of D-bindings of Φ that satisfy the deducibility constraints in D and the positive constraints in P. Each D-binding θ in this set that is a solution of C (i.e., satisfies also the negative constraints) must also be a solution of C', and the resulting frames Φ_θ and Φ'_θ must be D_θ-statically equivalent. Moreover, we must ensure that, if $\theta\gamma$ is a D'-binding of Φ' for which some negative constraint in N' is not satisfied, then $\theta\gamma$ must not be a solution of C. To test this in a finite manner we again rely on our (Φ, D)-unification algorithm. This procedure is described in detail in Algorithm 7.

Algorithm 7 Procedure for deciding the equivalence of constraint systems.

Input: Two constraint systems $C^1 = (\Phi^1, D^1, P^1, N^1)$ and $C^2 = (\Phi^2, D^2, P^2, N^2)$

Output: `true` if $C_1 \sim C_2$ are equivalent, `false` otherwise

1: **for** $\theta \in \mathsf{genUnifP}(\Phi^1, D^1, P^1)$
2: **choose** ι: $dom(D_\theta) \rightarrowtail \mathcal{N} \setminus (names(\Phi^1_\theta) \cup names(\Phi^2_\theta))$
3: **if** $\theta\iota \in \mathsf{sol}(C^1) \setminus \mathsf{sol}(C^2)$ **or** $\Phi^1_\theta \not\approx^s_{\mathcal{R},D_\theta} \Phi^2_\theta$
4: **return false**
5: **for** $(t \stackrel{?}{\neq} t') \in N$
6: **for** $\theta' \in \mathsf{genUnifP}(\Phi^2_\theta, D^2, \left\{t \stackrel{?}{=} t'\right\})$
7: **if** $\theta' \circ \theta \in \mathsf{sol}(C^1)$
8: **return false**
9: execute lines 1—8 reverting the roles of C^1 and C^2
10: **return true**

Theorem 8 establishes the correctness of the algorithm. Its proof can be found in Appendix B.

Theorem 8. *If Algorithm 6 terminates on input (C_1, C_2), then its output is* `true` *if $C_1 \sim C_2$ and* `false` *otherwise.*

4.5.3 Termination

Our algorithm does not terminate in general. We conjecture however that the algorithm terminates for subterm convergent rewriting systems.

More precisely, we say that a frame Φ is *ground* by a deducibility constraint sequence D if, for all $h \in dom(\Phi)$, either $h\Phi = f(\rho_1, \ldots, \rho_n)$ for some $f \in \Sigma_n$ and some (distinct) recipe variables $\rho_1, \ldots, \rho_n$, or $rvars(h\Phi) \subseteq dom(D)$. Note that, if Φ is ground by the empty deducibility constraint sequence ϵ, then Φ represents a ground frame in the usual sense, since including terms of the form $f(\rho_1, \ldots, \rho_n)$ in the range of Φ serves the purpose of allowing the attacker to apply function symbols, which is assumed in the standard formalization of an attacker's capabilities.

We conjecture that, if a frame Φ is ground by a deducibility constraint sequence D and $\mathcal{U}$ is any unification problem, then Algorithm 5 ((Φ, D)-unification) terminates on the input $(\Phi, D, id(dom(D)), \mathcal{U})$. The intuitive reason for this conjecture is that, during the execution of Algorithm 5, the only terms in the range of Φ that may increase the number of function symbols occurring in the unification problem are those where recipe variables occur and which are not of the form $f(\rho_1, \ldots, \rho_n)$ for some $f \in \Sigma_n$ and some recipe variables $\rho_1, \ldots, \rho_n$. However, since all recipe variables occurring in such terms are bound by D, these terms may not be arbitrarily nested, as in Example 13. Therefore, we believe that it is possible to impose a bound on the depth of the instantiations of each recipe variable in $dom(D)$ that may result from the execution of Algorithm 5, implying the termination of the algorithm.

If this conjecture holds, then it is straightforward to check that, throughout the execution of Algorithm 6 on input (Φ, D), all queries to Algorithm 5 have inputs satisfying the condition in our conjecture, and thus all unification queries terminate during the execution of the saturation algorithm.

We further conjecture that, assuming these conditions and a subterm convergent rewriting system $\mathcal{R}$, Algorithm 6 terminates on input (Φ, D). This conjecture is similar to the result that only subterms of the range of the frame Φ may occur during the saturation procedure under a subterm convergent rewriting system, and thereby Algorithm 3 terminates. Together, these two conjectures imply the decidability of trace equivalence for simple processes and subterm convergent equational theories.

Chapter 5

Symbolic Probabilistic Protocol Analysis

In this chapter we describe a framework for the symbolic probabilistic analysis of security protocols. We use as a running example and case study a well-known off-line guessing attack to the EKE protocol when implemented using RSA asymmetric encryption [40].

First we show how security relevant properties of cryptographic primitives can be expressed in our framework by means of relations between the input and output of cryptographic primitives. We illustrate the expressiveness of this framework by using it to model a random number generation algorithm that generates bitstrings representing primes of a certain length and an asymmetric cryptosystem whose valid public keys have some recognizable structure relevant in the analysis of our case study. We then show how such properties can be used to find attacks and estimate their success probability. Finally, we specialize our framework to the (automated) analysis of security against off-line guessing attacks.

In addition to the properties of cryptographic primitives that can be described in our framework and are used for probabilistic reasoning, we also consider equational properties of primitives, as usual. In contrast with the previous chapter, however, we assume that the underlying equational theory is a subterm theory, i.e., it is generated by a subterm convergent rewriting system $\mathcal{R}$.

Note that this framework only specifies a method for computing probabilities given a description of the cryptographic primitives and their implementations; other techniques must still be used to determine what are the relevant probabilities in the analysis of a given security protocol. In our EKE example we are concerned with off-line guessing attacks, and thus the equality tests that an attacker performs are determined by using standard techniques for deciding symbolic equivalence, such as those discussed in Chapters 3 and 4.

5.1 Our Probabilistic Setup

In this section we introduce the basic notions that are used in our framework. Concretely, we define the syntax and semantics of our property statements and illustrate their use in modeling properties of cryptographic primitives. We also describe our association of symbolic terms to random variables whose values are bitstrings.

Property statements. Intuitively, *property statements* in our framework express relations between the input and output of a given cryptographic primitive.

More precisely, we assume fixed a countable set $\mathcal{T}$ of *types*. Given a signature Σ, a *property statement* is a tuple $(f, T_1, \ldots, T_n, T)$, written

$$f[T_1, \ldots, T_n] \subseteq T,$$

where $f \in \Sigma_n$ and $T_1, \ldots, T_n, T \in \mathcal{T}$.

If $ps = (f[T_1, \ldots, T_n] \subseteq T)$, we introduce the following definitions:

- the *head symbol* of ps by $head(ps) = f$;
- the *domain of* ps by $dom(ps) = T_1 \times \cdots \times T_n$;
- the *range of* ps by $ran(ps) = T$.

Given a set PS of property statements and $f \in \Sigma$, we denote by PS_f the set of property statements in PS whose head symbol is f. Note that property statements are not simply type assertions: for example, we may have more than one property statement associated to each function symbol. We write $f[T_1, \ldots, T_n] \subseteq_{PS} T$ instead of $(f[T_1, \ldots, T_n] \subseteq T) \in PS$.

Syntax. The syntax of our setup is defined by a four-tuple

$$\langle \Sigma, \approx_{\mathcal{R}}, \mathcal{T}, PS \rangle,$$

where:

- Σ is a signature;
- $\approx_{\mathcal{R}}$ is an equational theory on T_Σ defined by a convergent rewriting system $\mathcal{R}$;
- $\mathcal{T}$ is a set of type names;
- PS is a set of property statements.

To simplify the presentation, it is convenient to treat names as constant function symbols. Formally, this means that the term algebra we consider in this chapter is $T_{\Sigma \cup \mathcal{N}}(\emptyset)$ instead of $T_\Sigma(\mathcal{N})$. Here, we assume that all names $a \in \mathcal{N}$ have arity 0, i.e., $ar(a) = 0$, or equivalently $(\Sigma \cup \mathcal{N})_0 = \mathcal{N} \cup \Sigma_0$.

These two notational choices are equivalent in terms of equational reasoning. We will write T_Σ instead of $T_{\Sigma \cup \mathcal{N}}$.

Interpretation functions. We write $\mathcal{B}$ for $\{0,1\}$. Given a set $\mathcal{T}$ of types, a *type interpretation function* is a function $[\![\cdot]\!] : \mathcal{T} \to \mathcal{P}(\mathcal{B}^*)$ that associates types to sets of bitstrings such that $[\![T]\!]$ is finite and non-empty for every $T \in \mathcal{T}$. Intuitively, type interpretation functions associate to each type a set of bitstrings, thereby providing a meaning to types. We extend this interpretation function to products, and define

$$[\![T_1 \times \cdots \times T_n]\!] = [\![T_1]\!] \times \ldots \times [\![T_n]\!] .$$

A *setup specification* is a pair $\mathcal{S} = \langle S, [\![\cdot]\!] \rangle$, where $S = \langle \Sigma, \approx_{\mathcal{R}}, \mathcal{T}, PS \rangle$ is a four-tuple defining the syntax of the setup as in the above paragraph and $[\![\cdot]\!]$ is an interpretation function, which consistently defines the behavior of all function symbols. Concretely, we require that:

- for all $f \in \Sigma$, $PS_f \neq \emptyset$;
- if $ps_1, ps_2 \in PS_f$, then $[\![dom(ps_1)]\!] \cap [\![dom(ps_2)]\!] = \emptyset$.

For $c \in \Sigma_0$, these conditions imply that there is a single $T \in \mathcal{T}$ such that $c \subseteq_{PS} T$. We denote this unique T by $type(c)$.

We assume that the function represented by a function symbol is undefined unless otherwise stated by a property statement concerning that function symbol: That is, we assume that, if $f \in \Sigma_n$ and there is no $ps \in PS_f$ such that $(b_1, \ldots, b_n) \in [\![dom(ps)]\!]$, then the function represented by the symbol f is undefined on the input $(b_1, \ldots, b_n)$. Under these conditions, we set the *domain of definability* of f to be $dom_{\mathcal{S}}(f) = \biguplus_{ps \in PS_f} [\![dom(ps)]\!]$. Note that

$$\emptyset \subsetneq dom_{\mathcal{S}}(f) \subseteq (\mathcal{B}^*)^n$$

for all $f \in \Sigma_n$.

Example 17. In this example we show how the notion of property statement defined above can be used to specify a simple yet realistic setup using a hash function h that maps any bitstring to a bitstring of length 256, a pairing function that given any pair of bitstrings returns their labeled concatenation, and a symmetric encryption scheme that uses a block cipher together with some reversible padding technique.

The equational theory of pairing, projection and symmetric encryption are as in Example 1. Our case study of the EKE protocol using RSA requires a more precise equational modeling of asymmetric encryption. We introduce our model of RSA encryption, consisting of an equational theory and a set of property statements, in detail in Example 18.

The types we will consider to model the remaining primitives, as well as their respective interpretations under $[\![\cdot]\!]$, are as follows:

- pw represents weak (e.g., human-chosen) passwords. We model these passwords as being encoded by 256-bit bitstrings, but sampled from a relatively small set: thus, $[\![\mathsf{pw}]\!] \subset \mathcal{B}^{256}$ and $|\,[\![\mathsf{pw}]\!]\,| = 2^{24}$;

- $\mathsf{sym\_key}$ represents symmetric keys, with $[\![\mathsf{sym\_key}]\!] = \mathcal{B}^{256}$;
- text represents one block of plaintext, with $[\![\mathsf{text}]\!] = \mathcal{B}^{256}$;
- $T_{\mathcal{B}^n}$ with $[\![T_{\mathcal{B}^n}]\!] = \mathcal{B}^n$ for each $n \in \mathbb{N}$;
- $T_{\mathcal{B}^{(n,m)}}$ with $[\![T_{\mathcal{B}^{(n,m)}}]\!] = \mathcal{B}^{(n,m)} = \bigcup_{i=n}^{m} \mathcal{B}^i$ for each $n, m \in \mathbb{N}$; and
- $T_{\mathcal{B}^{n\#m}}$ represents the set of bitstrings that are labeled concatenations of two bitstrings of size n and m. Thus, for each $n, m \in \mathbb{N}$, we have

$$[\![T_{\mathcal{B}^{n\#m}}]\!] = \mathcal{B}^{n\#m} \subseteq \mathcal{B}^{n+m+\lceil \log(n+m) \rceil}.$$

We define the set PS by

$$\begin{aligned} PS = \{ \quad & h[T_{\mathcal{B}^n}] \subseteq T_{\mathcal{B}^{256}}, \pi_1[T_{\mathcal{B}^{n\#m}}] \subseteq T_{\mathcal{B}^n}, \pi_2[T_{\mathcal{B}^{n\#m}}] \subseteq T_{\mathcal{B}^m}, \\ & \langle T_{\mathcal{B}^n}, T_{\mathcal{B}^m} \rangle \subseteq T_{\mathcal{B}^{n\#m}}, \\ & \{|T_{\mathcal{B}^{(256n+1,256(n+1)}}|\}_{T_{\mathcal{B}^{256}}} \subseteq T_{\mathcal{B}^{256(n+1)}}, \\ & \{|T_{\mathcal{B}^{256(n+1)}}|\}^{-1}_{T_{\mathcal{B}^{256}}} \subseteq T_{\mathcal{B}^{(256n+1,256(n+1))}} \mid n, m \in \mathbb{N}\}. \end{aligned}$$

Note that all functions are modeled as undefined on all arguments that fall outside the domains of these property statements. For example, symmetric encryption of any term is undefined unless the key is a 256-bit bitstring.

Example 18. We use our framework to formalize RSA encryption, taking into account properties of the key generation algorithm. An RSA public key is a pair (n, e), where $n = p \cdot q$, the *modulus*, is a product of large primes (typically around 512 bits) p and q, and e, the *exponent*, is relatively co-prime to $\varphi(n) = (p-1)(q-1)$. The private key d is the multiplicative inverse of e modulo $\varphi(n)$.

We extend the setup specification of Example 17. We add to the signature the following five primitives: the unary functions mod, expn, and inv, representing the extraction of the modulus, the exponent, and the exponent's multiplicative inverse, respectively, from a randomly generated RSA public-private key pair; a binary function $\{\cdot\}^{-1}_{\cdot} \in \Sigma_2^{\mathcal{DY}}$, representing the RSA decryption function; and a ternary function $\{\cdot\}^{\cdot}_{\cdot}$, representing RSA encryption.

The only rewriting rule that we must add to model RSA encryption is

$$\left\{ \{m\}^{\mathsf{expn}(k)}_{\mathsf{mod}(k)} \right\}^{-1}_{\mathsf{inv}(k)} \to m,$$

where m, k are variables.

The additional types that we will use to formalize relevant properties of these functions and their interpretations are as follows:

- random represents the random values used to generate an RSA public-private key pair, including two 512-bit prime numbers and the 1024-bit exponent, with $[\![\mathsf{random}]\!] \subseteq \mathcal{B}^{2048}$;

- prodprime represents the product of two 512-bit prime numbers, so that $[\![\mathsf{prodprime}]\!] \subseteq \mathcal{B}^{1024}$ and $|\,[\![\mathsf{prodprime}]\!]\,| \approx 2^{1008}$ (by the prime number theorem);
- odd represents 1024-bit odd numbers, with $[\![\mathsf{odd}]\!] \subseteq \mathcal{B}^{1024}$ and $|\,[\![\mathsf{odd}]\!]\,| = 2^{1023}$.

We add the following property statements:

- $\mathsf{mod}[\mathsf{random}] \subseteq \mathsf{prodprime}$, because the modulo of an RSA public key is the product of two primes.
- $\mathsf{expn}[\mathsf{random}] \subseteq \mathsf{odd}$, because the exponent of an RSA public key is always odd.
- $\mathsf{inv}[\mathsf{random}] \subseteq T_{\mathcal{B}^{1024}}$, because an RSA private key is a 1024-bit bitstring. Note that we do not allow extracting modulos, exponents, or inverses from anything other than a valid value for generating an RSA key pair.
- $\{T_{\mathcal{B}^{1024}}\}^{\mathsf{odd}}_{\mathsf{prodprime}} \subseteq T_{\mathcal{B}^{1024}}$. This property states that encrypting any 1024-bit plaintext with a valid RSA public key yields a 1024-bit bitstring. Note that the encryption is undefined if the plaintext is not a 1024-bit bitstring, the modulus is not the product of two primes, or the exponent is even.
- $\{T_{\mathcal{B}^{1024}}\}^{-1}_{T_{\mathcal{B}^{1024}}} \subseteq T_{\mathcal{B}^{1024}}$. RSA decryption takes a ciphertext and a private key which are both 1024-bit bitstrings, and outputs a 1024-bit plaintext.

For simplicity, we merely require here that the public-key exponent is odd, rather than requiring it to be co-prime with the modulus.

In the rest of this section we assume fixed a setup specification

$$\mathcal{S} = \langle\langle \Sigma, \approx_R, \mathcal{T}, PS\rangle, [\![\cdot]\!]\rangle.$$

Term assignments. Term assignments associate a value to each symbolic term. Recall that the domain of definability of a function symbol $f \in \Sigma_n$, $dom_{\mathcal{S}}(f)$, may be a proper subset of $(\mathcal{B}^*)^n$. However, no restriction is placed on the construction of symbolic terms. Therefore, an error/undefined value, represented by $\perp$, is also considered. We will write $\mathcal{B}^*_\perp$ for the set $\mathcal{B}^* \cup \{\perp\}$. Note that $\perp$ never occurs in the domain of definability of any function; thus, applying a function to an undefined argument always yields an undefined result, as expected.

Define $\mathbf{\Omega}$ by $\mathbf{\Omega} = \{\omega \mid \omega\colon T_\Sigma \to \mathcal{B}^*_\perp\}$, i.e., $\mathbf{\Omega}$ is the set of all functions mapping symbolic terms to bitstrings (or the undefined value $\perp$). We now define the notions of satisfaction of an equational theory, a property statement, a set of property statements, and a setup specification by a function $\omega \in \mathbf{\Omega}$ as follows.

- ω *satisfies* $\approx_{\mathcal{R}}$, written $\omega \models \approx_{\mathcal{R}}$, if, whenever $t \approx_{\mathcal{R}} t'$, either $\omega(t) = \omega(t')$, or $\omega(t) = \perp$, or $\omega(t') = \perp$;

- ω satisfies a property statement ps (under $[\![\cdot]\!]$), written $\omega \models_{[\![\cdot]\!]} ps$ if

$$(\omega(t_1), \ldots, \omega(t_n)) \in [\![dom(ps)]\!] \quad \Rightarrow \quad \omega(f(t_1, \ldots, t_n)) \in [\![ran(ps)]\!]\,;$$

- ω satisfies PS (under $[\![\cdot]\!]$), written $\omega \models_{[\![\cdot]\!]} PS$, if $\omega \models_{[\![\cdot]\!]} ps$ for all $ps \in PS$ and

$$\left(\forall_{ps \in PS_f} (\omega(t_1), \ldots, \omega(t_n)) \notin [\![dom(ps)]\!]\right) \quad \Rightarrow \quad \omega(f(t_1, \ldots, t_n)) = \bot;$$

- ω satisfies a setup presentation $\mathcal{S}$, written $\omega \models \mathcal{S}$, if $\omega \models \approx_R$ and $\omega \models_{[\![\cdot]\!]} PS$.

Note that the notions of satisfaction of a property statement and a set of property statements are defined for an interpretation function $[\![\cdot]\!]$. We write $\Omega_{\mathcal{S}}$ for the set of all $\omega \in \mathbf{\Omega}$ that satisfy $\mathcal{S}$.

Example 19. Functions ω that satisfy our equational theory may be such that $\omega(t) = \bot$ and $\omega(t') \neq \bot$ for terms t and t' such that $t \approx_R t'$. To see why this is allowed, recall from Example 17 that $\{|\cdot|\}_{\cdot}^{-1}$ represents a symmetric encryption algorithm in which valid keys always have 256 bits. Let $t, k \in \Sigma_0$, with $type(t) = \mathsf{text}$, and $t' = \{|\{|t|\}_k|\}_k^{-1}$. We have $t \approx_R t'$. Now, if ω represents a possible real-world assignment (of terms to bitstrings), we have $\omega(t) \neq \bot$ (since t represents a bitstring freshly sampled from $\mathcal{B}^{256}$). Moreover, if $\omega(k)$ is not a 256-bit bitstring, then $\omega(t') = \bot$ since our encryption and decryption functions are only defined for 256-bit keys. Therefore, $\omega(\{|\{|t|\}_k|\}_k^{-1}) = \bot$.

Finitely-generated events. Valid protocol execution traces are finite and, therefore, contain only finitely many terms. We are therefore interested in events that depend on only finitely many terms. For each finite set of terms $K \subseteq T_\Sigma$, let Λ_K be the set of functions $\lambda\colon K \to \mathcal{P}(\mathcal{B}^*_\bot)$ and, for each $\lambda \in \Lambda_K$, let

$$\Omega_\lambda = \{\omega \in \mathbf{\Omega} \mid \omega(t) \in \lambda(t) \text{ for each } t \in K\}.$$

Let

$$\Lambda = \bigcup_{K \in \mathcal{P}_{\mathit{fin}}(T_\Sigma)} \Lambda_K$$

and

$$\Omega_\Lambda = \{\Omega_\lambda \mid \lambda \in \Lambda\}\,,$$

where $\mathcal{P}_{\mathit{fin}}(X)$ is the set of *finite* subsets of X. Note that Ω_Λ is the set of subsets of $\mathbf{\Omega}$ whose specification depends on only the instantiation of finitely many terms. We want our probability measure to be defined in the σ-algebra generated by Ω_Λ. Let $\mathcal{F}$ be this σ-algebra; we say that $\mathcal{F}$ is the σ-algebra of *finitely generated events*.

Notation for events. The following notations will be used in proving several of our results. For each finite set of terms K, let Ψ_K be the set of functions $\psi\colon K \to \mathcal{B}^*_\perp$. Define

$$\Psi = \bigcup_{K \in \mathcal{P}_{fin}(T_\Sigma)} \Psi_K.$$

For each $\psi \in \Psi$, let

$$\Omega_\psi = \{\omega \in \mathbf{\Omega} \mid t \in dom(\psi) \Rightarrow \omega(t) = \psi(t)\}.$$

Let

$$\Omega_\Psi = \{\Omega_\psi \mid \psi \in \Psi\}$$

and, for each K,

$$\Omega_{\Psi_K} = \{\Omega_\psi \mid \psi \in \Psi_K\}.$$

Furthermore, for each $\lambda \in \Lambda_K$, define

$$\Psi(\lambda) = \{\psi \in \Psi_K \mid t \in K \Rightarrow \psi(t) \in \lambda(t)\}.$$

We remark that

$$\Omega_\lambda = \biguplus_{\psi \in \Psi(\lambda)} \Omega_\psi,$$

and that $\Psi(\lambda)$ is the only subset of Ψ_K such that the above equality holds.

Probabilistic models. Given a setup specification $\mathcal{S}$, we consider probability spaces $(\mathbf{\Omega}, \mathcal{F}, \mu)$, where $\mathbf{\Omega}$ and $\mathcal{F}$ are as defined above and $\mu\colon \mathcal{F} \to [0,1]$ is a probability measure. Note that $\mathbf{\Omega}$ and $\mathcal{F}$ are fixed for a given $\mathcal{S}$; we analyze the probability measure μ.

If $t \in T_\Sigma$, we write $\widehat{t}\colon \mathbf{\Omega} \to \mathcal{B}^*_\perp$ to denote the random variable on $\mathbf{\Omega}$ defined by $\widehat{t}(\omega) = \omega(t)$. We adopt standard (abuses of) notation from probability theory. If $C(b_1, \ldots, b_n)$ is a condition whose satisfaction depends on the bitstring values $b_1, \ldots, b_n$, we use the notational convention that

$$P_\mu[C(\widehat{t_1}, \ldots, \widehat{t_n})] = \mu(\{\omega \in \mathbf{\Omega} \mid C(\widehat{t_1}(\omega), \ldots, \widehat{t_n}(\omega))\}),$$

provided that

$$\{\omega \in \mathbf{\Omega} \mid C(\widehat{t_1}(\omega), \ldots, \widehat{t_n}(\omega))\} \in \mathcal{F}.$$

If $\Omega \in \mathcal{F}$, we will also write $P_\mu[\Omega]$ instead of $\mu(\Omega)$. We use standard notation for conditional probability. Namely, if $P_\mu[B] > 0$, then

$$P_\mu[A \mid B] = P_\mu[A, B]/P_\mu[B]$$

is the *conditional probability of A given B*.

We define the satisfaction of an equational theory, a set of property statements, and a setup specifications by a probability distribution μ as expected: μ satisfies each of these specifications if it gives probability 1 to the set of functions ω which satisfy them. More formally:

- μ satisfies $\approx_{\mathcal{R}}$, written $\mu \models \approx_{\mathcal{R}}$, if $\mu(\{\omega \mid \omega \models \approx_{\mathcal{R}}\}) = 1$;
- μ satisfies PS (under $[\![\cdot]\!]$), written $\mu \models_{[\![\cdot]\!]} PS$, if $\mu(\{\omega \mid \omega \models PS\}) = 1$;
- μ satisfies, or is a *model* of, the setup specification $\mathcal{S}$, written $\mu \models \mathcal{S}$, if $\mu \models \approx_{\mathcal{R}}$ and $\mu \models_{[\![\cdot]\!]} PS$.

We remark that μ is a model of $\mathcal{S}$ if and only if $\mu(\Omega_{\mathcal{S}}) = 1$.

5.2 A Generalized Random Oracle Model

In this section we propose an algorithm for sampling the random variables associated with symbolic terms. This sampling algorithm interprets functions as random oracles subject to satisfying our setup specification $\mathcal{S} = \langle\langle \Sigma, \approx_R, \mathcal{T}, PS\rangle, [\![\cdot]\!]\rangle$.

This algorithm is intended only to provide a clear definition of the probability measure that we use for a given a setup specification. In Section 5.3 we show how specific probabilities can be more efficiently computed using the probability distribution defined by this sampling algorithm.

5.2.1 Tentative term sampling in the ROM

Term sampling. Suppose that $K \subset T_\Sigma$ is a finite set of terms and P is a partition of K. We define $\approx_P$ to be the smallest congruence relation on T_Σ such that $\approx_{\mathcal{R}} \subseteq \approx_P$ and $t \approx_P t'$ whenever there is $p \in P$ such that $t, t' \in p$.

Algorithm 8 builds a function ψ_{ROM} mapping a finite set of terms to $\mathcal{B}^*_\perp$. We denote by $P(\psi_{ROM})$ the partition of $dom(\psi_{ROM})$ given by

$$P(\psi_{ROM}) = \left\{\psi^{-1}_{ROM}(b) \mid b \in ran(\psi_{ROM})\right\}.$$

The algorithm is probabilistic: in line 12, it samples a random bitstring from a finite subset of $\mathcal{B}^*_\perp$. We write $x \xleftarrow{U} S$ to denote that x is assigned a value sampled from a finite set S with uniform probability distribution. We also assume fixed some total order $\prec$ on the set of terms such that, if $t \in psub(t')$, then $t \prec t'$. We say that such an order is *subterm-compatible*.

Algorithm 8 samples terms in order (lines 2–3), by interpreting each function symbol as a random oracle with uniform probability distribution (line 12), and respecting the equational theory in case an equal term has already been sampled (lines 9–10), as long as all its argument values (previously sampled) are defined and form a tuple in its domain of definability (lines 5–6).

Problems with Algorithm 8. We show that Algorithm 8 does not necessarily yield the desired probability measure over $\mathcal{F}$.

Given a finite set $K \subseteq T_\Sigma$ and a subterm-compatible order $\prec$, Algorithm 8 is a probabilistic algorithm, and thus outputs a function $\psi \colon sub[K] \to \mathcal{B}^*_\perp$ with some probability distribution. We would therefore like to define a model μ of $\mathcal{S}$

Algorithm 8 (Tentative) term sampling algorithm.

Input: a finite set of terms $K \subseteq T_\Sigma$
Output: a function $\psi_{ROM}\colon sub[K] \to \mathcal{B}^*_\perp$

$\psi_{ROM} \leftarrow \emptyset$
choose $t_1, \ldots, t_k$ **s.t.** $t_1 \prec \ldots \prec t_k$ **and** $sub[K] = \{t_1, \ldots, t_k\}$
for $i \in \{1, \ldots, k\}$
  $f(t'_1, \ldots, t'_n) \leftarrow t_i$
  if $(\psi_{ROM}(t'_1), \ldots \psi_{ROM}(t'_n)) \notin dom_{\mathcal{S}}(f)$
    $\psi_{ROM}(t_i) \leftarrow \perp$
    continue
  let ps **be the unique** $ps \in PS_f$ **s.t.**
      $(\psi_{ROM}(t'_1), \ldots \psi_{ROM}(t'_n)) \in [\![dom(ps)]\!]$
  if there is $t' \in dom(\psi_{ROM})$ **s.t.**
      $t \approx_{P(\psi_{ROM})} t'$ **and** $\psi_{ROM}(t') \neq \perp$
    $\psi_{ROM}(t_i) \leftarrow \psi_{ROM}(t')$
    continue
  $\psi_{ROM}(t_i) \xleftarrow{U} [\![ran(ps)]\!]$
return ψ_{ROM}

by defining $\mu(\Omega_\lambda)$ for each generator Ω_λ of $\mathcal{F}$ as the probability that executing Algorithm 8 on input $dom(\lambda)$ yields as output a function ψ_{ROM} such that, for each $t \in dom(\lambda)$, $\psi_{ROM}(t) \in \lambda(t)$.

The next example shows that this is not well-defined in general. We consider two terms, t and a, and show that the algorithm samples t and a to the same bit-string with a probability that depends on the input set K and the order relation $\prec$. Thus, letting $\lambda_b = \{t \mapsto b, a \mapsto b\}$ for each $b \in \mathcal{B}^*_\perp$, we have that the probability of the (measurable) set $\bigcup_{b \in \mathcal{B}^*_\perp} \Omega_{\lambda_b}$ depends on the input set K and the order relation $\prec$ considered.

Example 20. Suppose that $a, b, k \in \Sigma_0$ are such that $type(a) = T_{\mathcal{B}^{1024}}$, $type(b) = T_{\mathcal{B}^{1024}}$ and $type(k) = \mathsf{random}$. Let

$$t = \left\{ \{a\}^{\mathsf{expn}(k)}_{\mathsf{mod}(k)} \right\}^{-1}_b ,$$

and consider executing Algorithm 8 on the set $\{t\}$. Algorithm 8 outputs a function $\psi\colon sub(t) \to \mathcal{B}^*_\perp$. Let us consider the probability that $\psi(t) = \psi(a)$. It is simple to check that both $\psi(t)$ and $\psi(a)$ are sampled by Algorithm 8 with uniform probability distribution from $\mathcal{B}^{1024}$. Therefore, we obtain

$$P[\psi(t) = \psi(a)] = 2^{-1024}$$

regardless of the order $\prec$ chosen.

Now, consider executing Algorithm 8 on the set $\{t, \mathsf{inv}(k)\}$. If $t \prec \mathsf{inv}(k)$, then the execution of Algorithm 8 will be exactly the same until $\psi(s)$ is sampled for all

terms $s \in sub(t)$, and $\psi(\mathsf{inv}(k))$ is only sampled afterwards. Therefore, $\psi(s)$ is sampled according to the same probability distribution for all $s \in sub(t)$, and the probability that $\psi(t) = \psi(a)$ is still 2^{-1024}. However, if $\mathsf{inv}(k) \prec b$, we have a probability of 2^{-1024} that $\psi(b) = \psi(\mathsf{inv}(k))$. If $\psi(b) = \psi(\mathsf{inv}(k))$, then we have $\psi(t) = \psi(a)$ with probability 1. Otherwise, $\psi(t)$ and $\psi(a)$ will still be sampled from $\mathcal{B}^{1024}$ with uniform probability distribution, and the probability that they are sampled to the same value is again 2^{-1024}. In this case, we conclude that

$$P[\psi(a) = \psi(t)] = 2^{-1024} \cdot (2 - 2^{-1024}) \neq 2^{-1024}.$$

Thus, the probability that $\psi(t) = \psi(a)$ depends on both the set of terms K input to the algorithm and the order $\prec$.

Nevertheless, the following result shows that, given a fixed finite set of terms K and a subterm-compatible order $\prec$, Algorithm 8 does yield a probability distribution on the σ-algebra $\mathcal{F}_K$ generated by the set $\{\Omega_\lambda \mid \lambda \in \Lambda_K\}$, i.e., the σ-algebra of events that depend only on the instantiation of terms in the set K.

Theorem 9. *There is a unique probability distribution $\mu^{K,\prec}\colon \mathcal{F}_{sub[K]} \to [0,1]$ such that, for each $\lambda \in \Lambda_K$, $\mu^{K,\prec}(\Omega_\lambda)$ is the probability that executing Algorithm 8 on input K and using the order $\prec$ yields a function ψ_{ROM} such that, for each $t \in K$, $\psi_{ROM}(t) \in \lambda(t)$.*

5.2.2 Revised term sampling in the ROM

To avoid problems like the one illustrated by Example 20 we impose two additional hypotheses on the setup specification $\mathcal{S}$. We will explicitly distinguish a set of weak function symbols and consider a revised algorithm that uses this distinction. This revised algorithm is equivalent to Algorithm 8 when all functions are treated as weak. We show that, under these hypotheses, we can define a probability measure from this new sampling algorithm, while also simplifying the calculation of probabilities.

Weak terms. We assume fixed a set $\Sigma^W \subseteq \Sigma$ of *weak function symbols*. We say that a term $t \in T_\Sigma$ is *weak* if $head(t) \in \Sigma^W$, and denote by T^W the set of weak terms.

Intuitively, weak function symbols are those that represent functions whose outputs are sampled from "small" sets, and a probabilistic model must therefore take into account the possibility of collisions between them. By contrast, non-weak function symbols are those that represent functions whose outputs are sampled from large enough sets, so that ignoring the possibility of collisions changes our probability estimates only negligibly. This idea is made precise by Theorem 12, below.

Example 21. In our running example, we consider the set of weak function symbols $\Sigma^W = \{\mathsf{h}\} \cup \{a \in \Sigma_0 \mid a \subseteq_{PS} \mathsf{pw}\}$. That is, a term is weak if it is an hash or if it is derived from a humanly-chosen password.

Term sampling revisited. If K and K' are sets of terms and P is a partition of K, we let $P|_{K'} = \{p \cap K' \mid p \in P\}$. Note that $P|_{K'}$ is a partition of $K \cap K'$. We denote by $W(\psi_{ROM})$ the partition $P(\psi_{ROM})|_{T^W}$.

Algorithm 9 is our revised term sampling algorithm, targeted at solving the anomaly described in Example 20. It is the same as Algorithm 8 with the exception that we replace the condition $t \approx_{P(\psi_{ROM})} t'$ by $t \approx_{W(\psi_{ROM})} t'$ in line 9.

Algorithm 9 Revised term sampling algorithm.

`Input:` a finite set of terms $K \subseteq T_\Sigma$
`Output:` a function $\psi_{ROM}\colon sub[K] \to \mathcal{B}^*_\perp$

1: $\psi_{ROM} \leftarrow \emptyset$
2: **choose** $t_1, \ldots, t_k$ **s.t.** $t_1 \prec \ldots \prec t_k$ **and** $sub[K] = \{t_1, \ldots, t_k\}$
3: **for** $i \in \{1, \ldots, k\}$
4: $\quad f(t'_1, \ldots, t'_n) \leftarrow t_i$
5: $\quad$ **if** $(\psi_{ROM}(t'_1), \ldots \psi_{ROM}(t'_n)) \notin dom_{\mathcal{S}}(f)$
6: $\quad\quad \psi_{ROM}(t_i) \leftarrow \perp$
7: $\quad\quad$ **continue**
8: $\quad$ **let** ps **be the unique** $ps \in PS_f$ **s.t.** $(\psi_{ROM}(t'_1), \ldots \psi_{ROM}(t'_n)) \in [\![dom(ps)]\!]$
9: $\quad$ **if there exists** $t' \in dom(\psi_{ROM})$ **s.t.** $t \approx_{W(\psi_{ROM})} t'$ **and** $\psi_{ROM}(t') \neq \perp$
10: $\quad\quad$ **continue**
11: $\quad \psi_{ROM}(t_i) \xleftarrow{U} [\![ran(ps)]\!]$
12: **return** ψ_{ROM}

This revised algorithm yields a probability distribution on $\mathcal{F}$ provided that the setup specification $\mathcal{S}$ satisfies two reasonable conditions: disjointness and compatibility. We discuss these conditions next.

Disjointness. The first condition we require on the specification $\mathcal{S}$ is that weak function symbols do not occur in the rewriting system $\mathcal{R}$.

Intuitively, this disjointness condition implies that the equality of terms depends only on the equalities between their weak subterms. Therefore, sampling terms in a different order does not affect any equalities because terms are sampled only after all their subterms have been sampled. This condition excludes cases like the one described in Example 20: because $\mathsf{inv} \notin \Sigma^W$, we never have

$$\left\{\{a\}^{\mathsf{expn}(k)}_{\mathsf{mod}(k)}\right\}^{-1}_b \approx_{W(\psi_{ROM})} a$$

(even if $\psi_{ROM}(b) = \psi_{ROM}(\mathsf{inv}(k))$). The key idea is that equalities between non-weak terms may be disregarded, as the terms are equal only with negligible probability. We remark that ignoring equalities between non-weak terms, besides allowing us to consistently define a probability measure, also simplifies the calculation of probabilities.

$\mathcal{R}$-closed partitions. Let K be a finite set of terms and P be a partition of K. Recall the definition of $\approx_P$ given in Section 5.2. We say that P is $\approx_{\mathcal{R}}$*-closed* if, for all $t, t' \in K$, if $t \approx_P t'$, then there is $p \in P$ such that $t, t' \in p$. We are interested in partitions of weak terms. Thus, given a finite set K, we denote by $\mathcal{P}^W_{\mathcal{R}}(K)$ the set of $\approx_{\mathcal{R}}$-closed partitions of $sub[K] \cap T^W$.

Example 22. Consider the set of terms

$$K = \left\{ \mathsf{h}(\{|\{|t|\}_k|\}^{-1}_{k'}), \mathsf{h}(t) \right\},$$

and suppose that

$$k \subseteq_{PS} \mathsf{pw}, \qquad k' \subseteq_{PS} \mathsf{pw}, \qquad \text{and} \qquad t \notin T^W.$$

Then,

$$\left\{ \left\{k, k'\right\}, \left\{ \mathsf{h}(\{|\{|t|\}_k|\}^{-1}_{k'}) \right\}, \{\mathsf{h}(t)\} \right\}$$

is a partition of $sub[K] \cap T^W$ that is not $\approx_{\mathcal{R}}$-closed.

Renaming. A P*-renaming* is an injective function $\tau \colon P \rightarrowtail \mathcal{N}_p$ mapping P-equivalence classes to names in $\mathcal{N}_p$. We will denote by P^* be the partition of

$$(T^W_\Sigma \cap sub[K]) \cup \tau[P]$$

given by

$$P^* = \{p \cup \{\tau(p)\} \mid p \in P\}.$$

Deciding $\approx_W$. We now show that, if the disjointness condition is satisfied, the relation $\approx_W$ is decidable: That is, there exists a procedure which, given a partition $P \in \mathcal{P}^W_{\mathcal{R}}(K)$ and two terms t and t', decides whether $t \approx_P t'$. We proceed as follows: In Lemma 7, given a partition $P \in \mathcal{P}^W_{\mathcal{R}}(K)$ and a P-renaming τ, we define a function κ^τ_P such that two terms are equal (under P) if and only if they have the same image under κ^τ_P. Lemma 8 shows that the function κ^τ_P is the same as the function τ^* computed by Algorithm 10. Thus, given two terms t and t', we have $t \approx_P t'$ if and only if $\tau^*(t) = \tau^*(t')$, and this can be decided by using Algorithm 10 to compute the images of t and t' under τ^*.

We assume that K is a finite set of terms, $P \in \mathcal{P}^W_{\mathcal{R}}(K)$, $\prec$ is a subterm-compatible order, and $\mathcal{N}_p$ is an infinite set of names disjoint from Σ (and thus also from $\mathcal{N}$).

We define

$$\kappa^\tau_P \colon T_\Sigma(\mathcal{N}_p) \to T_\Sigma(\mathcal{N}_p)$$

inductively as follows:

- if there is $p \in P^*$ and $t' \in p$ such that $t' \approx_{P^*} t$, then $\kappa^\tau_P(t) = \tau(p)$;
- otherwise, we define $\kappa^\tau_P(f(t_1, \ldots, t_n)) = f(\kappa^\tau_P(t_1), \ldots, \kappa^\tau_P(t_1))\!\downarrow$.

We note that, for all $t \in T_\Sigma(\mathcal{N}_p)$,

$$\kappa_P^\tau(\kappa_P^\tau(t)) = \kappa_P^\tau(t) \qquad \text{and} \qquad \kappa_P^\tau(t) \approx_{P^*} t.$$

We can drop both τ and P when they are clear from context.

Lemma 7. *For all $t, t' \in T_\Sigma$, all finite sets of terms K, all $P \in \mathcal{P}_\mathcal{R}^W(K)$, and all P-renamings τ, we have $t \approx_P t'$ if and only if $\kappa_P^\tau(t) = \kappa_P^\tau(t')$.*

Proof. Let $\sim$ be the relation on terms given by $t \sim t'$ if and only if $\kappa(t) = \kappa(t')$. Since $t, t' \in T_\Sigma$, we have $t \approx_P t'$ if and only if $t \approx_{P^*} t'$; therefore, it is sufficient to prove that $\sim = \approx_{P^*}$. Because $t \approx_{P^*} \kappa(t)$ for all t, it is clear that $\sim \subseteq \approx_{P^*}$.

It remains to prove that $\approx_{P^*} \subseteq \sim$. By definition of $\approx_{P^*}$, it is sufficient to show:

(1) $\sim$ is a congruence relation;

(2) $\approx_\mathcal{R} \subseteq \sim$;

(3) if $p \in P$ and $t, t' \in p$, then $t \sim t'$.

(1) It is clear that $\sim$ is reflexive, symmetric and transitive. Suppose that

$$t_1 \sim t_1', \ldots, t_n \sim t_n', f \in \Sigma_n,$$

and let

$$t = f(t_1, \ldots, t_n), \qquad t' = f(t_1', \ldots, t_n').$$

If there is $p \in P$ and $t'' \in p$ such that $t \approx_P t''$, then, because $\sim \subseteq \approx_{P^*}$, we have $t' \approx_{P^*} t \approx_{P^*} t''$; and because $t, t', t'' \in T_\Sigma$, we have $t' \approx_P t''$. Thus, $\kappa(t) = \tau(p) = \kappa(t')$, and $t \sim t'$. Otherwise, we have

$$\begin{aligned} \kappa(t) &= f(\kappa(t_1), \ldots, \kappa(t_n))\downarrow \\ &= f(\kappa(t_1'), \ldots, \kappa(t_n'))\downarrow \\ &= \kappa(t'). \end{aligned}$$

Thus, $\sim$ is a congruence relation.

(2) In light of (1), it is sufficient to show that $t \sim t\downarrow$ for all t. Consider the *pattern term* $t_{pt} \in T_\Sigma(\mathcal{X})$ obtained from t as follows: at each position $p \in pos(t)$ such that there is $p' \in P$ and $t' \in p'$ satisfying $t\,|_p \approx_P t'$, we replace $t\,|_p$ by a fresh variable. Let V be the set of variables occurring in t_{pt}.

Let $\sigma_{pt} \colon V \to T_\Sigma$ be the substitution such that $t_{pt}\sigma_{pt} = t$, and let $\sigma_\tau \colon V \to ran(\tau)$ be given by $x\sigma_\tau = \tau(p')$, where p' is the (unique) $p' \in P$ such that there is $t' \in p'$ satisfying $x\sigma_{pt} \approx_P t'$. We have

$$\kappa(t) = (t_{pt}\sigma_\tau)\downarrow .$$

Now, because $x\sigma_{pt} \in T_\Sigma^W$ for all $x \in V$ and function symbols in Σ^W do not occur in $\mathcal{R}$, we also have

$$t\downarrow = t_{pt}\downarrow (\sigma_{pt}\downarrow). \tag{5.1}$$

For all $x \in V$, there is $p' \in P$ and $t' \in p'$ such that $x\sigma_{pt} \approx_P t'$. Because $\approx_\mathcal{R} \subseteq \approx_P$, we also have $x(\sigma_{pt}\downarrow) \approx_P t'$, and thus $\kappa(x(\sigma_{pt}\downarrow)) = x\sigma_\tau$. We conclude that

$$\kappa(t\downarrow) = (t_{pt}\downarrow \sigma_\tau)\downarrow . \tag{5.2}$$

The convergence of $\mathcal{R}$ implies that

$$(t_{pt}\downarrow \sigma_\tau)\downarrow = (t_{pt}\sigma_\tau)\downarrow;$$

combining this fact with equations (5.1) and (5.2), we obtain

$$\kappa(t) = (t_{pt}\sigma_\tau)\downarrow = (t_{pt}\downarrow \sigma_\tau)\downarrow = \kappa(t\downarrow),$$

as desired.

(3) Whenever $p \in P$ and $t \in p$, we have $\kappa(t) = \tau(p)$. Property (3) follows.

□

The following Algorithm defines another function τ^*. Lemma 8 below shows that the theoretical definition of κ is equivalent to this operational definition of τ^*.

Algorithm 10 (P, R)-renaming algorithm

`Input:` $K, P \in \mathcal{P}_\mathcal{R}^W(K)$, a subterm-compatible order $\prec$, and a P-renaming τ
`Output:` functions $\tau^+, \tau^* \colon sub[K] \to T_\Sigma(\mathcal{N}_p)$

1: $\tau^+, \tau^* \leftarrow \emptyset$
2: let $t_1, \ldots, t_k$ be such that $t_1 \prec \ldots \prec t_k$ and $sub[K] = \{t_1, \ldots, t_k\}$
3: `for` i `from` 1 `to` k
4: let $t_i = f(t'_1, \ldots, t'_n)$
5: $\tau^+(t_i) \leftarrow f(\tau^*(t'_1), \ldots, \tau^*(t'_n))\downarrow$
6: `if` there is $j \leq i$ s.t. $\tau^+(t_j) = \tau^+(t_i)$ and $t_j \in p$ for some $p \in P$
7: $\tau^*(t_i) \leftarrow \tau(p)$
8: `else` $\tau^*(t_i) \leftarrow \tau^+(t_i)$
9: `return` τ^+, τ^*

We now show that τ^* coincides with κ_P^τ for all terms in $sub[K]$. Since the function κ can be used to decide $\approx_P$ by Lemma 7, it follows that $\approx_P$ by using Algorithm 10.

Lemma 8. *Consider the function τ^* output by Algorithm 10 on input $(K, P, \prec, \tau)$. For all $t \in sub[K]$, we have $\tau^*(t) = \kappa_P^\tau(t)$.*

Proof. The proof is by induction on $|sub[K]|$. The result is clear if $|sub[K]| = 0$. Suppose then that $|sub[K]| = n + 1$. Let $t_1, \ldots, t_{n+1}$ be such that

$$sub[K] = \{t_1, \ldots, t_{n+1}\} \qquad \text{and} \qquad t_1 \prec \ldots \prec t_{n+1}.$$

Our induction hypothesis is that, for all $i \in \{1, \ldots, n\}$, we have $\tau^*(t_i) = \kappa(t_i)$. For each $i \in \{1, \ldots, n+1\}$, let

$$t_i = f_i(t_{i,1}, \ldots, t_{i,k_i}).$$

If there is $j \leq n + 1$ and $p \in P$ such that $t_j \in p$ and $\tau^+(t_j) = \tau^+(t_{n+1})$, the induction hypothesis implies that

$$\tau^+(t_j) = f_j(\tau^*(t_{j,1}), \ldots, \tau^*(t_{j,k_j}))\downarrow \approx_{P^*} t_j,$$

and, similarly,

$$\tau^+(t_{n+1}) = f_{n+1}(\tau^*(t_{n+1,1}), \ldots, \tau^*(t_{n+1,k_{n+1}}))\downarrow \approx_{P^*} t_{n+1}.$$

This implies that $t_j \approx_{P^*} t_{n+1}$, and because $t_j, t_{n+1} \in T_\Sigma$, we also have $t_j \approx_P t_{n+1}$. Thus, we have $\kappa(t_{n+1}) = \tau^*(t_{n+1}) = \tau(p)$.

Suppose then that there is no $p \in P$ and $t \in p$ such that $t_{n+1} \approx_P t$. The reasoning above implies that there is no $j \leq n + 1$ and $p \in P$ such that $t_j \in p$ and $\tau^+(t_j) = \tau^+(t_{n+1})$. Thus,

$$\begin{aligned} \tau^*(t_{n+1}) &= \tau^+(t_{n+1}) \\ &= f_{n+1}(\tau^*(t_{n+1,1}), \ldots, \tau^*(t_{n+1,k_{n+1}}))\downarrow \\ &= f_{n+1}(\kappa(t_{n+1,1}), \ldots, \kappa(t_{n+1,k_{n+1}}))\downarrow \\ &= \kappa(t_{n+1}), \end{aligned}$$

using the induction hypothesis.

It remains to consider the case that there is $p \in P$ and $t \in p$ such that $t_{n+1} \approx_P t$. In this case, we have $\kappa(t_{n+1}) = \tau(p)$; therefore, we have to prove that $\tau^*(t_{n+1}) = \tau(p)$ as well. Because $t \in sub[K]$, we have $t = t_j$ for some $j \leq n + 1$. The induction hypothesis yields

$$\begin{aligned} \tau^+(t_{n+1}) &= f_{n+1}(\tau^*(t_{n+1,1}), \ldots, \tau^*(t_{n+1,k_{n+1}}))\downarrow \\ &= f_{n+1}(\kappa(t_{n+1,1}), \ldots, \kappa(t_{n+1,k_{n+1}}))\downarrow . \end{aligned}$$

Using again the induction hypothesis and noting that $\kappa(t)$ is in normal form for all t and $f_j \in \Sigma^W$ does not occur in $\mathcal{R}$, we have

$$\begin{aligned} \tau^+(t_j) &= f_j(\tau^*(t_{j,1}), \ldots, \tau^*(t_{j,k_j}))\downarrow \\ &= f_j(\kappa(t_{j,1}), \ldots, \kappa(t_{j,k_j}))\downarrow \\ &= f_j(\kappa(t_{j,1}), \ldots, \kappa(t_{j,k_j})). \end{aligned}$$

Combining the fact that $\tau^+(t) \approx_{P^*} t$ for all t with the two equalities above, it follows that

$$f_{n+1}(\kappa(t_{n+1,1}), \ldots, \kappa(t_{n+1,k_{n+1}}))\downarrow \approx_{P^*} f_j(\kappa(t_{j,1}), \ldots, \kappa(t_{j,k_j})).$$

Furthermore, $\kappa(t)$ is always in normal form: therefore, we have either

$$f_{n+1}(\kappa(t_{n+1,1}),\ldots,\kappa(t_{n+1,k_{n+1}}))\downarrow = f_{n+1}(\kappa(t_{n+1,1}),\ldots,\kappa(t_{n+1,k_{n+1}}))$$

or

$$f_{n+1}(\kappa(t_{n+1,1}),\ldots,\kappa(t_{n+1,k_{n+1}}))\downarrow \in sub(f_{n+1}(\kappa(t_{n+1,1}),\ldots,\kappa(t_{n+1,k_{n+1}}))).$$

In the first case, if $f_{n+1} \notin \Sigma^W$, then

$$f_{n+1}(\kappa(t_{n+1,1}),\ldots,\kappa(t_{n+1,k_{n+1}})) \not\approx_{P^*} t_j,$$

since $head(t_j) \in \Sigma^W$; it follows that $t_{n+1} \not\approx_{P^*} t_j$, a contradiction. Therefore, we must have $f_{n+1} \in \Sigma^W$. It follows that $t_{n+1} \in T_\Sigma^W$, and because P is $\approx_R$-closed, we have $t_{n+1} \in p$ and $\tau^*(t_{n+1}) = \tau(p)$.

In the second case, we note that, whenever $s \in sub(\kappa(t))$, we have $\kappa(s) = s$: Therefore, we have

$$f_{n+1}(\kappa(t_{n+1,1}),\ldots,\kappa(t_{n+1,k_{n+1}}))\downarrow = \kappa(f_{n+1}(\kappa(t_{n+1,1}),\ldots,\kappa(t_{n+1,k_{n+1}}))\downarrow),$$

and because $t_{n+1} \approx_P t_j$ and $t_j \in p$, we have

$$\tau^+(t_{n+1}) = \kappa(f_{n+1}(\kappa(t_{n+1,1}),\ldots,\kappa(t_{n+1,k_{n+1}}))\downarrow) = \tau(p),$$

as desired.

□

Selection functions. A *selection function for* K is a function

$$\iota\colon sub[K] \to PS \cup \{\bot\}$$

such that, for each $t \in sub[K]$, either $\iota(t) = \bot$ or $head(\iota(t)) = head(t)$.

Given $\omega \in \mathbf{\Omega}$, we say that ω *satisfies* ι if, for all $t = f(t_1,\ldots,t_n) \in sub[K]$, either

$$(\omega(t_1),\ldots,\omega(t_n)) \in [\![dom(\iota(t))]\!] \qquad \text{and} \qquad \omega(t) \in [\![ran(\iota(t))]\!]$$

or

$$(\omega(t_1),\ldots,\omega(t_n)) \notin dom_{\mathcal{S}}(f) \qquad \text{and} \qquad \iota(t) = \omega(t) = \bot.$$

We denote by $I(K)$ the set of selection functions for K, and by $I_{\mathcal{S}}(K) \subseteq I(K)$ the set of selection functions ι for K such that there is $\omega \in \mathbf{\Omega}$ that satisfies ι. The following Lemma, proved in the Appendix, shows that, given a finite set of terms K, $I_{\mathcal{S}}(K)$ is a finite and computable set.

Lemma 9. *For all valid setups $\mathcal{S}$ and all finite sets of terms K, $I_{\mathcal{S}}(K)$ is a finite and computable set.*

A selection function for a finite set K determines which property statement applies to each term in $sub[K]$. Note that, if $\omega \in \mathbf{\Omega}$ is an assignment satisfying PS and K is a finite set of terms, there exists exactly one selection function $\iota \in I(K)$ satisfied by ω: it is the function ι that associates each term $f(t_1, \ldots, t_n)$ to the unique property statement

$$ps \in PS_f \quad \text{such that} \quad (\omega(t_1), \ldots, \omega(t_n)) \in [\![dom(ps)]\!],$$

or $\perp$ if no such ps exists.

Compatibility. The compatibility condition is that, if K is a finite set of terms, $t \in sub[K]$, $P \in \mathcal{P}^W_{\mathcal{R}}(K)$, $\iota \in I_{\mathcal{S}}(K)$, and $\iota(t) \neq \perp$, then there is $t' \in sub(t)$ such that:

- $t \approx_{P|_{psub(t)}} t'$;
- for all $t'' \in sub[K]$ such that $t \approx_{P|_{psub(t)}} t''$, we have either

$$\iota(t'') = \perp \qquad \text{or} \qquad [\![ran(\iota(t'))]\!] \subseteq [\![ran(\iota(t''))]\!].$$

Intuitively, the compatibility condition requires the equational theory $\approx_{\mathcal{R}}$ and the property statements in PS to be compatible. It is a basic requirement that should be satisfied by any meaningful setup specification. The following example illustrates this.

Example 23 (Incompatibility between $\approx_{\mathcal{R}}$ and PS)**.** Consider a rewriting system $\mathcal{R}$ containing the symmetric decryption rewrite rule

$$\left\{\left|\{|x|\}_y\right|\right\}^{-1}_y \to x$$

and the following property statements:

- $\{|T_{\mathcal{B}^{256}}|\}^{-1}_{T_{\mathcal{B}^{256}}} \subseteq T_{\mathcal{B}^{128}}$;
- $\{|T_{\mathcal{B}^{256}}|\}_{T_{\mathcal{B}^{256}}} \subseteq T_{\mathcal{B}^{256}}$.

Let

$$t' = \{|\{|t|\}_k|\}^{-1}_k,$$

where $t, k \in \Sigma_0$ and $type(t) = type(k) = T_{\mathcal{B}^{256}}$.

In this case, we have

$$\iota(t) = T_{\mathcal{B}^{256}} \Rightarrow [\![ran(\iota(t))]\!] = T_{\mathcal{B}^{256}}$$

and

$$\iota(t') = T_{\mathcal{B}^{128}} \Rightarrow [\![ran(\iota(t'))]\!] = T_{\mathcal{B}^{128}}$$

for all selection functions $\iota \in I_{\mathcal{S}}(\{t, t'\})$.

Because $t \approx_{\mathcal{R}} t'$ and $\mathcal{B}^{128} \cap \mathcal{B}^{256} = \emptyset$, it follows that there is no $\omega \in \mathbf{\Omega}$ that satisfies $\approx_{\mathcal{R}}$ and PS.

Note however that, having

$$\{|T_{\mathcal{B}^{256}}|\}^{-1}_{T_{\mathcal{B}^{256}}} \subseteq T_{\mathcal{B}^{256}} \qquad \text{instead of} \qquad \{|T_{\mathcal{B}^{256}}|\}^{-1}_{T_{\mathcal{B}^{256}}} \subseteq T_{\mathcal{B}^{128}},$$

we could have $[\![type(t)]\!] = B$ for any non-empty set $B \subseteq \mathcal{B}^{256}$ without violating our compatibility condition.

Example 24. With the choice of Σ^W described in Example 21, our running example (described in Examples 17, 18 and 19) satisfies the disjointness and compatibility conditions.

Probability measure. We show that, under the disjointness and compatibility hypotheses, the revised sampling algorithm yields a probability measure μ_{ROM} satisfying $\mathcal{S}$. For each total subterm-compatible order $\prec$, each $\lambda \in \Lambda$, and each finite set of terms K such that $dom(\lambda) \subseteq K$, let $\mu^{K,\prec}(\lambda)$ be the probability that executing the sampling the revised version of Algorithm 8 on input $dom(\lambda)$ using the order $\prec$ yields a function $\psi_{ROM}\colon sub[K] \to \mathcal{B}^*_\perp$ such that $\psi_{ROM}(t) \in \lambda(t)$ for all $t \in K$.

Theorem 10 shows that the problem in Example 20 does not occur anymore: the probability distribution of terms output by Algorithm 9 does not depend on the input set of terms or on the (subterm-compatible) order chosen.

Theorem 10. *Suppose that the disjointness and compatibility conditions are satisfied for the subterm-compatible orders $\prec$ and $\prec'$. Then, if $\lambda, \lambda' \in \Lambda$ are such that $\Omega_\lambda = \Omega_{\lambda'}$, we have $\mu^{\prec}(\lambda) = \mu^{\prec'}(\lambda')$.*

In light of Theorem 10, we define the function $\mu_{ROM}\colon \Omega_\Lambda \to [0, 1]$ for each $\lambda \in \Lambda$ as $\mu_{ROM}(\Omega_\lambda) = \mu^{\prec}(\lambda)$ for any subterm-compatible order $\prec$.

Theorem 11 now yields our desired probability distribution.

Theorem 11. *There exists a unique extension of μ_{ROM} to $\mathcal{F}$ that is a probability measure. Using the same symbol μ_{ROM} to refer to this extension, we have $\mu_{ROM}(\Omega_{\mathcal{S}}) = 1$. Hence, μ_{ROM} is a model of $\mathcal{S}$.*

We adopt the abuse of notation used in the above theorem and use the same symbol μ_{ROM} to refer to the unique extension of μ_{ROM} to $\mathcal{F}$ that is a probability measure. Section 5.3 provides an alternative, algebraic definition of μ_{ROM} that is well-suited for computing probabilities.

5.2.3 Comparing the two probability measures

We describe the relationship between the probability measures $\mu^{K,\prec}$ described in Theorem 9 and the probability measure μ_{ROM} described in Theorem 11.

For each $f \in \Sigma$, we define

$$L_f = \min_{ps \in PS_f} |\, [\![ran(ps)]\!] \,| \qquad \text{and} \qquad L = \min_{f \in \Sigma \setminus \Sigma^W} L_f.$$

Note that, if we assume that non-weak terms are always sampled from "large" sets of bitstrings whenever they are defined, then L is large as well. Intuitively, Theorem 12 shows that, if this is the case, the different probability measures we have described coincide except on a set whose probability is "small". More precisely, the two probability measures coincide except on a set whose probability is $\mathcal{O}(1/L)$.

Theorem 12. *For any finite set of terms K, there exists a set $\Omega(K)$ such that, for any subterm-compatible order $\prec$:*

(1) for any $\lambda \in \Lambda_K$, $\mu^{K,\prec}(\Omega_\lambda \cap \Omega(K)) = \mu_{ROM}(\Omega_\lambda \cap \Omega(K))$;

(2) $\mu^{K,\prec}(\boldsymbol{\Omega} \setminus \Omega(K)) = \mu_{ROM}(\boldsymbol{\Omega} \setminus \Omega(K)) \leq |K|^2 \cdot |I_{\mathcal{S}}(K)| \cdot (1/L)$.

Note that the statement of Theorem 12 is stronger than merely bounding the difference in the probability of sets in Ω_Λ. For example, Theorem 12 implies that the probability of two terms being sampled to the same bitstring as measured by the two different probability measures also differs by at most $|K|^2 \cdot |I_{\mathcal{S}}(K)| \cdot (1/L)$.

Asymptotic interpretation. A function $f\colon \mathbb{N} \to \mathbb{R}$ is *negligible* if it decreases faster than the inverse of any polynomial. More formally, f is negligible if

$$\forall_{k \in \mathbb{N}}.\ \exists_{\eta_k \in \mathbb{N}}.\ \forall_{\eta \in \mathbb{N}}.\ \left(\eta > \eta_k \Rightarrow f(\eta) < \eta^{-k}\right).$$

Suppose that, for each $\eta \in \mathbb{N}$, $[\![\cdot]\!]_\eta$ is a type interpretation function and $\mathcal{S}_\eta = \langle \langle \Sigma, \approx_R, \mathcal{T}, PS \rangle, [\![\cdot]\!]_\eta \rangle$ is a setup specification which satisfies the disjointness and compatibility conditions. For each $f \in \Sigma$ and each $\eta \in \mathbb{N}$, define

$$L_{f,\eta} = \min_{ps \in PS_f} [\![ran(ps)]\!]_\eta$$

and

$$L_\eta = \min_{f \in \Sigma \setminus \Sigma^W} L_{f,\eta},$$

and suppose that $1/L_\eta$ is negligible as a function of η.

Note that this condition is equivalent to requiring, for each function symbol $f \in \Sigma \setminus \Sigma^W$ and each $ps \in PS_f$, that $1/|\, [\![ran(ps)]\!]_\eta \,|$ is negligible as a function of η. Intuitively, this condition requires that non-weak terms, when defined, are always mapped to bitstrings sampled from large enough sets. Specifically, the sizes of the sets from which outputs of f are sampled should grow faster than any polynomial as a function of the parameter η, i.e.,

$$\forall_{k \in \mathbb{N}}.\ \exists \eta_{k,f} \in \mathbb{N}.\ \forall_{f \in \Sigma \setminus \Sigma^W}.\ \forall_{ps \in PS_f}.\ \forall_{\eta \in \mathbb{N}}.\ \left(\eta > \eta_{k,f} \Rightarrow |\, [\![ran(ps)]\!]_\eta \,| > \eta^k\right).$$

Let $\mu_\eta^{K,\prec}$ be the probability measure given by Theorem 9 when Algorithm 8 is executed using the interpretation function $[\![\cdot]\!]_\eta$. Similarly, let $\mu_{ROM,\eta}$ be the probability measure given by Theorem 11 when Algorithm 9 is executed using the interpretation function $[\![\cdot]\!]_\eta$. Then, the following is a corollary of Theorem 12.

Corollary 1. *Let K be a finite set of terms, and suppose that $|I_{\mathcal{S}_\eta}(K)|$ grows polynomially as a function of η. For any finite set of terms K, there exists a set $\Omega(K)$ such that, for any subterm-compatible order $\prec$:*

(1) for any $\lambda \in \Lambda_K$, $\mu_\eta^{K,\prec}(\Omega_\lambda \cap \Omega(K)) = \mu_{ROM,\eta}(\Omega_\lambda \cap \Omega(K))$;

(2) $\mu_\eta^{K,\prec}(\mathbf{\Omega} \setminus \Omega(K)) = \mu_{ROM,\eta}(\mathbf{\Omega} \setminus \Omega(K))$, and both quantities are negligible as functions of η.

5.3 Computing Probabilities

In this section we show that the probability measure μ_{ROM} can be equivalently defined algebraically. This algebraic definition reduces the problem of computing probabilities of the form

$$P_{\mu_{ROM}}[t_1 \in B_1, \ldots, t_n \in B_n, t'_1 = t''_1, \ldots, t'_{n'} = t''_{n'}] \tag{5.3}$$

(where $B_1, \ldots, B_n$ are sets of bitstrings) to computing the sizes of intersections of sets in $\{B_1, \ldots, B_n\} \cup \mathcal{T}$. A full specification of the interpretations of types is not necessary.

Throughout this section we assume that K is a finite set of terms.

Types error **and** empty**.** We define additional types error and empty, and extend $[\![\cdot]\!]$ such that $[\![\mathsf{error}]\!] = \{\bot\}$ and $[\![\mathsf{empty}]\!] = \emptyset$. Furthermore, when $X \subseteq \mathcal{B}^*$, we may use a type $T_X \notin \mathcal{T}$, and further extend $[\![\cdot]\!]$ such that $[\![T_X]\!] = X$. Note that we still refer to $[\![\cdot]\!]$ as an interpretation function. Let $\iota \in I(K)$ be a selection function for K.

Supremum support. We define the *supremum ι-support* function $\overline{supp_\iota}\colon K \to \mathcal{T} \cup \{\mathsf{error}\}$ by error if $\iota(t) = \bot$ and $\overline{supp_\iota}(t) = ran(\iota(t))$ otherwise.

$\approx_P^+$ and $\approx_P^*$-equivalence classes. If $t \in sub[K]$, we define

$$[t]_P^* = \left\{(\tau^*)^{-1}(t) \mid t \in \tau^*[sub[K]] \setminus \mathcal{N}^+\right\}$$

to be the τ^*-equivalence class of t. Analogously,

$$[t]_P^+ = \left\{(\tau^+)^{-1}(t) \mid t \in \tau^+[sub[K]]\right\}$$

is the τ^*-equivalence class of t, where τ^* is as computed by Algorithm 10. If K is a set of terms, we define:

- $[K]^*_P$ is the set of τ^*-equivalence classes of terms in K;
- $[K]^+_P$ is the set of τ^+-equivalence classes of terms in K;
- $\approx^+_P$ is the equivalence relation such that $t \approx^+_P t'$ iff $\tau^+(t) = \tau^+(t')$;
- $\approx^*_P$ is the equivalence relation such that $t \approx^+_P$ iff $t \approx^*_P t'$;

It is clear that $\approx_P \subseteq \approx^*_P$. Thus, for each $\approx^*_P$-equivalence class C, there is a $\approx^+_P$-equivalence class C' such that, for all $t \in C$, $[t]^*_P = C'$; for each $C \in [K]^+_P$, we denote by $[C]^*_P$ this unique class C'.

Let $\prec$ be some subterm-compatible order, $\iota \in I(K)$, and $C \in [sub[K]]^+_P$, and suppose that t is the least element of C (with respect to $\prec$) such that $\iota(t) \neq \bot$. We define

$$\overline{supp_\iota}(C) = ran(\iota(t)).$$

Lemma 22, proved in Appendix C, shows that $\overline{supp_\iota}$ is well-defined and, for each $C \in [sub[K]]^+_P$ and each $t \in C$, either $\overline{supp_\iota}(t) = \bot$ or $\overline{supp_\iota}(C) \subseteq \overline{supp_\iota}(t)$. Note also that, if there is $a \in \Sigma_0$ such that $a \in C$, then $\overline{supp_\iota}(C) \neq \mathsf{error}$.

Intuitively, the supremum support of each equivalence class $C \in [sub[K]]^+_P$ is the set of bitstrings from which the bitstring corresponding to the least term $t \in C$ is sampled during the execution of Algorithm 9, assuming that all the sampling of all the terms sampled before t satisfies the partition P and the selection function ι.

Infimum support. Given $\lambda \in \Lambda_K$ and $\iota \in I(K)$, we define $\mathcal{P}^W_{\mathcal{R},\bot}(\iota)$ as the set of partitions P of $sub[K] \cap T^W$ for which there is $p_\bot \in P$ such that, for all $t \in sub[K] \cap T^W$, $t \in p_\bot$ if and only if $\iota(t) = \bot$. If $P \in \mathcal{P}^W_{\mathcal{R},\bot}(\iota)$, we define the infimum (ψ, ι, P)-support function for each $t \in sub[K]$ as follows: For each $t \in sub[K]$, $supp_{\lambda,\iota,P}(t)$ is the smallest set such that:

- $\overline{supp_\iota}(t) \subseteq supp_{\lambda,\iota,P}(t)$;
- if $t \in K$, then $T_{\lambda(t)} \in supp_{\lambda,\iota,P}(t)$;
- if $t \approx_P t'$, $T \in supp_{\lambda,\iota,P}(t')$, and $\mathsf{error} \notin \{\overline{supp_\iota}(t), \overline{supp_\iota}(t')\}$, then $T \in supp_{\lambda,\iota,P}(t)$.

For each class $C \in [K]^*_P$ there is a (finite) set $\mathbf{T}$ such that, for each $t \in C$, we have $supp_{\lambda,\iota,P}(t) = \mathbf{T}$ or $\overline{supp_\iota}(t) = \mathsf{error}$. We define $supp_{\lambda,\iota,P}(C)$ for each class $C \in [K]^*_P$ by

- $supp_{\lambda,\iota,P}(C) = \mathsf{empty}$ if there is $t \in C$ such that $\overline{supp_\iota}(t) = \mathsf{error}$ and there is $T \in supp_{\lambda,\iota,P}(t)$ such that $\bot \notin [\![T]\!]$,
- and $supp_{\lambda,\iota,P}(C) = \mathbf{T}$ (where $\mathbf{T}$ is as above) otherwise.

Note that $supp_{\lambda,\iota,P}(C) = \mathsf{empty}$ if and only if there is $t \in C$ such that $\overline{supp_\iota}(t) = \mathsf{error}$ and there is $T \in supp_{\lambda,\iota,P}(t)$ such that either $T \in \mathcal{T}$ or $T = T_B$ for some $B \subseteq \mathcal{B}^*$ which does not contain $\bot$. Furthermore, $supp_{\lambda,\iota,P}(C) = \mathsf{error}$ if and only if $supp_{\lambda,\iota,P}(t) = \mathsf{error}$ for all $t \in C \cap K$.

Intuitively, a partition P and a selection function ι impose several restrictions on the bitstrings that each term $t \in sub[K]$ may be mapped to: namely, if there exists $t' \in sub[K]$ such that $t \approx_P t'$, then t' maps to either $\bot$ or to a bitstring in $[\![ran(\iota(t'))]\!]$. Simiilarly, if there exists $f(t_1, \ldots, t_n) \in sub[K]$ such that $t \approx_P t_i$ for some i, and

$$dom(\iota(f(t_1, \ldots, t_n))) = T_1 \times \ldots \times T_n,$$

then t' maps to either $\bot$ (if $ran(\iota(t)) = \mathsf{error}$) or to a bistring in $[\![T_i]\!]$.

In contrast with the supremum support of a class $C \in [sub[K]]^+_P$, the infimum support of a class $C \in [K]^*_P$ is the intersection of all the sets of bitstrings which the mapping of terms in C must belong to given the partition P and the selection function ι, according to the reasoning above. In other words, it is the largest set of bitstrings that terms in C may map to while still satisfying P and ι.

The idea is that, given a partition P and a selection function ι, a sampling of the terms in $sub[K]$ satisfies P and ι if and only if, for all the terms t such that $ran(\iota(t)) \neq \mathsf{error}$ and t is the minimal element of its equivalence class $[t]^+_P$ (with respect to some subterm-compatible order), t is sampled to a bitstring in $supp_{\lambda,\iota,P}([t]^*_P)$. Note that only these terms t are actually sampled by Algorithm 9, since all the others are mapped either to $\bot$ or to the same bitstring that t was mapped to. Moreover, observe that t is mapped to a bitstring in $\overline{supp_\iota}([t]^+_P)$. This allows us to compute probabilities in our model in a much more efficient algebraic manner, as formalized in the next paragraph.

Function sets. Suppose that $\lambda \in \Lambda_K$, $\iota \in I(K)$, and $P \in \mathcal{P}^W_{\mathcal{R},\bot}(K)$.

We define the following sets:

- $\Psi^D(\lambda, \iota, P)$ is the set of functions $\psi \colon [sub[K]]^+_P \to \mathcal{B}^*_\bot$ such that

$$C \in [sub[K]]^+_P \quad \Rightarrow \quad \psi(C) \in [\![\overline{supp_\iota}(C)]\!]\,;$$

- $\Psi^U(\lambda, \iota, P)$ is the set of functions $\psi \colon [sub[K]]^*_P \to \mathcal{B}^*_\bot$ such that

$$C \in [sub[K]]^*_P \quad \Rightarrow \quad \psi(C) \in supp_{\lambda,\iota,P}(C)$$

and

$$\big(C, C' \in [sub[K] \cap T^W]^*_P \wedge C \neq C'\big) \quad \Rightarrow \quad \psi(C) \neq \psi(C').$$

Finally, if $\psi \in \Psi_K$, we define the sets:

- $\Psi^D(\psi, \iota, P) = \Psi^D(\lambda(\psi), \iota, P)$,

- and $\Psi^U(\psi, \iota, P) = \Psi^U(\lambda(\psi), \iota, P)$,

where $\lambda(\psi) \in \Lambda_K$ is defined by $\lambda(\psi)(t) = \{\psi(t)\}$ for all $t \in K$.

Definition 7. *We define the function* $\mu_C \colon \Lambda \to [0,1]$ *for each* $\lambda \in \Lambda$ *by*

$$\mu_C(\lambda) = \sum_{\iota \in I(K)} \left(\sum_{P \in \mathcal{P}^W_{\mathcal{R},\perp}(K)} \frac{|\Psi^U(\lambda, \iota, P)|}{|\Psi^D(\lambda, \iota, P)|} \right),$$

where $K = dom(\lambda)$.

Theorems 13 and 14 establish that μ_C is a well-defined probability distribution in $\mathcal{F}$ which coincides with the probability distribution μ_{ROM} defined in the previous chapter. Their proofs can be found in Appendix C, Section C.2.

Theorem 13. *If* K *is a finite set of terms and* $\lambda, \lambda' \in \Lambda_K$ *are such that* $\Omega_\lambda = \Omega_{\lambda'}$, *then* $\mu_C(\lambda) \in [0,1]$ *and* $\mu_C(\lambda) = \mu_C(\lambda')$.

In light of Theorem 13, we use the symbol μ_C for the function $\mu_C \colon \Omega_\Lambda \to [0,1]$ defined, for each $\lambda \in \Lambda$, by $\mu_C(\Omega_\lambda) = \mu_C(\lambda)$.

Theorem 14. *There exists a unique extension of* μ_C *to* $\mathcal{F}$ *that is a probability measure. Abusing notation and using the symbol* μ_C *to refer to this extension, we have* $\mu_C = \mu_{ROM}$.

Implementation. An implementation of our algorithm for computing probabilities is available in [2]. It can be used to compute probabilities of the form (5.3) for the cryptographic primitives and respective properties considered in our running example. The user must, however, specify the sizes of intersections of the sets of bitstrings $B_1, \ldots, B_n$ with the specified property types.

Let $T = \{t_1, \ldots, t_n, t'_1, t''_1, \ldots, t'_{n'}, t'_{n'}\}$. Because we must consider $\approx_R$-closed partitions of the set $T^W_\Sigma \cap sub[T]$ of weak subterms of T, the complexity of the computation is exponential in the number of such weak subterms. However, for the setup specification considered in our running example, if T contains no subterms of the form $\pi_i(t)$ for some $i \in \{1,2\}$ and some term t such that $head(t) \neq \langle \cdot, \cdot \rangle$, the complexity is linear in the number of non-weak subterms of T. In these cases the tool calculates probabilities with up to ten weak subterms in under one second in a standard laptop.

5.4 Off-line Guessing Examples

We now present a few examples of off-line guessing attacks. These examples illustrate that such attacks can result from implementation details that, while often trivial, are outside the scope of traditional symbolic methods. We show how such details can be modeled in our framework and used to estimate the probability of

attacks. All probability calculations in this section rely on the setup specification described in our running example and are performed automatically by our implementation.

Attacker model. Recall that the algebra of terms we consider is now $T_{\Sigma\cup\mathcal{N}}$, with $ar(a) = 0$ for all $a \in \mathcal{N}$. Our definitions of frame and recipe must be slightly adapted in this setting: a frame is a pair $(\tilde{n}, \sigma)$, written $\nu\tilde{n}.\sigma$, where $\tilde{n}$ is a finite set of Σ_0 (representing a finite set of constant function symbols which may not be used in ϕ-recipes) and $\sigma\colon \mathcal{X} \to T_{\Sigma\cup\mathcal{N}}$. The set of ϕ-recipes is then $T(\phi) = T_{\Sigma\setminus\tilde{n}}(dom(\sigma))$.

Example 25 (Attack on a stored password hash)**.** This simple example considers an authentication server that stores password hashes instead of the users' passwords themselves. Let $s \in \Sigma_0$ be a weak password (i.e., $type(s) = \mathsf{pw}$). Suppose that an attacker obtains its hash $h(s)$ and wants to use it to off-line guess s. The attacker's knowledge is represented by $\phi = \nu\tilde{n}.\sigma = \nu\{s\}.\{x_1 \mapsto h(s)\}$.

To analyze off-line guessing in our framework, consider the frames $\phi_s = \nu\{s, w\}.\{x_1 \mapsto h(s), x \mapsto s\}$ and $\phi_w = \nu\{s, w\}.\{x_1 \mapsto h(s), x \mapsto w\}$. In order to verify his guess, an attacker tests whether hashing it with h yields $h(s) = x_1\phi$; that is, an attacker checks whether $h(x)\phi = x_1\phi$.

Recall that

$$[\![\mathsf{pw}]\!] \subseteq \mathcal{B}^{256} \qquad \text{and} \qquad (h[\mathcal{B}^{256}] \subseteq \mathcal{B}^{256}) \in PS.$$

Thus, the probability of each wrong guess passing the attacker's test is given by

$$P[h(x) = x_1] = 2^{-256}.$$

Since $|\,[\![\mathsf{pw}]\!]\,| \approx 2^{24}$, there are $2^{24} - 1$ wrong guesses to consider. Hence, the expected number of guesses w satisfying $h(w) = h(s)$ is

$$1 + \frac{2^{24} - 1}{2^{256}},$$

and we obtain an estimated probability of success of

$$\frac{1}{1 + \frac{2^{24}-1}{2^{256}}} \approx \frac{1}{1 + \frac{1}{2^{232}}}.$$

Example 26. The EKE (Encrypted Key Exchange) protocol is designed to allow two parties to exchange authenticated information using a weak symmetric key without allowing off-line guessing attacks. It is known that the redundancy of RSA public keys can be exploited to mount off-line guessing attacks on this protocol [40]. We show now how our methods can be used to estimate the success probability of this off-line guessing attack.

For representing these attacks, it is sufficient to consider the first step of the protocol. Let A and B be agents sharing a weak password $s \in \Sigma_0$, with $type(s) = \mathsf{pw}$.

For the first message, A randomly samples a bitstring from $[\![\mathsf{random}]\!]$, represented by a term $r \in \Sigma_0$ such that $type(r) = \mathsf{random}$. Afterwards, she uses it to compute an RSA public key $\langle \mathsf{mod}(r), \mathsf{expn}(r) \rangle$. Then, A (symmetrically) encrypts this public key with the shared password s and sends the encryption to B. To keep our analysis simple, we assume that the participants encrypt the modulus and the exponent separately and send them over the network as a pair of ciphertexts (instead of the encryption of the pair). Thus, this first message is represented by the term $\langle \{|\mathsf{mod}(r)|\}_s, \{|\mathsf{expn}(r)|\}_s \rangle$. See [40] for a full description of the protocol.

After observing this message in the network, the attacker's knowledge is described by the frame $\phi = \nu\tilde{n}.\sigma$, where $\sigma = \{x_1 \mapsto \langle \{|\mathsf{mod}(r)|\}_s, \{|\mathsf{expn}(r)|\}_s \rangle\}$ and $\tilde{n} = \{r\}$. The relevant frames for the analysis of off-line guessing attacks are $\phi_s = \nu\tilde{n}_w.\sigma_s$ and $\phi_w = \nu\tilde{n}_w.\sigma_w$, where $\tilde{n}_w = \tilde{n} \cup \{w\}$, $\sigma_s = \sigma \cup \{x_2 \mapsto s\}$, and $\sigma_w = \sigma \cup \{x_2 \mapsto w\}$.

While it may be infeasible to check whether the modulus is indeed the product of two large prime factors, an attacker can nevertheless use his guess w to decrypt the pair sent by A and test whether the resulting modulus has small prime factors and whether the exponent e is odd. The probability that each wrong guess satisfies these two properties is

$$P_\mu \left[\{|\widehat{\pi_1(x_1)}|\}_{x_2}^{-1} \in [\![\mathsf{nspf}]\!], \{|\widehat{\pi_2(x_1)}|\}_{x_2}^{-1} \in [\![\mathsf{odd}]\!] \right] \approx \frac{1}{48}.$$

Since there are $2^{24} - 1$ wrong guesses, we estimate the probability of success of this off-line guessing attack as described above to be

$$\frac{1}{1 + (2^{24} - 1)/48} \approx 2^{-18.5}.$$

Example 27. Consider now the same setup as in Example 26, except that only the exponent of the RSA public key is encrypted in the first message. The authors of EKE note [40] that the protocol is still vulnerable to off-line guessing attacks: Since the exponent of an RSA key is always odd, one can decrypt each encryption of a public key with each guess. For the right guess, decrypting each encryption will yield an odd exponent. The probability that a wrong guess achieves this decreases exponentially with the number of ciphertexts available to the attacker.

To formalize this in our setting, we let $\phi = \nu\tilde{n}.\sigma$ be the frame representing the attacker's knowledge, where

$$\sigma = \{x_i \mapsto \langle \mathsf{mod}(r_i), \{|\mathsf{expn}(r_i)|\}_s \rangle \mid i \in \{1, \ldots, n\}\}$$

and

$$\tilde{n} = \{r_1, \ldots, r_n, s\}.$$

The frames ϕ_s and ϕ_w used are as expected: $\phi_s = \nu\tilde{n}_w.\sigma_s$ and $\phi_w = \nu\tilde{n}_w.\sigma_w$, where $\tilde{n}_w = \tilde{n} \cup \{w\}$, $\sigma_s = \sigma \cup \{x_{n+1} \mapsto s\}$, and $\sigma_w = \sigma \cup \{x_{n+1} \mapsto w\}$.

In contrast with the previous example, using the redundancy of RSA modulus to validate or refute each guess is not a feasible strategy in this case. However, an

attacker may still test whether or not each of his guesses yields an odd value when used to decrypt the messages

$$\{|\mathsf{expn}(r_1)|\}_s, \ldots, \{|\mathsf{expn}(r_n)|\}_s.$$

The probability of each wrong guess satisfying this condition is

$$P\left[\{|\pi_2(x_1)|\}^{-1}_{x_{n+1}} \in [\![\mathsf{odd}]\!], \ldots, \{|\pi_2(x_1)|\}^{-1}_{x_{n+1}} \in [\![\mathsf{odd}]\!]\right] = \frac{1}{2^n}.$$

As in Example 26, we thus obtain $\frac{1}{1+\frac{2^{24}-1}{2^n}} = \frac{2^n}{2^n+2^{24}-1}$ as an estimate for the probability of success of this off-line guessing attack.

Chapter 6

Summary and Future Work

We discuss our contributions and future work. In Section 6.1 we discuss our work on equivalence properties, namely the decision procedures for static and trace equivalence presented in Chapters 3 and 4. Section 6.2 discusses our framework for symbolic probabilistic protocol analysis, presented in Chapter 5.

6.1 Equivalence Properties

We presented an efficient algorithm for deciding static equivalence under subterm convergent equational theories, as well as a general algorithm for deciding the trace equivalence of deterministic, bounded applied-pi calculus processes under equational theories generated by convergent rewriting systems for which a finitary unification algorithm exists. Applications of these algorithms include the analysis of all security properties that are modelled using trace equivalence, such as resistance to off-line guessing attacks, anonymity, strong secrecy, and e-voting secrecy, provided that the underlying equational theory is generated by a subterm convergent rewriting system.

Implementation. The goal of this work is the automated analysis of security protocols. This requires fully implementing our procedure for deciding equivalence of sets of constraint systems and integrating it with a tool which, given a protocol, generates the processes representing bounded executions of that protocol and the constraint systems necessary to solve trace equivalence.

At present we have prototype implementations of our (Φ, D)-unification procedure and our procedure for deciding ϵ-static equivalence, that is, D-static equivalence for the empty deducibility constraint system ϵ. This corresponds to the natural generalization of the notion of static equivalence when generalized frames (containing recipe variables) are considered. With minimal optimizations and running in a regular computer, the former terminates in a few milliseconds and the latter in at most a few seconds for moderately sized inputs. It is worth noting that our procedure for ϵ-static equivalence involves computing an ϵ-saturation of a frame.

Despite its more involved definition, the complexity of computing D-saturations is often not greater than that of computing ϵ-saturations, as the added constraints mean that the execution does not branch as much.

However, as pointed out in [64], the greatest complexity blow-up in constraint solving-based security protocol analysis is the number of constraint systems that need to be considered for representing even a small number of protocol sessions. It is our hope that this problem may be significantly reduced by using techniques such as constraint differentiation [127].

Future work. There are several directions in which our results can be expanded and improved. First, we would like to prove that our procedure terminates for subterm convergent equational theories, as described in Section 4.5.3. Other equational theories for which we believe that termination holds include blind signatures [5], malleable encryption, encryption with the prefix property [76], and trapdoor commitment [69]. Second, we would like to adapt our procedure to the more general problem of equivalence of sets of constraint systems, thereby supporting the analysis of trace equivalence of (any two) bounded processes. We believe that our technique is well-suited to this generalization. Finally, we would also like to investigate the termination of our Φ-unification procedure: Namely, we would like to obtain sufficient conditions for its termination, as well as investigating the decidability of the existence of finite complete sets of (Φ, D)-unifiers.

More broadly, trace equivalence and other notions of equivalence in the applied-pi calculus remain of crucial importance in security protocol analysis and are far less studied than trace-based properties. Importantly, despite the practical relevance of AC operators such as XOR or Diffie-Hellman, no procedures exist to decide trace equivalence in the presence of such operators together with standard cryptographic primitives (although [91] provides an algorithm for these equational theories when considered in isolation). Therefore, deciding equivalence properties in the presence of AC operators and standard cryptographic primitives remains a challenging and important research goal.

6.2 Symbolic Probabilistic Protocol Analysis

In Chapter 5 we presented a symbolic, automatable probabilistic framework for the analysis of security protocols. Our framework allows one to express properties of cryptographic primitives beyond standard equational properties, thereby modeling a stronger attacker than in the standard Dolev-Yao model. We illustrated the usefulness of this approach by modeling non-trivial properties of RSA encryption and using them to analyze off-line guessing attacks on the EKE protocol, currently outside the scope of other symbolic methods.

We have proposed a probability distribution based on interpreting functions as random oracles subject to satisfying the properties of cryptographic primitives described in our setup. This is a non-trivial generalization of the random oracle

model. By using this probability distribution, we can reason about an attack's success probability. We provide a prototype implementation of our methods, which computes probabilities in our formalization of a Dolev-Yao attacker using RSA asymmetric encryption. Our implementation is available in [3].

More generally, our approach can be used to analyze a broad range of attacks and weaknesses of cryptographic primitives that could not previously be analyzed by symbolic models. These include some forms of cryptanalysis (such as differential cryptanalysis to AES, DES or hash functions, as in [128]) and side-channel attacks [114]. Symbolic methods are also ill-suited to the analysis of short-string authentication, used in device pairing protocols [119], and distance-bounding protocols relying on rapid-bit exchange, such as [129], as their analysis is intrinsically probabilistic. However, such protocols are amenable to analysis using our framework.

As future work, we plan to integrate this approach with a symbolic protocol model-checker capable of generating protocol execution traces and the probabilities relevant for deciding whether a trace allows an attack, even if only against passive attackers. We expect that such an approach will allow us to find numerous new protocol attacks which depend on the cryptographic primitives used and their implementations.

Appendix A

Proofs for Chapter 3

In this Appendix we provide proofs for the results given in Chapter 3.

A.1 Auxiliary Algorithms

Proof (Lemma 2)**.** We prove that, throughout all executions of the loop in lines 3–18, two loop invariants are preserved: (A) the forest $\mathcal{T}_{\mathsf{min}}$ is minimal, and (B), for each $v \in dom(\mathsf{min})$, $\mathsf{term}_{\mathcal{T}_{\mathsf{min}}}(\mathsf{min}(v)) = (\mathsf{term}_{\mathcal{T}}(v))\!\downarrow$. The proof is by induction on the number of executions of the loop. Clearly, the initial definitions of min and $\mathcal{T}_{\mathsf{min}}$ on lines 1 and 2 satisfy these properties.

We prove that these properties are preserved by each execution of the inner loop (lines 4–16), that is, by each visit to a node in V. If $\mathsf{out}_{\mathcal{T}}(v) \not\subseteq dom(\mathsf{min})$, then neither min nor $\mathcal{T}_{\mathsf{min}}$ are changed. Otherwise, by the induction hypothesis, we have $\mathsf{term}_{\mathcal{T}_{\mathsf{min}}}(\mathsf{min}(\mathsf{e}_{i,t}(v))) = (\mathsf{term}_{\mathcal{T}}(\mathsf{e}_{i,\mathcal{T}}(v)))\!\downarrow$ for each i.

If the condition on line 7 holds, then no other vertices are added to $\mathcal{T}_{\mathsf{min}}$, so that (A) remains valid. We also have $\mathsf{term}_{\mathcal{T}_{\mathsf{min}}}(v_{\mathsf{min}}) = (\mathsf{term}_{\mathcal{T}}(v))\!\downarrow$ (by the induction hypothesis), and thus (B) is also preserved after line 8.

Suppose then the condition on line 7 does not hold. There are two cases: either the condition on line 10 holds (case (1)) or not (case (2)). In case (1), $r\sigma_l$ is a proper subterm of $l\sigma_l$ (otherwise the rewriting system would not be terminating); thus, $r\sigma_l$ is a subterm of $\mathsf{term}_{\mathcal{T}_{\mathsf{min}}}(\mathsf{min}(\mathsf{e}_{i,\mathcal{T}}(v)))$ for some i. Therefore, there is a vertex $v_r \in V_{\mathsf{min}}$ such that $\mathsf{term}_{\mathcal{T}_{\mathsf{min}}}(v_r) = r\sigma_l$. We have:

$$
\begin{aligned}
&\mathsf{term}_{\mathcal{T}}(v)\\
&\quad = (\lambda(v))(\mathsf{term}_{\mathcal{T}}(\mathsf{e}_{1,\mathcal{T}}(v)), \ldots, \mathsf{term}_{\mathcal{T}}(\mathsf{e}_{ar(\lambda(v)),\mathcal{T}}(v)))\\
&\quad \approx_{\mathcal{R}} (\lambda(v))(\mathsf{term}_{\mathcal{T}_{\mathsf{min}}}(\mathsf{min}(\mathsf{e}_{1,\mathcal{T}}(v))), \ldots, \mathsf{term}_{\mathcal{T}_{\mathsf{min}}}(\mathsf{min}(\mathsf{e}_{ar(\lambda(v)),\mathcal{T}}(v))))\\
&\quad \rightarrow_{\mathcal{R}} r\sigma_l\\
&\quad = \mathsf{term}_{\mathcal{T}_{\mathsf{min}}}(v_r).
\end{aligned}
$$

By the induction hypothesis, $\mathsf{term}_{\mathcal{T}_{\mathsf{min}}}(v_r)$ is in normal form. Thus, condition (B) is preserved. Condition (A) is preserved since no vertex is added to $\mathcal{T}_{\mathsf{min}}$.

Now, consider the case when the condition on line 10 is false. In this case, we add a vertex v_{min} to the forest $\mathcal{T}_{\mathsf{min}}$ and set $\mathsf{min}(v) = v_{\mathsf{min}}$. After updating $\mathcal{T}_{\mathsf{min}}$ in lines 12–16, we have:

$$\begin{aligned}\mathsf{term}_{\mathcal{T}_{\mathsf{min}}}(v_{\mathsf{min}}) &= (\lambda(v))(\mathsf{term}_{\mathcal{T}_{\mathsf{min}}}(\mathsf{min}(\mathsf{e}_{1,\mathcal{T}}(v))), \ldots, \\ &\qquad \mathsf{term}_{\mathcal{T}_{\mathsf{min}}}(\mathsf{min}(\mathsf{e}_{ar(\lambda(v)),\mathcal{T}_{\mathsf{min}}}(v)))) \\ &\approx_{\mathcal{R}} (\lambda(v))(\mathsf{term}_{\mathcal{T}}(\mathsf{e}_{1,\mathcal{T}}(v)), \ldots, \mathsf{term}_{\mathcal{T}}(\mathsf{e}_{ar(\lambda(v)),\mathcal{T}}(v))) \\ &= \mathsf{term}_{\mathcal{T}}(v),\end{aligned}$$

where the second equality uses the induction hypothesis. Since the condition on line 10 is false, we conclude that this term is in normal form (note that all its proper subterms are in normal form by the induction hypothesis). Thus, (B) is preserved.

Since the condition on line 7 is also false, it is clear that there is no other vertex $v'_{\mathsf{min}} \in V'_{\mathsf{min}}$ such that

$$\begin{aligned}\mathsf{term}_{\mathcal{T}_{\mathsf{min}}}(v'_{\mathsf{min}}) &= (\lambda(v))(\mathsf{term}_{\mathcal{T}_{\mathsf{min}}}(\mathsf{min}(\mathsf{e}_{1,\mathcal{T}}(v))), \ldots, \\ &\qquad \mathsf{term}_{\mathcal{T}_{\mathsf{min}}}(\mathsf{min}(\mathsf{e}_{ar(\lambda(v)),\mathcal{T}_{\mathsf{min}}}(v)))) \\ &= \mathsf{term}_{\mathcal{T}_{\mathsf{min}}}(v_{\mathsf{min}}).\end{aligned}$$

We conclude that (A) is also preserved.

It is easy to see that, in the end of the loop in lines 3–18, $V \subseteq dom(\mathsf{min})$. At this point of the execution of the algorithm, $\mathcal{T}_{\mathsf{min}}$ is a minimal DAG-forest and $\mathsf{min}\colon V \to V_{\mathsf{min}}$ such that, for each $v \in V$, $\mathsf{term}_{\mathcal{T}_{\mathsf{min}}}(\mathsf{min}(v)) = (\mathsf{term}_{\mathcal{T}}(v))\downarrow$. After lines 19–20, $\mathcal{T}_{\mathsf{min}}$ is still minimal, and $\mathsf{term}_{\mathcal{T}_{\mathsf{min}}}(v)$ is in normal form for all vertices $v \in V\mathsf{min}$. It is clear that $\mathsf{roots}(\mathcal{T}_{\mathsf{min}}) \subseteq \mathsf{min}[\mathsf{roots}(\mathcal{T})]$; thus, for all $v_{\mathsf{min}} \in \mathsf{roots}(\mathcal{T}_{\mathsf{min}})$, there is v in $\mathcal{T}$ such that $\mathsf{min}(v) = v_{\mathsf{min}}$, and we have $\mathsf{term}_{\mathcal{T}_{\mathsf{min}}}(v_{\mathsf{min}}) = (\mathsf{term}_{\mathcal{T}}(v))\downarrow$. Finally, for all $v \in \mathsf{roots}(\mathcal{T})$, $\mathsf{min}(v)$ a vertex in $\mathcal{T}_{\mathsf{min}}$ such that $\mathsf{term}_{tmin}(v_{\mathsf{min}}) = (\mathsf{term}_{T}(v))\downarrow$. We conclude that $\mathcal{T}_{\mathsf{min}}$ is a minimal normal-form of $\mathcal{T}$, as desired, and we have already seen that min satisfies the lemma.

To estimate the complexity of the algorithm, note first that at most $|V_{\mathsf{min}}| \leq |V|$, and adding each vertex in

Consider then a visit to a vertex v. There number of rules, as well as their size, is $\mathcal{O}(1)$. Checking whether the term represented by v matches an instance of a rule involves only checking the labels of a constant number of vertices, and possibly comparing two vertices for equality if the same variable occurs twice in l (note that, since $\mathcal{T}_{\mathsf{min}}$ is minimal, two vertices v and v' represent the same term if and only if $v = v'$). These checks take time $\mathcal{O}(\log|\mathcal{T}|)$. Our choice of data structure ensures that the test in line 7, and possibly adding a vertex to $\mathcal{T}_{\mathsf{min}}$ (in lines 12—16) can also be done in time $\mathcal{O}(\log|T|)$. Thus, we obtain a complexity of $\log|\mathcal{T}|$ for each visit to a vertex.

To count the number of visits to vertices, observe that each leaf is visited once and each other vertex v is visited at most $|\mathsf{out}_{\mathcal{T}}(v)|$ times. Thus, the total number of visits is at most

$$|\mathsf{leaves}(\mathcal{T})| + \sum_{v \in V} |\mathsf{out}_{\mathcal{T}}(v)| = |\mathsf{leaves}(\mathcal{T})| + |E| \in \mathcal{O}(|\mathcal{T}|).$$

We obtain a total complexity of $\mathcal{O}(|\mathcal{T}| \log |\mathcal{T}|)$.

□

Proof (Lemma 3)**.** $\mathcal{T}_{\phi,\mathcal{R}}$ contains one vertex for each $s \in sub(ran(\phi)) \cup \{\tau\}$ and at most $\mathcal{O}(1)$ vertices for each $(l \to r) \in \mathcal{R}$ and each substitution $\sigma_l \colon vars(l) \to sub(ran(\phi)) \cup \{\tau\}$. There are $\mathcal{O}(|\phi|^{nvars(\mathcal{R})})$ such substitutions, and we conclude that $|\mathcal{T}_{\phi,\mathcal{R}}| \in \mathcal{O}(|\phi|^{nvars(\mathcal{R})})$. The fact that $\mathcal{T}_{\phi,\mathcal{R}}$ is minimal is clear because $\mathcal{T}(\phi)$ is minimal and line 12 ensures that $\mathcal{T}_{\phi,\mathcal{R}}$ is kept minimal whenever we add new vertices.

To obtain our complexity estimate, we first note that each execution of the loop in lines $12 - 13$ takes only time $\mathcal{O}(\log |\phi|)$. This is because at most $|l| \in \mathcal{O}(1)$ vertices need to be added to the forest. For each of these vertices we need to check whether $\mathcal{T}_{\phi,\mathcal{R}}$ already contains a vertex representing the same term represented by that vertex, which can be done in time $\mathcal{O}(\log |\mathcal{T}_{\phi,\mathcal{R}}|)$. Since $|\mathcal{T}_{\phi,\mathcal{R}}| \in \mathcal{O}(|\phi|^{nvars(\mathcal{R})})$, this is the same as $\mathcal{O}(\log |\phi|)$. Adding the vertex to $\mathcal{T}_{\phi,\mathcal{R}}$ can be done with the same complexity. Computing the initial forest $\mathcal{T}_\phi$ can be done in time $\mathcal{O}(|\mathcal{T}| \log |\mathcal{T}|)$ (by Lemma 2), and the loop in lines $12 - 13$ is executed $\mathcal{O}(|\phi|^{nvars(\mathcal{R})})$ times; thus, we obtain a total complexity of $\mathcal{O}(|\phi|^{nvars(\mathcal{R})} \log |\phi|)$.

To see (2), we note that $\mathsf{rw}(v)$ can be computed from v in $\mathcal{O}(1)$ time (for example, by storing $\mathsf{rw}(v)$ as a data member in the object used to represent v). By a similar reasoning we obtain property (3), noting that the values stored under ζ here are DAG-representations of elements in $dom(\phi_s)$ and τ.

Property (8) follows from the loop in lines $11 - 13$. Property (4) is a consequence of (8) and the facts that $\mathcal{T}_{\phi,\mathcal{R}}$ is minimal and $\mathcal{R}$ is not empty.

Property (5) is ensured by the instruction on line 13, and properties (6) and (7) are ensured by the initialization of ζ in lines 4–7.

□

Lemma 10. *Let t be a term and $\phi = \nu\tilde{n}.\sigma$ be a frame. Finding (or determining that there is no) $t' \in T(\phi)$ such that $t'\phi = t$ can be done in time*

$$\mathcal{O}((|t| + |\phi|) \log(|t| + |\phi|)).$$

Proof. Note that such a t' exists if and only if either $t \in ran(\phi)$, $t \in \mathcal{N} \setminus \tilde{n}$ or $t = f(t_1, \ldots, t_n)$ for some $f \in \Sigma_n$ and some $t_1, \ldots, t_n \in \phi[T(\phi)]$. Thus, at most, we need to check whether $s \in (\mathcal{N} \setminus \tilde{n}) \cup ran(\phi)$ for each subterm $s \in sub(t)$. There are at most $|t|$ such subterms. Creating a minimal tree containing all the subterms of t and all subterms of terms in the range of ϕ can be done in time $\mathcal{O}((|t| + |\phi|) \log(|t| + |\phi|))$ and, by using such a tree, this check takes time $\mathcal{O}(\log(|t| + |\phi|))$ for each $s \in sub(t)$. We obtain a total complexity of

$$\mathcal{O}((|t| + |\phi|) \log(|t| + |\phi|)),$$

as stated by the Lemma. Note that, if one such term t' exists, then it is built by performing this procedure top-down on the subterms of t.

□

Lemma 11. *Let $\mathcal{R}$ be a subterm convergent rewriting system, $(l \to r) \in \mathcal{R}$ be a rule, and $t = f(t_1, \ldots, t_n)$ be a term such that all t_i are in normal form and $t = l\sigma_l$ for some $\sigma_l \colon vars(l) \to T_\Sigma(\mathcal{N})$. Then $r\sigma_l$ is in normal form.*

Proof. Either $r\sigma_l \in T(\Sigma, \emptyset)$ or $r\sigma_l \in sub(f(t_1, \ldots, t_n))$. In the first case, $r\sigma_l$ is in normal form by hypothesis. In the second case, $r\sigma_l$ must be a proper subterm of $l\sigma_l$ (otherwise the rewriting system would not be convergent). But $l\sigma_l = f(t_1, \ldots, t_n)$, and since $t_1, \ldots, t_n$ are in normal form, so are all their subterms. We conclude that all proper subterms of $l\sigma_l$ are in normal form and the result follows. □

A.2 Saturation Algorithm

Proof (Lemma 4)**.** We prove that, for all $\zeta \in T(\phi)$:

(1) $(\zeta\phi)\!\downarrow \ \in \phi_s[T(\phi_s)]$;

(2) if $(\zeta\phi)\downarrow$ is represented by some vertex v in $\mathcal{T}_{\phi,\mathcal{R}}$ (that is, if there exists $v \in V_{\phi,\mathcal{R}}$ such that $(\zeta\phi)\!\downarrow = \mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(v)$), then $v \in dom([\![\cdot]\!]^{DAG}_\Phi)$ and there is $\zeta' \in T(\phi_s)$ such that $(\zeta'\phi_s)\!\downarrow = (\zeta\phi)\!\downarrow$ and $[\![v]\!]^{DAG}_\Phi$ is a DAG-representation of $[\![\zeta']\!]_\Phi$.

The proof is by induction on ζ. If $\zeta \in dom(\phi)$, then (1) holds due to the initialization of $[\![\cdot]\!]^{DAG}_\Phi$ in line 2. (2) holds due to the property (6) in Lemma 3. If $\zeta \in \mathcal{N} \setminus \tilde{n}$, then (1) trivially holds, and (2) holds because in the first execution of the loop 5–21 the algorithm visits the leaves of $\mathcal{T}_{\phi,\mathcal{R}}$ and adds to the domain of $[\![\cdot]\!]^{DAG}_\Phi$ those vertices representing terms in $\mathcal{N} \setminus \tilde{n}$ (lines 7–8).

Now, suppose that $\zeta = f(\zeta_1, \ldots, \zeta_n)$ for some $f \in \Sigma_n$ and some $\zeta_1, \ldots, \zeta_n \in T(\phi)$. By the induction hypothesis, we have

$$(\zeta_1\phi)\!\downarrow, \ldots, (\zeta_n\phi)\!\downarrow \in \phi_s[T(\phi_s)].$$

We consider two cases: either $f((\zeta_1\phi)\!\downarrow, \ldots, (\zeta_n\phi)\!\downarrow)$ is in normal form or not. In the first case, clearly (1) holds. To prove (2), note that if there is some vertex v in $\mathcal{T}_{\phi,\mathcal{R}}$ representing

$$f((\zeta_1\phi)\!\downarrow, \ldots, (\zeta_n\phi)\!\downarrow) = (f((\zeta_1\phi)\!\downarrow, \ldots, (\zeta_n\phi)\!\downarrow))\!\downarrow,$$

then $\mathcal{T}_{\phi,\mathcal{R}}$ also contains vertices representing $(\zeta_1\phi)\!\downarrow, \ldots, (\zeta_n\phi)\!\downarrow$. Let $v_1, \ldots, v_n$ be these vertices. Let $dom_k([\![\cdot]\!]^{DAG}_\Phi)$ and $dom_k([\![\cdot]\!]_\Phi)$ be the domains of $[\![\cdot]\!]^{DAG}_\Phi$ and $[\![\cdot]\!]_\Phi$ (respectively) after the k-th execution of the loop in lines 5–21. Let $\mathsf{visitnow}_k$ denote the set visitnow at the beginning of the k-th execution of this loop. Now, there is k such that

$$\{v_i \mid i \in \{1, \ldots, n\}\} \in dom_{k+1}([\![\cdot]\!]^{DAG}_\Phi) \setminus dom_k([\![\cdot]\!]^{DAG}_\Phi);$$

that is, at the k-th iteration of the loop in lines 5–21, all vertices v_i have been added to $dom([\![\cdot]\!]_\Phi^{DAG})$. It is then clear that $v \in \mathsf{visitnow}_{k+1}$. By the induction hypothesis, there are ϕ_s-recipes $\zeta'_1, \ldots, \zeta'_n$ for $\zeta_1\downarrow, \ldots, \zeta_n\downarrow$ (respectively) such that $\mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(v_i) = [\![t'_i]\!]_\Phi$ for all i. Line 13 then ensures that $v \in dom([\![\cdot]\!]_\Phi^{DAG})$, and $[\![v]\!]_\Phi^{DAG}$ is a DAG-representation of $f(\mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(v_1), \ldots, \mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(v_n))$. This is the same as $[\![f(\zeta'_1, \ldots, \zeta'_n)]\!]_\Phi$, and $f(\zeta'_1, \ldots, \zeta'_n)$ is a ϕ_s-recipe for $f((\zeta_1\phi)\downarrow, \ldots, (\zeta_n\phi)\downarrow) = (\zeta\phi)\downarrow$, proving (2).

Let $\zeta' = f((\zeta_1\phi)\downarrow, \ldots, (\zeta_n\phi)\downarrow)$, and suppose then that ζ' is not in normal form. In this case, there exist $(l \to r) \in \mathcal{R}$ and a substitution $\sigma_l\colon vars(l) \to T_\Sigma(\mathcal{N})$ such that $f((\zeta_1\phi)\downarrow, \ldots, (\zeta_n\phi)\downarrow) = l\sigma_l$. Since $r\sigma_l$ is a proper subterm of $l\sigma_l$, we conclude that $r\sigma_l$ is in normal form.

We again consider two cases: $r\sigma_l \in sub(ran(\phi))$ and $r\sigma_l \notin sub(ran(\phi))$. If $r\sigma_l \notin sub(ran(\phi))$, then either $r\sigma_l \in T_\Sigma(\emptyset)$, or $r\sigma_l \in sub(l\sigma_l)$. In the first case (1) is clear. (2) holds because we have already proven that it holds for all the leaves of $\mathcal{T}_{\phi,\mathcal{R}}$. Now, we consider the second case. We have $l\sigma_l \in \phi_s[T(\phi_s)]$. The check in line 16 implies that all terms in the range of $[\![\cdot]\!]_\Phi$ are in $sub(ran(\phi))$, and thus $sub(ran(\phi_s)) \subseteq sub(ran(\phi))$. Since $l\sigma_l \in \phi_s[T(\phi_s)]$, this implies that, for all positions p of $l\sigma_l$ such that $l\ |_p{=}\ r\sigma_l$, we must have $p \in pos(\zeta)$ and $\zeta\ |_p\ \phi_s = r\sigma_l$. Thus, $r\sigma_l \in \phi_s[T(\phi_s)]$, and (1) holds.

Consider now the case that $r\sigma_l \in sub(ran(\phi))$. Since $l\sigma_l \in \phi_s[T(\phi_s)]$ and we have seen that $sub(ran(\phi_s)) \subseteq sub(ran(\phi))$, it follows that, for each $x \in vars(l)$, either $x\sigma_l \in sub(ran(\phi))$ or $x\sigma_l \in \phi_s[T(\phi_s)]$. In the second case, for each position $p \in pos(l)$ such that $l\ |_p{=}\ x$, we must have $p \in pos(\zeta')$ and $\zeta'\ |_p\ \phi_s = x\sigma'_l$.

Consider the substitution $\sigma'_l\colon vars(l) \to T_\Sigma(\mathcal{N})$ such that, for each $x \in vars(l)$:

$$x\sigma'_l = \begin{cases} x\sigma_l & \text{if } x\sigma_l \in sub(ran(\phi)) \\ \tau & \text{if } x\sigma_l \notin sub(ran(\phi)) \end{cases}.$$

Consider the set of positions $p \in pos(l)$ such that $l\ |_p{=}\ x$ for some variable x such that $x\sigma_l \notin sub(ran(\phi))$. Consider the recipe ζ'' obtained from ζ' by replacing the subrecipes $\zeta'\ |_p$ of ζ' at such positions p by τ. Then, we have that $\zeta''\phi_s = l\sigma'_l$, implying that $l\sigma'_l \in \phi_s[T(\phi_s)]$.

Let $l = f(l_1, \ldots, l_n)$. For each $i \in \{1, \ldots, n\}$, $l_i\sigma'_l$ is in normal form, since it can be obtained by replacing subterms of $(\zeta_i\phi)\downarrow$ by τ. Moreover, as $r\sigma_l \in sub(ran(\phi))$, we have that, for each $x \in vars(r)$, $x\sigma_l \in sub(ran(\phi))$; thus, $r\sigma_l = r\sigma'_l$. By Property (8) of Lemma 3, there is $v \in V_{\phi,r}$ such that $\mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(v) = l\sigma'_l$, $v \in dom(\mathsf{rw})$, and $\mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(\mathsf{rw}(v)) = r\sigma'_l$. Thus, there are vertices $v_i \in V_{\phi,r}$ such that $\mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(v_i) = l_i\sigma'_l$ for each $i \in \{1, \ldots, n\}$, and by the induction hypothesis we conclude that $v_i \in dom([\![\cdot]\!]_\Phi)$ for each i.

Now, there is k such that

$$\{v_i \mid i \in \{1, \ldots, n\}\} \in dom_{k+1}([\![\cdot]\!]_\Phi^{DAG}) \setminus dom_k([\![\cdot]\!]_\Phi^{DAG});$$

that is, at the k-th iteration of the loop in lines 5–21 all vertices v_i have been added to $dom([\![\cdot]\!]_\Phi^{DAG})$. It is then clear that $v \in \mathsf{visitnow}_{k+1}$. By a reasoning similar to the

one used above, line 13 ensures that $[\![v]\!]^{DAG}_{\Phi}$ is a DAG-representation of $[\![\zeta']\!]_{\Phi}$, where ζ' is a ϕ_s-recipe for $r\sigma'_l$. Now, if x is the chosen fresh variable, then $x\phi_s = r\sigma'_l$, and line 18 ensures that $r\sigma'_l = (\zeta\phi) \downarrow\in ran(\phi_s)$. Lines 15 and 19 imply that $[\![\mathsf{rw}(v)]\!]^{DAG}_{\Phi} = [\![x]\!]_{\Phi}$, and x is a ϕ_s-recipe for $r\sigma'_l = \mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(\mathsf{rw}(v))$. We have $[\![x]\!]_{\Phi} \in T(\phi)$ by the induction hypothesis, as before. This proves both (1) and (2).

ϕ_s is a saturation due to the property (1) just proved and from the fact that, for all $x \in dom(\phi_s)$, $[\![x]\!]_{\Phi}$ is a DAG-representation of a ϕ-recipe for $x\phi_s$.

To see that $|[\![v]\!]_{\Phi}| \in \mathcal{O}(|\phi|)$, we prove that in fact $|[\![v]\!]^{DAG}_{\Phi}| \in \mathcal{O}(|\phi|)$ for all $v \in dom([\![\cdot]\!]^{DAG}_{\Phi}) \cap V_\phi$. To prove this, suppose that a vertex v is added to the domain of $[\![\cdot]\!]^{DAG}_{\Phi}$. This can happen executing either line 12 or line 14. In line 12, it is clear that only one vertex is added to the vertices occurring in the DAGs $[\![v]\!]^{DAG}_{\Phi}$ for $v \in dom([\![\cdot]\!]^{DAG}_{\Phi}) \cap V_\phi$. In line 14, $[\![\mathsf{rw}(v)]\!]^{DAG}_{\Phi}$ is a DAG-representation of an instance of the left-hand side of a rewrite rule whose variables are instantiated with elements of $sub(ran(\phi)) \cup \Upsilon$. Thus, $[\![x]\!]_{\Phi}$ contains at most $|l| \in \mathcal{O}(1)$ vertices that do not occur in the DAGs $[\![v]\!]_{\Phi}$ for $v \in dom([\![\cdot]\!]_{\Phi}) \cap V_\phi$. We conclude that the set of vertices occurring in the DAG-representations $[\![v]\!]_{\Phi}$ for $v \in dom([\![\cdot]\!]_{\Phi}) \cap V_\phi$ is augmented at most $|sub(ran(\phi))|$ times, each time adding at most $\mathcal{O}(|1|)$ vertices. Thus, the set of vertices occurring in a DAG $[\![v]\!]_{\Phi}$ for some $v \in dom([\![\cdot]\!]_{\Phi}) \cap V_\phi$ has at most $\mathcal{O}(|\phi|)$ elements. Since whenever we increase the range of $[\![\cdot]\!]_{\Phi}$ with a new element $[\![x]\!]_{\Phi}$ (in line 19), $[\![x]\!]_{\Phi} = [\![\mathsf{rw}(v)]\!]_{\Phi}$ for some vertex $\mathsf{rw}(v) \in dom([\![\cdot]\!]_{\Phi}) \cap V_\phi$, it follows that $|[\![x]\!]_{\Phi}| \in \mathcal{O}(|\phi|)$.

To prove that $\mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(v) \in \phi_s[T(\phi_s)]$ implies $v \in dom(\zeta)$ we proceed by induction. Observe first that all vertices representing terms in $ran(\phi)$ are in the domain of $[\![\cdot]\!]^{DAG}_{\Phi}$, due to property (6) of 3. All other terms added to the range of ϕ_s are added in the execution of line 16. In this case, it is clear that $\mathsf{rw}(v)$ is the only vertex in $\mathcal{T}_{\phi,\mathcal{R}}$ representing the new element $x\phi_s$ in the range of ϕ_s, and $\mathsf{rw}(v) \in dom([\![\cdot]\!]^{DAG}_{\Phi})$; thus, the property still holds. If $\mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(v) \in \mathcal{N} \setminus \tilde{n}$, then $v \in \mathsf{leaves}(\mathcal{T})$, and the first execution of the loop in lines 5–21 ensures that $v \in dom([\![\cdot]\!]^{DAG}_{\Phi})$. Finally, if

$$\mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(v) = f(t_1, \ldots, t_n)$$

for some $t_1, \ldots, t_n \in \phi_s[T(\phi_s)]$, then there are vertices $v_1, \ldots, v_n$ such that $\mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(v_i) = t_i$ for all i, and, by the induction hypothesis, $v_i \in dom([\![\cdot]\!]_{\Phi})$ for all i. By a reasoning similar to the one used above, there is a minimum k such that

$$v_1, \ldots, v_n \in dom_k([\![\cdot]\!]^{DAG}_{\Phi}),$$

and it follows that

$$v \in dom_k([\![\cdot]\!]^{DAG}_{\Phi}) \subseteq dom([\![\cdot]\!]^{DAG}_{\Phi}).$$

This concludes the proof.

To estimate the complexity of the algorithm, observe that the number of vertices in $\mathcal{T}_{\phi,\mathcal{R}}$ is $\mathcal{O}(|\phi|^{nvars}(\mathcal{R}))$. Each leaf is visited once, and each other vertex is visited at most once for each outgoing edge (because a vertex v is added to the

list of vertices to visit in the next execution of the loop if, in the current execution of the loop, the algorithm adds to the domain of $[\![\cdot]\!]^{DAG}_{\Phi}$ a vertex v' such that there is an edge from v to v'). We obtain that the total number of visits to vertices is $\mathcal{O}(|\phi|^{nvars(\mathcal{R})})$.

Consider now each visit to a vertex. Unless the condition in line 14 is verified, all operations performed by the visit take time at most $\mathcal{O}(\log|\phi|)$. If this condition is verified, it is sufficient to associate to $[\![x]\!]_{\Phi}$ a pointer to the vertex v; the desired DAG-representation can then be retrieved in logarithmic time by computing $[\![\mathsf{rw}(v))]\!]^{DAG}_{\Phi}$. This can be done in $\mathcal{O}(1)$. This loop is only executed at most $\mathcal{O}(|\phi|)$ times, since each of its executions adds to the domain of $[\![\cdot]\!]_{\Phi}$ a vertex representing a term in $sub(ran(\phi)) \cup \{\tau\}$ and there are only $\mathcal{O}(|\phi|)$ such terms. Therefore, the total time spent on the loop executed if the condition in line 14 is $\mathcal{O}(|\phi|)$. Note that, to efficiently decide whether this condition holds, we must store the information of whether the term represented by a vertex is a subterm of the range of ϕ locally, *e.g.*, as a boolean field in the object representing that vertex.

We conclude that all visits take time $\mathcal{O}(|\phi|^{nvars(\mathcal{R})}\log|\phi|$, the same as that of constructing the forest $\mathcal{T}_{\phi,\mathcal{R}}$. Thus, the total complexity of the procedure is $\mathcal{O}(|\phi|^{nvars(\mathcal{R})}\log|\phi|)$.

□

A.3 Deducibility

Proof (Theorem 1)**.** By Lemma 2, computing (a minimal DAG-representation of) the normal form $t\downarrow$ of t takes time $\mathcal{O}(|t|\log|t|)$. By Lemma 4, computing the saturated frame ϕ_s takes time $\mathcal{O}(|\phi|^{nvars(\mathcal{R})}\log|\phi|)$. By Lemma 10, deciding whether $t\downarrow \in \phi_s[T(\phi_s)]$ can be done in time $\mathcal{O}(|\phi_s||t|)$, that is, $\mathcal{O}(|\phi|^2|t|)$, since $|\phi_s| \in \mathcal{O}(|\phi|^2)$. We thus obtain an overall complexity of

$$\mathcal{O}(|t|\log|t| + |\phi|^2|t| + |\phi|^{nvars(\mathcal{R})}\log|\phi|)$$

for the whole procedure.

□

A.4 Static Equivalence

Lemma 12. *Suppose that the frames ϕ and ϕ' are such that $T(\phi') = T(\phi)$. Assume that, for all $x \in dom(\phi_s)$ and $\zeta' \in T(\phi_s)$,*

$$x\phi_s = \zeta'\phi_s \Rightarrow [\![x]\!]_{\Phi}\phi' \approx_{\mathcal{R}} [\![\zeta']\!]_{\Phi}\phi'.$$

Then, for all $\zeta, \zeta' \in T(\phi_s)$, we have

$$\zeta\phi_s = \zeta'\phi_s \Rightarrow [\![\zeta]\!]_{\Phi}\phi' \approx_{\mathcal{R}} [\![\zeta']\!]_{\Phi}\phi'.$$

Proof. The lemma is proved by induction on ζ. For the base case, we must prove that the result holds for $\zeta \in \mathcal{N} \setminus \tilde{n}$ and for $\zeta \in dom(\phi_s)$. If $\zeta \in dom(\phi_s)$, the result is trivial. If $\zeta \in \mathcal{N} \setminus \tilde{n}$, then $\zeta'\phi_s = \zeta\phi_s$ implies $\zeta'\phi_s = \zeta$. Thus, either $\zeta' \in dom(\phi_s)$ or $\zeta' = \zeta$. In both cases, the result follows.

Now let $\zeta = f(\zeta_1, \ldots, \zeta_n)$ for some $\zeta_1, \ldots, \zeta_n \in T(\phi)$ and some $f \in \Sigma_n$. By the induction hypothesis, $\zeta_i\phi_s = \zeta_i'\phi_s$ implies $[\![\zeta_i]\!]_\Phi\phi' \approx_\mathcal{R} [\![\zeta_i']\!]_\Phi\phi'$. Now, $\zeta\phi_s = \zeta'\phi_s$ implies that either $\zeta' \in dom(\phi_s)$ or $\zeta' = f(\zeta_1', \ldots, \zeta_n')$ for some $\zeta_1', \ldots, \zeta_n' \in T(\phi)$ such that $\zeta_i'\phi_s = \zeta_i\phi_s$ for all $i \in \{1, \ldots, n\}$. In the first case, the result is trivial. In the second, it follows from the induction hypothesis.

□

Lemma 13. *Suppose that:*

- $T(\phi) = T(\phi')$*;*
- *for all $x \in dom(\phi_s)$ and all $\zeta' \in T(\phi_s)$ such that $x\phi_s = \zeta'\phi_s$, we have*

$$[\![x]\!]_\Phi\phi' \approx_\mathcal{R} [\![\zeta']\!]_\Phi\phi';$$

- *for all $(l \to r) \in \mathcal{R}$, all $\sigma_l \colon vars(l) \to sub(ran(\phi_s)) \cup \Upsilon$, if*

$$\zeta_l \in T(\phi_s) \qquad \textit{and} \qquad \zeta_l\phi_s = l\sigma_l,$$

there exists $\zeta_r \in T(\phi_s)$ such that

$$\zeta_r\phi_s = r\sigma_l \qquad \textit{and} \qquad [\![\zeta_l]\!]_\Phi\phi' \approx_\mathcal{R} [\![\zeta_r]\!]_\Phi\phi'.$$

Then, for all $\zeta \in T(\phi)$, there exists $\zeta_{\mathsf{nf}} \in T(\phi_s)$ such that

$$(\zeta\phi)\!\downarrow = \zeta_{\mathsf{nf}}\phi_s \qquad \textit{and} \qquad \zeta\phi' \approx_\mathcal{R} [\![\zeta_{\mathsf{nf}}]\!]_\Phi\phi'.$$

Proof. The proof is by induction on ζ. In the base case we have either $\zeta \in \mathcal{N}\setminus\tilde{n}$ or $t \in dom(\phi)$. In both cases we may choose $\zeta_{\mathsf{nf}} = \zeta$, and the result follows.

Now suppose that $\zeta = f(\zeta_1, \ldots, \zeta_n)$ for some $f \in \Sigma_n$ and some $\zeta_1, \ldots, \zeta_n \in T(\phi)$. By the induction hypothesis, there are $\zeta_{1,\mathsf{nf}}, \ldots, \zeta_{n,\mathsf{nf}} \in T(\phi_s)$ such that $\zeta_{i,\mathsf{nf}}\phi_s = (\zeta_i\phi)\!\downarrow$ and $\zeta_i\phi' \approx_\mathcal{R} [\![\zeta_{i,\mathsf{nf}}]\!]_\Phi\phi'$ for all $i \in \{1, \ldots, n\}$. It is clear that

$$\begin{aligned} & f(\zeta_{1,\mathsf{nf}}, \ldots, \zeta_{n,\mathsf{nf}})\phi_s \\ &= f(\zeta_{1,\mathsf{nf}}\phi_s, \ldots, \zeta_{n,\mathsf{nf}}\phi_s) \\ &\approx_\mathcal{R} f(\zeta_1\phi', \ldots, \zeta_n\phi') \\ &= \zeta\phi'. \end{aligned} \tag{A.1}$$

Thus, if $f(\zeta_{1,\mathsf{nf}}, \ldots, \zeta_{n,\mathsf{nf}})\phi_s$ is in normal form, it is sufficient to take $\zeta_{\mathsf{nf}} = \zeta_{\mathsf{nf}}'$.

Otherwise, let

$$\zeta_{\mathsf{nf}}^l = f(\zeta_{1,\mathsf{nf}}, \ldots, \zeta_{n,\mathsf{nf}}) \in T(\phi_s),$$

and suppose that

$$\zeta_{\mathsf{nf}}^l\phi_s = f(\zeta_{1,\mathsf{nf}}\phi_s, \ldots, \zeta_{n,\mathsf{nf}}\phi_s)$$

is not in normal form. Then there is some rule $(l \to r) \in \mathcal{R}$ and some substitution $\sigma_l \colon vars(l) \to T_\Sigma(\mathcal{N})$ such that $l\sigma_l = f(\zeta_{1,\mathsf{nf}}, \ldots, \zeta_{n,\mathsf{nf}})\phi_s$ (because $\zeta_{i,\mathsf{nf}}\phi_s$ is in normal form for all i, by the induction hypothesis). By Lemma 11, we conclude that $r\sigma_l$ is in normal form. By the definition of saturation, there is $\zeta^r_{\mathsf{nf}} \in T(\phi_s)$ such that $\zeta^r_{\mathsf{nf}}\phi_s = r\sigma_l$.

Consider an injective function $\tau \colon vars(l) \to \Upsilon$ (recall that, by construction, Υ is at least as large as the set of variables occurring in any rule). We define the substitution $\sigma'_l \colon vars(l) \to sub(ran(\phi)) \cup \Upsilon$ by:

$$x\sigma'_l = \begin{cases} x\sigma_l & \text{if } x\sigma_l \in sub(ran(\phi)) \\ \tau(x) & \text{if } x\sigma_l \notin sub(ran(\phi)) \end{cases} .$$

For each position $p \in pos(l)$ such that $l \mid_p = x$ for some variable x such that $x\sigma_l \notin sub(ran(\phi))$, we must have $p \in pos(\zeta^l_{\mathsf{nf}})$ and $\zeta^l_{\mathsf{nf}} \mid_p \phi_s = x\sigma_l$. Consider the recipe ζ_l obtained from ζ^l_{nf} by replacing the subrecipes $\zeta^l_{\mathsf{nf}} \mid_p$ of ζ^l_{nf} at such positions p by $\tau(x)$. We have

$$\zeta_l \in T(\phi_s) \qquad \text{and} \qquad \zeta_l\phi_s = l\sigma'_l.$$

We construct ζ_r such that

$$\zeta_r \in T(\phi_s) \qquad \text{and} \qquad \zeta_r\phi_s = r\sigma'_l$$

in the analogous way.

We have $\sigma'_l \colon vars(l) \to sub(ran(\phi)) \cup \Upsilon$, and thus, by hypothesis, there exists $\zeta'_r \in T(\phi_s)$ such that

$$\zeta'_r\phi_s = r\sigma'_l \qquad \text{and} \qquad [\![\zeta'_l]\!]_\Phi\phi' \approx_\mathcal{R} [\![\zeta'_r]\!]_\Phi\phi'.$$

Because

$$\zeta'_r\phi_s = \zeta_r\phi_s = r\sigma'_l,$$

the hypothesis and Lemma 12 imply that

$$[\![\zeta_r]\!]_\Phi\phi' \approx_\mathcal{R} [\![\zeta'_r]\!]_\Phi\phi' \approx_\mathcal{R} [\![\zeta_l]\!]_\Phi\phi'. \tag{A.2}$$

Suppose that $x \in vars(l)$ is some variable such that $x\sigma_l \notin sub(ran(\phi))$. Then, for each $p \in pos(l)$ such that $l \mid_p = x$ we have $\zeta^l_{\mathsf{nf}} \mid_p \phi_s = x\sigma_l$. Thus, by the hypothesis and Lemma 12, there is $t'_{\tau(x)}$ such that $[\![\zeta^l_{\mathsf{nf}} \mid_p]\!]_\Phi\phi_s = t'_{\tau(x)}$ for all such positions p. Moreover, for all such positions, we have

$$\zeta_l \mid_p \phi_s = \tau(x) \qquad \text{and} \qquad \zeta^l_{\mathsf{nf}} \mid_p \phi_s = x\sigma_l.$$

Since these are the only positions in which ζ_l and ζ'_{nf} differ, it follows that $[\![\zeta^l_{\mathsf{nf}}]\!]_\Phi\phi'$ can be obtained from $[\![\zeta_l]\!]_\Phi\phi'$ by replacing each occurrence of a name a in the range of τ by t'_a. By a similar argument, $[\![\zeta^r_{\mathsf{nf}}]\!]_\Phi\phi'$ can also be obtained from $[\![\zeta_r]\!]_\Phi\phi'$ by replacing each occurrence of a name $a \in \tau[vars(l)]$ by t'_a.

Because $\approx_{\mathcal{R}}$ is stable under substitution of names by arbitrary terms, it follows from equation (A.2) and equation (A.1) that

$$\zeta\phi' \approx_{\mathcal{R}} [\![\zeta^l_{\mathsf{nf}}]\!]_\Phi \phi' \approx_{\mathcal{R}} [\![\zeta^r_{\mathsf{nf}}]\!]_\Phi \phi'.$$

Since $\zeta^r_{\mathsf{nf}}\phi_s$ is in normal form, the result is proved by taking $\zeta_{\mathsf{nf}} = \zeta^r_{\mathsf{nf}}$.

□

Lemma 14. *Suppose that $T(\phi) = T(\phi')$. Then, there exist $\zeta, \zeta' \in T(\phi)$ such that*

$$\zeta\phi \approx_{\mathcal{R}} \zeta'\phi \qquad \textit{and} \qquad \zeta\phi' \not\approx_{\mathcal{R}} \zeta'\phi'$$

if and only if at least one of the following conditions holds:

(1) *there are $x \in dom(\phi_s)$ and $\zeta_x \in T(\phi_s)$ such that*

$$x\phi_s = \zeta_x\phi_s \qquad \textit{and} \qquad [\![x]\!]_\Phi \phi' \not\approx_{\mathcal{R}} [\![\zeta_x]\!]_\Phi \phi';$$

(2) *there is $(l \to r) \in \mathcal{R}$ and a substitution $\sigma_l \colon vars(l) \to sub(ran(\phi)) \cup \Upsilon$ such that, for all $\zeta_l, \zeta_r \in T(\phi_s)$, we have*

$$(\zeta_l\phi_s = l\sigma_l \wedge \zeta_r\phi_s = r\sigma_l) \Rightarrow [\![\zeta_l]\!]_\Phi \phi' \not\approx_{\mathcal{R}} [\![\zeta_r]\!]_\Phi \phi'.$$

Proof. ($\Leftarrow$) If (1) holds, the result is proved by taking $\zeta = [\![x]\!]_\Phi$ and $\zeta' = [\![\zeta_x]\!]_\Phi$. If (2) holds, the result is proved by taking $\zeta = [\![\zeta_l]\!]_\Phi$ and $\zeta' = [\![\zeta_r]\!]_\Phi$.

($\Rightarrow$) Let $\zeta, \zeta' \in T(\phi)$ be such that $\zeta\phi \approx_{\mathcal{R}} \zeta'\phi$. We show that if neither condition holds then $\zeta\phi' \approx_{\mathcal{R}} \zeta'\phi'$. Since $\zeta\phi \approx_{\mathcal{R}} \zeta'\phi$, we have $(\zeta\phi)\!\downarrow = (\zeta'\phi)\!\downarrow$, and we know by Lemma 1 that $(\zeta\phi)\!\downarrow \in \phi_s[T(\phi_s)]$. By Lemma 13, there are terms $\zeta_{\mathsf{nf}}, \zeta'_{\mathsf{nf}} \in T(\phi_s)$ such that

$$\zeta_{\mathsf{nf}}\phi_s = (\zeta\phi)\!\downarrow \qquad \text{and} \qquad [\![\zeta_{\mathsf{nf}}]\!]_\Phi \phi' \approx_{\mathcal{R}} [\![\zeta]\!]_\Phi \phi',$$

and analogously

$$\zeta'_{\mathsf{nf}}\phi_s = (\zeta'\phi)\!\downarrow \qquad \text{and} \qquad [\![\zeta'_{\mathsf{nf}}]\!]_\Phi \phi' \approx_{\mathcal{R}} [\![\zeta']\!]_\Phi \phi'.$$

Lemma 13 thus implies

$$\zeta\phi' \approx_{\mathcal{R}} [\![\zeta_{\mathsf{nf}}]\!]_\Phi \phi' \approx_{\mathcal{R}} [\![\zeta'_{\mathsf{nf}}]\!]_\Phi \phi' \approx_{\mathcal{R}} \zeta'\phi',$$

yielding the result.

□

Proof (Theorem 2)**.** We prove that lines 1-11 output `false` if and only if there are $\zeta, \zeta' \in T(\phi)$ such that $\zeta\phi \approx_{\mathcal{R}} \zeta'\phi$ and $\zeta\phi' \not\approx_{\mathcal{R}} \zeta'\phi'$. Line 12 outputs `false` if and only if the same property holds exchanging ϕ and ϕ'. If neither of the repetitions of the loop outputs `false`, then $\phi \approx^s_{\mathcal{R}} \phi'$, and the algorithm returns `true` as desired.

It is clear that if the cycle in lines 5—8 returns `false` then $[\![x_{i(k)}]\!]_\Phi$ and $[\![\zeta]\!]_\Phi$ are ϕ-recipes such that

$$[\![x_{i(k)}]\!]_\Phi\phi \approx_\mathcal{R} [\![\zeta]\!]_\Phi\phi \qquad \text{and} \qquad [\![x_{i(k)}]\!]_\Phi\phi' \not\approx_\mathcal{R} [\![\zeta]\!]_\Phi\phi'.$$

Thus, the two frames are not statically equivalent, and the algorithm outputs the correct result. Similarly, if the cycle in lines 9—11 returns `false`, then $[\![v]\!]^{DAG}_\Phi$ and $[\![(]\!]^{DAG}_\Phi \mathsf{rw}(v)$ witness the fact that $\phi \not\approx^s_\mathcal{R} \phi'$, and the algorithm correctly outputs `false`.c

To prove the converse, we show that: (A) condition (1) in Lemma 14 implies that the loop in lines 5-8 returns `false`; and (B) condition (2) in Lemma 14 implies that there is some vertex v for which the condition on lines 10 does not hold, and the cycle in lines 9-11 returns `false`.

To prove (A), suppose that there is $x \in dom(\phi_s)$ and $\zeta \in T(\phi_s)$ such that $x\phi_s = \zeta\phi_s$ and $[\![x]\!]_\Phi\phi' \not\approx_\mathcal{R} [\![\zeta]\!]_\Phi\phi'$. Let k be such that $x = x_{i(k)}$. There are two cases, depending on whether or not $\zeta \in dom(\phi_s)$.

If $\zeta \in dom(\phi_s)$, then there is k' such that $\zeta = x_{i(k')}$. Thus, either $k' > k$ or $k > k'$ (if $k = k'$ then $[\![x]\!]_\Phi\phi' \approx_\mathcal{R} [\![\zeta]\!]_\Phi\phi'$). If $k' > k$, then $x_{i(k')}\phi_s = x_{i(k)}\phi_s$, and $x_{i(k)} \in T(\phi(k'))$, and thus Algorithm 4 outputs `false` in lines 5—8. We reason analogously if $k > k'$.

If $\zeta \notin dom(\phi_s)$, let $x \in vars(t)$. We have $|\zeta\phi_s| > |x\phi_s|$ for all $x \in vars(t)$ and $x_{i(k)}\phi_s = t\phi_s$. Hence, $|x_{i(k)}\phi_s| > |x\phi_s|$, and $t \in T_{\phi_{s,k-1}}$ since $x_{i(k')} \notin vars(t)$ for all $k' \geq k$.

Now we only need to prove that it does not matter which recipe we choose in line 6. Suppose that the iterator i in this loop reaches k. Then, for all $j \in \{0, \ldots, k-1\}$, there is no $t \in T_{\phi_{s,j-1}}$ such that $x_{i(j)}\phi_s = t\phi_s$ and $x_{i(j)}\zeta\phi' \not\approx_\mathcal{R} t\zeta\phi'$. From what we have just proved, it follows that there is no $x \in \{x_{i(1)}, \ldots, x_{i(k-1)}\}$ and $t \in T_{\phi_{s,k-1}}$ such that $x\phi_s = t\phi_s$ and $x\zeta\phi' \not\approx_\mathcal{R} x\zeta\phi'$. From Lemma 12, we conclude that for all $t, t' \in T_{\phi_{s,k-1}}$, $t\phi_s = t'\phi_s$ implies $t\zeta\phi' \approx_\mathcal{R} t'\zeta\phi'$. Thus, if t and t' are $\phi_{s,k-1}$-recipes for $x_{i(k)}$, we have $t\zeta\phi' \approx_\mathcal{R} t'\zeta\phi$, and it does not matter which we choose.

To prove (B), suppose that condition (2) in Lemma 14 holds for a rewrite rule $l \to r$, a substitution σ_l and ϕ-recipes t_l and ζ_r for $l\sigma_l$ and $r\sigma_l$, respectively. Then, by property (8) of Lemma 3, there is a vertex $v \in dom(\mathsf{rw})$ such that $\mathsf{term}_{\mathcal{T}_{\phi,\mathcal{R}}}(v) = l\sigma_l$. Since $l\sigma_l \in \phi_s[T(\phi_s)]$, we conclude, by Lemma 4, that $v \in dom(\zeta)$, and $\zeta(v)$ is a (DAG-representation of) a ϕ-recipe for $l\sigma_l$. Similarly, $\mathsf{rw}(v) \in dom(\zeta)$, and $\zeta(\mathsf{rw}(v))$ is a (DAG-representation of) $r\sigma_l$. We have $\zeta(v)\phi = t_l\phi = l\sigma_l$. By Lemma 12, either condition (1) of Lemma 14 holds, and the correct output `false` has already been output by the first loop, or we have $\zeta(v)\phi' \approx_\mathcal{R} t_l\phi'$. Similarly, we conclude that either the first loop has already returned `false` or $\zeta(\mathsf{rw}(v))\phi' \approx_\mathcal{R} \zeta_r\phi'$. Thus, if $(\zeta(v))\phi' \approx_\mathcal{R} (\zeta(\mathsf{rw}(v)))\phi'$, we conclude that $t_l\phi' \approx_\mathcal{R} \zeta_r\phi'$, which contradicts our assumption. It follows that $(\zeta(v))\phi' \not\approx_\mathcal{R} (\zeta(\mathsf{rw}(v)))\phi'$, and the algorithm outputs the correct result `false`.

Choosing the bijection and storing the $\phi_{s,k}$ on lines 2 and 3 can be done in time $\mathcal{O}(|\phi| \log |\phi|)$. We have $|x_{i(k)}\phi_s| \in \mathcal{O}(|\phi|)$, and we have seen that $|\phi_s| \in |\phi|^2$.

Thus, deciding whether $x_{i(k)}\phi_s \in \phi_s[T_{\phi_{s,k-1}}]$ can be done in time $|\phi|^2 \log |\phi|$ by Lemma 10. The comparison on line 8 takes time $|\phi| \log |\phi|$.

We now show that, by traversing the tree $\mathcal{T}_{\phi,\mathcal{R}}$ bottom-up and performing a logarithmic-time operation in each node, it is possible to associate each node $v \in dom([\![\cdot]\!]_\Phi^{DAG})$ to the term $[\![v]\!]_\Phi^{DAG}\phi'\downarrow$. To do this, we first use Algorithm 1 to build the tree $\mathcal{T}_{\phi',\mathcal{R}}$, using time $\mathcal{O}(|\phi'|^{nvars(\mathcal{R})})$. Then, traversing the tree $\mathcal{T}_{\phi,\mathcal{R}}$ bottom-up, we perform the following operation for each vertex $v \in dom([\![\cdot]\!]_\Phi^{DAG})$: If $[\![v]\!]_\Phi^{DAG} \in dom(\phi)$, then $[\![v]\!]_\Phi^{DAG}\phi'\downarrow$ is represented by a node v in $\mathcal{T}_{\phi',\mathcal{R}}$, and can be computed in time $|[\![v]\!]_\Phi^{DAG}\phi'| \in \mathcal{O}(|\phi'|)$. If $[\![v]\!]_\Phi^{DAG} \in \mathcal{N}$, then $[\![v]\!]_\Phi^{DAG}\phi'\downarrow = [\![v]\!]_\Phi^{DAG}$, and it can be computed in linear-time. Otherwise, if $[\![v]\!]_\Phi^{DAG} = f(\zeta_1, \ldots, \zeta_n)$, then the vertex v has outgoing edges to vertices $v_1, \ldots, v_n$ such that $[\![v_1]\!]_\Phi^{DAG} = \zeta_1$, $\ldots$, $[\![v_n]\!]_\Phi^{DAG} = \zeta_n$. By construction, these vertices are associated to vertices $v'_1, \ldots, v'_n$ in $\mathcal{T}_{\phi',\mathcal{R}}$ such that, for each $i \in \{1, \ldots, n\}$,

$$\mathsf{term}_{\mathcal{T}_{\phi',\mathcal{R}}}(v'_i) = [\![v_i]\!]_\Phi^{DAG}\phi'\downarrow .$$

Therefore, we have

$$[\![v]\!]_\Phi^{DAG}\phi'\downarrow = f(\mathsf{term}_{\mathcal{T}_{\phi',\mathcal{R}}}(v'_1), \ldots, \mathsf{term}_{\mathcal{T}_{\phi',\mathcal{R}}}(v'_n))\downarrow .$$

Now, all proper subterms of

$$f(\mathsf{term}_{\mathcal{T}_{\phi',\mathcal{R}}}(v'_1), \ldots, \mathsf{term}_{\mathcal{T}_{\phi',\mathcal{R}}}(v'_n))\downarrow$$

are in normal form; since $\mathcal{R}$ is subterm convergent, its normal form can be obtained by applying at most one rewriting rule. Given a rewrite rule l, checking whether there exists a substitution α such that

$$l\alpha = f(\mathsf{term}_{\mathcal{T}_{\phi',\mathcal{R}}}(v'_1), \ldots, \mathsf{term}_{\mathcal{T}_{\phi',\mathcal{R}}}(v'_n))$$

can be done in time $|l|$. Checking whether there exists a rule $(l \to r) \in \mathcal{R}$ such that the above equation holds can thus be done in time

$$\mathcal{O}\left(\sum_{(l\to r)\in\mathcal{R}} |l|\right) = \mathcal{O}(1).$$

Computing the normal form amounts to checking whether this term is an instance of some rewrite rule and, if so, obtaining the proper subterm, which can be done in time $\mathcal{O}(\log(|\phi'|))$. Therefore, the whole procedure has complexity

$$\mathcal{O}((|\phi| + |\phi'|)^{nvars(\mathcal{R})} \log(|\phi| + |\phi'|)).$$

The loop in lines 9–11 requires comparing $\mathcal{O}(|\phi|^{nvars(\mathcal{R})})$ pairs of recipes for equality under ϕ'. By using the construction above, each such test can be done in time $\mathcal{O}(\log(|\phi'|))$, by obtaining, for each recipe ζ, the corresponding vertex $v' \in \mathcal{T}_{\phi,\mathcal{R}}$ such that

$$\zeta\phi'\downarrow = \mathsf{term}_{\mathcal{T}_{\phi',\mathcal{R}}}(v'),$$

and then checking whether these vertices are the same for each recipe in the pair. Thus, the whole loop takes time

$$\mathcal{O}((|\phi| + |\phi'|)^{nvars(\mathcal{R})} \log(|\phi| + |\phi'|)).$$

We obtain a total time complexity of at most

$$\mathcal{O}((|\phi| + |\phi'|)^{\max(2,nvars(\mathcal{R}))} \log(|\phi| + |\phi'|)).$$

□

Appendix B

Proofs for Chapter 4

In this Appendix we give proofs for the results in Chapter 4. The organization of this Appendix in sections closely follows the organization of Chapter 4. Section B.1 we prove properties of the (Φ, D, θ)-unification procedure described in Section 4.2.3. Section B.2 is dedicated to proving Theorem 5, which establishes the correctness of, and gives a termination result for, Algorithm 6. In Section B.3 we prove Theorem 6, concerning our algorithm for deciding static equivalence. Finally, in Section B.4 we prove our main result, Theorem 8, as well as Theorem 7, used in its proof.

B.1 Unification Algorithm

Proof (Lemma 5)**.** Since the range of ϱ contains no recipe variables and only fresh term variables, it is clear that $\mathcal{U}_\varrho$ is in linear-left form.

Suppose then that Δ_ϱ is a finite complete set of (Φ_θ, D_θ)-solutions of $\mathcal{U}_\varrho$. For each $(\alpha, \gamma) \in \Delta_\varrho$ and each $x \in tvars(\mathcal{U}) \cup rvars(\mathcal{U})$, we have

$$x(\alpha, \gamma)_{\Phi_\theta} = x\varrho(\alpha, \gamma)_{\Phi_\theta}.$$

Thus, for each $(t \stackrel{?}{=} t') \in \mathcal{U}$, we have

$$\begin{aligned} t(\alpha, \gamma)_{\Phi_\theta} &= t\varrho(\alpha, \gamma)_{\Phi_\theta} \\ &= t'\varrho(\alpha, \gamma)_{\Phi_\theta} \\ &= t'(\alpha, \gamma)_{\Phi_\theta}, \end{aligned}$$

with the second equality following from the fact that ϱ is a classical unifier for $\mathcal{U}$. We conclude that each $(\alpha, \gamma) \in \Delta_\varrho$ is a Φ_θ-solution of $\mathcal{U}$, and because $rvars(\mathcal{U}_\varrho) = rvars(\mathcal{U})$, it follows that it is also a (Φ_θ, D_θ)-solution of $\mathcal{U}$.

Completeness. Let (α', γ') be a (Φ_θ, D_θ)-solution of $\mathcal{U}$. We denote the set $rvars(\mathcal{U}_{\varrho,L})$ of recipe variables occurring in the left-side of $\mathcal{U}_\varrho$ by $RV(L)$, and define

$$TV(L) = tvars(\mathcal{U}_{\varrho,L}) \qquad \text{and} \qquad TV(R) = tvars(\mathcal{U}_{\varrho,R})$$

be the sets of term variables occurring in the left and right sides of $\mathcal{U}_\varrho$, respectively. By our assumptions on the algorithm Unif, we have $TV(L) \cap TV(R) = \emptyset$ and no recipe variables occur in the right-hand side of $\mathcal{U}_\varrho$. Furthermore, observe that $RV(L) = rvars(\mathcal{U})$ and $TV(L) = tvars(\mathcal{U})$.

Since $(\alpha', \gamma')_{\Phi_\theta}$ is a classical solution of $\mathcal{U}$, there must be some term substitution α such that

$$(\alpha', \gamma')_{\Phi_\theta} \mid_{\mathcal{U}} = \alpha \circ \varrho \mid_{\mathcal{U}};$$

thus, we have $x(\alpha', \gamma')_{\Phi_\theta} = x\varrho\alpha$ for all $x \in TV(L) \cup RV(L)$. Let us consider the $(\Phi_\theta, \mathcal{X}_R, \mathcal{X}_T)$-substitution (α_*, γ_*) given by

$$(\alpha_*, \gamma_*) = (\alpha' \mid_{TV(L)} \cup \alpha \mid_{TV(R)}, \gamma').$$

For each $x \in tvars(\mathcal{U}) \cup rvars(\mathcal{U}) = TV(L) \cup RV(L)$, we have

$$x(\alpha_*, \gamma_*)_{\Phi_\theta} = x(\alpha', \gamma')_{\Phi_\theta} = x\varrho\alpha = x\varrho(\alpha_*, \gamma_*)_{\Phi_\theta},$$

where the last equality follows from the facts that

$$tvars(x\varrho) \subseteq TV(R)$$

and

$$rvars(\varrho[TV(L) \cup RV(L)]) = \emptyset.$$

Thus, (α_*, γ_*) is a (Φ_θ, D_θ)-solution of $\mathcal{U}_\varrho$.

Since Δ_ϱ is a complete set of Φ_θ-solutions of $\mathcal{U}_\varrho$, there must be some $(\alpha, \gamma) \in \Delta_\varrho$ such that

$$(\alpha, \gamma) \preceq^{\mathcal{U}_\varrho, \Phi_\theta} (\alpha_*, \gamma_*).$$

Since

$$tvars(\mathcal{U}) \subseteq tvars(\mathcal{U}_\varrho) \qquad \text{and} \qquad rvars(\mathcal{U}) \subseteq rvars(\mathcal{U}_\varrho),$$

we also have

$$(\alpha, \gamma) \preceq^{\mathcal{U}, \Phi_\theta} (\alpha_*, \gamma_*).$$

Now, we observe that

$$\alpha_* \mid_{\mathcal{U}} = \alpha' \mid_{\mathcal{U}} \qquad \text{and} \qquad \gamma_* \mid_{\mathcal{U}} = \gamma' \mid_{\mathcal{U}} .$$

Combining these results, it follows that

$$(\alpha, \gamma) \preceq^{\mathcal{U}, \Phi_\theta} (\alpha', \gamma').$$

Thus, Δ_ϱ is a complete set of (Φ_θ, D_θ)-solutions of $\mathcal{U}$.

□

Proof (Lemma 6)**.** Let $(t \stackrel{?}{=} t') \in \mathcal{U}$. If $t \in \mathcal{X}_T$, then

$$t(\alpha, \gamma)_{\Phi_\theta} = t' = t'(\alpha, \gamma)_{\Phi_\theta}.$$

Otherwise, we have $t \in \mathcal{X}_R$, and

$$t(\alpha, \gamma)_{\Phi_\theta} = t\gamma\Phi_\theta = t'\alpha_P.$$

Thus, (α, γ) is a Φ_θ-solution of $\mathcal{U}$. Each $\rho \in dom(D_\theta)$ occurs at most once in $\mathcal{U}$ and is mapped by γ to a recipe variable. Therefore, (α, γ) is also a (Φ_θ, D_θ)-solution of $\mathcal{U}$.

Completeness. Suppose that (α', γ') is a (Φ_θ, D_θ)-solution of $\mathcal{U}$. Consider a recipe substitution

$$\gamma'_* \colon \iota[rvars(\mathcal{U}) \setminus dom(D_\theta)] \to T(\Phi_\theta)$$

such that, for each $x \in dom(\gamma'_*)$, $x\iota\gamma'_* = \rho\gamma'$ for some (arbitrarily chosen) ρ such that $(\rho \stackrel{?}{=} x) \in \mathcal{U}$. We define $\gamma_* = \gamma'_* \cup \theta_{D_\theta,\gamma'}$.

For each $\rho \in rvars(\mathcal{U})$, there is $x \in tvars(\mathcal{U}_{\mathcal{X}_R})$ such that $(\rho \stackrel{?}{=} x) \in \mathcal{U}$, and we have

$$\begin{aligned} \rho\gamma'\Phi_\theta &= x\iota\gamma'_*\Phi_\theta \\ &= \rho\gamma\gamma_*\Phi_\theta. \end{aligned}$$

In other words, there exists a position $p = \epsilon \in pos(\rho\gamma)$ such that $p \in pos(\rho\gamma')$ and

$$\rho\gamma' \mid_p \Phi_\theta = \rho\gamma \mid_p \gamma_*\Phi_\theta. \tag{B.1}$$

Define

$$\alpha_* = \alpha' \mid_{\mathcal{U}_{\mathcal{X}_T}} \cup(\Phi_\theta \circ \gamma_* \circ \iota).$$

Suppose that $x \in tvars(\mathcal{U}_R)$. If $x \in tvars(\mathcal{U}_{\mathcal{X}_R})$, then there is ρ such that $(\rho \stackrel{?}{=} x) \in \mathcal{U}$, and

$$\begin{aligned} x\alpha' &= \rho\gamma'\Phi_\theta \\ &= \rho\gamma\gamma_*\Phi_\theta \\ &= x\iota\gamma_*\Phi_\theta \\ &= x\alpha_*. \end{aligned} \tag{B.2}$$

If $x \in tvars(\mathcal{U}_{\mathcal{X}_T,R}) \setminus \mathcal{U}_{\mathcal{X}_R}$, then trivially

$$\begin{aligned} x\alpha' &= x\alpha\alpha' \\ &= x\alpha\alpha_* \\ &= x\alpha(\alpha_*, \gamma_*)_{\Phi_\theta}. \end{aligned} \tag{B.3}$$

Finally, for all $x \in tvars(\mathcal{U}_L)$, there is t such that $(x \stackrel{?}{=} t) \in \mathcal{U}$. Since $rvars(t) = \emptyset$, we have

$$x\alpha = t(\alpha, \gamma)_{\Phi_\theta} = t\alpha;$$

therefore,

$$\begin{aligned} x\alpha' &= t\alpha' \\ &= t\alpha\alpha_* \\ &= x\alpha\alpha_*, \end{aligned} \tag{B.4}$$

where the second equality follows from equations (B.2) and (B.3).

From equations (B.2), (B.3) and (B.4), we conclude that

$$\alpha' \mid_{\mathcal{U}} = (\alpha_* \circ \alpha) \mid_{\mathcal{U}} .$$

Combining this equation with (B.1), we conclude that

$$(\alpha, \gamma) \preceq^{\mathcal{U},\Phi_\theta}_{(\alpha_*,\gamma_*)} (\alpha', \gamma')$$

(with $\Phi_\theta \circ \gamma_*$ as the replacement function); thus, $\{(\alpha, \gamma)\}$ is a complete set of (Φ_θ, D_θ)-solutions of $\mathcal{U}$.

□

Proof (Theorem 4)**.** The proof is by induction on the depth of recursive calls to the algorithm. The correction in the base case follows from Lemma 6. If $\mathcal{U}$ is not in linear-left form, then the correction follows from the induction hypothesis and Lemma 5.

Suppose then that $\mathcal{U}$ is in linear-left form, but not in solved form. Then, there must be $(\rho \stackrel{?}{=} t) \in \mathcal{U}$ such that $\rho \in \mathcal{X}_R$ and $t \notin \mathcal{X}_T$, so that choosing $(\rho \stackrel{?}{=} t) \in \mathcal{U}$ in line 7 is possible.

$\Delta_\mathcal{U}$ is set of (Φ_θ, D_θ)-solutions. We first prove that the set

$$\Delta_\mathcal{U} = \Delta^a_\mathcal{U} \cup \left(\bigcup_{h \in dom(\Phi_\theta)} \Delta^h_\mathcal{U} \right)$$

is a set of (Φ_θ, D_θ)-solutions of $\mathcal{U}$.

Suppose first that $(\alpha, \gamma) \in \Delta^h_\mathcal{U}$ for some $h \in dom(\Phi_\theta)$. Then, there is $(\alpha', \gamma') \in \Delta_h$ such that $\alpha = \alpha'$ and $\gamma = \gamma' \circ \{\rho \mapsto (h, \iota)\}$ for some injective substitution ι as in the definition of the algorithm. By the induction hypothesis, (α', γ') is a $(\Phi_{\theta'}, D_{\theta'})$-solution of $\mathcal{U}^h$. If $\rho \notin dom(D)$, then $\theta' = \theta$. Otherwise, because γ' is a $(\Phi_{\theta'}, D_{\theta'})$-recipe substitution with $\theta' = \{\rho \mapsto (h, \iota)\} \circ \theta$, we must have $\rho \notin rvars(h\Phi_\theta\iota)$, and

$$\Phi_\theta \mid_{\mathcal{X}_R \setminus \{\rho\}} = \Phi_{\theta'} \mid_{\mathcal{X}_R \setminus \{\rho\}} .$$

In both cases, we conclude that

$$h\Phi_\theta\iota\gamma'\Phi_\theta = h\Phi_\theta\iota\gamma'\Phi_{\theta'}.$$

Using this fact, we have

$$\begin{aligned}
\rho(\alpha, \gamma)_{\Phi_\theta} &= (h, \iota)\Phi_\theta(\Phi_\theta \circ \gamma') \\
&= h\Phi_\theta\iota\gamma'\Phi_\theta \\
&= h\Phi_\theta\iota\gamma'\Phi_{\theta'} \\
&= h\Phi_\theta\iota(\alpha', \gamma')\Phi_{\theta'} \\
&= t(\alpha', \gamma')\Phi_{\theta'} \\
&= t(\alpha, \gamma)\Phi_\theta.
\end{aligned}$$

where the fifth equality uses the fact that $(h\Phi_\theta\iota \stackrel{?}{=} t) \in \mathcal{U}^h$ and (α', γ') is a $(\Phi_{\theta'}, D_{\theta'})$-solution of $\mathcal{U}^h$, and the last equality uses the fact that $\rho \notin rvars(t)$ (because $\mathcal{U}$ is in linear-left form). Moreover, for all $(t \stackrel{?}{=} t') \in \mathcal{U} \setminus \left\{(\rho \stackrel{?}{=} t)\right\}$, we have that

$$(t \stackrel{?}{=} t') \in \mathcal{U}^h \qquad \text{and} \qquad t(\alpha', \gamma')\Phi_{\theta'} = t'(\alpha', \gamma')\Phi_{\theta'}.$$

We have $\rho \notin rvars(t) \cup rvars(t')$, because $\mathcal{U}$ is in linear-left form, and, for all recipe variables $\rho' \neq \rho$,

$$\begin{aligned}\rho'\gamma\Phi_\theta &= \rho'\gamma\left\{\rho \mapsto (h,\iota)\right\}\Phi_{\theta'}\\ &= \rho'\gamma'\Phi_{\theta'}.\end{aligned}$$

Combining these results, we obtain

$$\begin{aligned}t(\alpha,\gamma)\Phi_\theta &= t(\alpha',\gamma')\Phi_{\theta'}\\ &= t'(\alpha',\gamma')\Phi_{\theta'}\\ &= t'(\alpha,\gamma)\Phi_\theta.\end{aligned}$$

This shows that (α,γ) is a Φ_θ-solution of $\mathcal{U}^h$.

It remains to prove that γ is a (Φ_θ, D_θ)-recipe substitution. For $\rho_D \in dom(D_\theta)$, we have

$$\gamma[\rho_D] = (\gamma' \circ \{\rho \mapsto (h,\iota)\})[\rho_D].$$

Now, γ' is a $(\Phi_{\theta'}, D_{\theta'})$-recipe substitution, and $\rho \notin rvars(ran(\Phi_{\theta'}))$. We have two cases: either $\rho_D = \rho$ or not. If $\rho_D \neq \rho$, we have $\gamma[\rho_D] = \{\gamma'[\rho_D]\}$. Since γ' is a $(\Phi_{\theta'}, D_{\theta'})$-recipe substitution and $D_{\theta'}(\rho) = D_\theta(\rho)$, it follows that $\gamma[\rho_D]$ is a set with a single element in $T(\Phi_\theta \mid_{D_\theta(\rho_D)})$. In the second case, i.e., $\rho_D = \rho$, we have $\rho \in dom(D_\theta)$, and thus, for all $\rho' \in ran(\iota)$, $\rho' \in dom(D_{\theta'})$ and $D_{\theta'}(\rho') = D_\theta(\rho)$. Because γ' is a $(\Phi_{\theta'}, D_{\theta'})$-recipe substitution, it follows that there exists a $(\Phi_{\theta'}, D_{\theta'})$-recipe $\zeta_{\rho',\gamma'}$ such that

$$\gamma'[\rho'] = \left\{\zeta_{\rho',\gamma'}\right\} \qquad \text{and} \qquad \zeta'_{\rho',\gamma'} \in T(\Phi_{\theta'} \mid_{D_{\theta'}} (\rho')).$$

Since $D_{\theta'}(\rho') = D_\theta(\rho)$ and $h \in D_\theta(\rho)$, we also have

$$(h, \left\{\rho' \mapsto \zeta_{\rho',\gamma'} \mid \rho' \in ran(\iota)\right\} \circ \iota) \in T(\Phi_\theta \mid_{D_\theta(\rho)}).$$

Thus, we have that

$$\gamma[\rho] = \left\{h\Phi_\theta\iota\left\{\rho' \mapsto \zeta_{\rho',\gamma'} \mid \rho' \in ran(\iota)\right\}\right\}$$

is a set with a single element contained in $T(\Phi_\theta \mid_{D_\theta(\rho)})$. This concludes the proof that γ is a (Φ_θ, D_θ)-recipe substitution.

It remains to consider the case that $(\alpha,\gamma) \in \Delta^a_{\mathcal{U}}$, i.e., there is $(\alpha',\gamma') \in \Delta^a$ of $\mathcal{U}^a$ such that $\gamma = \gamma' \cup \{\rho \mapsto t\}$ and $\alpha' = \alpha$. By the induction hypothesis, (α',γ') is a $(\Phi_{\theta'}, D_{\theta'})$-solution of $\mathcal{U}^a$. We have

$$\rho(\alpha,\gamma)_{\Phi_\theta} = t = t(\alpha,\gamma)_{\Phi_\theta},$$

with the last equality following from the fact that $t \in \mathcal{N}$. Moreover, for all $(s' \stackrel{?}{=} t') \in \mathcal{U} \setminus \left\{(\rho \stackrel{?}{=} t)\right\}$, we have $(s' \stackrel{?}{=} t') \in \mathcal{U}^a$, and, as above,

$$\begin{aligned}s'(\alpha,\gamma)_\Phi &= s'(\alpha',\gamma')_\Phi\\ &= t'(\alpha',\gamma')_\Phi\\ &= t'(\alpha,\gamma)_\Phi,\end{aligned}$$

because $(\alpha', \gamma')_\Phi$ coincides with $(\alpha, \gamma)_\Phi$ for all term variables and recipe variables except ρ, which does not occur in $\{s', t'\}$ because $\mathcal{U}$ is in linear-left form. Thus, (α, γ) is a Φ-solution of $\mathcal{U}$. Since (α', γ') is a (Φ^a, D^a)-solution of $\mathcal{U}^a$, $D^a = D$, $\rho \notin dom(D)$, and γ coincides with γ' for all recipe variables except ρ, it follows that (α, γ) is also a (Φ, D)-solution of $\mathcal{U}$.

We conclude that $\Delta_\mathcal{U}$ is a set of (Φ_θ, D_θ)-solutions of $\mathcal{U}$.

$\Delta_\mathcal{U}$ is complete. It remains to prove that $\Delta_\mathcal{U}$ complete. To prove this, let (α', γ') be a (Φ_θ, D_θ)-solution of $\mathcal{U}$. We need to prove that there exists $(\alpha, \gamma) \in \Delta_\mathcal{U}$ such that $(\alpha, \gamma) \preceq^{\mathcal{U}, \Phi_\theta} (\alpha', \gamma')$. Let us consider $\rho\gamma'$. Since $t \notin \mathcal{X}_T \cup \mathcal{X}_R$, we cannot have $t(\alpha', \gamma')_\Phi \in \mathcal{X}_R$; thus, we must also have $\rho\gamma' \notin \mathcal{X}_R$.

Suppose first that $\rho\gamma' = t$ for some $t \in \mathcal{N} \setminus \tilde{n}$. Then, we have

$$\begin{aligned} a &= \rho(\alpha', \gamma')_{\Phi_\theta} \\ &= t(\alpha', \gamma')_{\Phi_\theta} \\ &= t. \end{aligned}$$

We have $\mathcal{U}^a = \mathcal{U} \setminus \{(\rho, t)\}$ and $\theta' = \{\rho \mapsto t\} \circ \theta$; thus, it is clear that (α', γ') is also a $(\Phi_{\theta'}, D_{\theta'})$-solution of $\mathcal{U}^a$. By the induction hypothesis, Δ_a is a complete set of $(\Phi^a_{\theta'}, D_{\theta'})$-solutions of $\mathcal{U}^a$; thus, there is $(\alpha'', \gamma'') \in \Delta_a$ such that

$$(\alpha'', \gamma'') \prec^{\mathcal{U}^a, \Phi_{\theta'}} (\alpha', \gamma').$$

Taking

$$(\alpha, \gamma) = (\alpha'', \gamma'' \cup \{\rho \mapsto t\}),$$

we have $(\alpha, \gamma) \in \Delta^a_\mathcal{U}$ and $(\alpha, \gamma) \preceq^{\mathcal{U}, \Phi_\theta} (\alpha', \gamma')$.

It remains to consider the case that $\rho\gamma' = (h, \delta)$ for some $h \in dom(\Phi_\theta)$ and some (Φ_θ, D_θ)-recipe substitution δ. Define γ'' as follows:

$$\rho'\gamma'' = \begin{cases} \rho'\delta & \text{if } \rho' = \rho''\iota \text{ for some } \rho'' \in rvars(h\Phi_\theta) \\ \rho'\gamma'' & \text{if } \rho' \in dom(D_\theta) \cup rvars(\mathcal{U}_R). \end{cases}.$$

Then,

$$\begin{aligned} \rho(\alpha', \gamma')_{\Phi_\theta} &= (h, \delta)\Phi_\theta \\ &= (h\Phi_\theta)(\Phi_\theta \circ \delta) \\ &= (h\Phi_\theta\iota)(\Phi_{\theta'} \circ \gamma'') \\ &= (h\Phi_\theta\iota)(\alpha', \gamma'')_{\Phi_{\theta'}}, \end{aligned}$$

with $\theta' = \{\rho \mapsto (h, \emptyset)\} \circ \theta$ if $\rho \in dom(D_\theta)$ and $\theta' = \theta$ otherwise.

Now, $(\alpha', \gamma'')_{\Phi_{\theta'}}$ coincides with $(\alpha', \gamma')_{\Phi_\theta}$ for all term variables and recipe variables except possibly ρ and those in $ran(\iota)$. Since t only contains term variables, we obtain

$$\begin{aligned} (h\Phi_{\theta'}\iota)(\alpha', \gamma'')_{\Phi_{\theta'}} &= \rho(\alpha', \gamma')_{\Phi_\theta} \\ &= t(\alpha', \gamma')_{\Phi_\theta} \\ &= t(\alpha', \gamma'')_{\Phi_{\theta'}}. \end{aligned}$$

Since ι is chosen so that $ran(\iota) \cap rvars(\mathcal{U}) = \emptyset$ and $\mathcal{U}$ is in linear-left form, we also conclude that, for all $(s' \stackrel{?}{=} t') \in \mathcal{U} \setminus \left\{\rho \stackrel{?}{=} t\right\}$, we have

$$\begin{aligned} s'(\alpha, \gamma'')_{\Phi_{\theta'}} &= s'(\alpha', \gamma')_{\Phi_\theta} \\ &= t'(\alpha', \gamma')_{\Phi_\theta} \\ &= t'(\alpha, \gamma'')_{\Phi_{\theta'}}. \end{aligned}$$

Thus, (α', γ'') is a $(\Phi_{\theta'}, D_{\theta'})$-solution of $\mathcal{U}^h$. By the induction hypothesis, there is a $(\Phi_{\theta'}, D_{\theta'})$-solution $(\alpha^h, \gamma^h) \in \Delta^h$ of $\mathcal{U}^h$ such that

$$(\alpha^h, \gamma^h) \prec^{\mathcal{U}^h, \Phi_{\theta'}} (\alpha', \gamma'').$$

Let

$$(\alpha, \gamma) = (\alpha^h, \gamma^h \circ \{\rho \mapsto (h, \iota)\}).$$

We have $(\alpha, \gamma) \in \Delta^h_{\mathcal{U}}$. There is α_* such that $\alpha_* \circ \alpha^h = \alpha'$ and

$$\gamma_* \colon vars(rvars(ran(\gamma^h))) \to T_\Sigma(\mathcal{N} \cup \mathcal{X}_R)$$

such that, for each $\rho' \in dom(\gamma^h)$ and each position $p \in pos(\rho'\gamma^h)$, we have $p \in pos(\rho'\gamma'')$ and either

$$head(\rho'\gamma^h \mid_p) = head(\rho'\gamma'' \mid_p)$$

or

$$\rho'\gamma'' \mid_p \Phi_{\theta'} = \rho'\gamma^h \mid_p \gamma_*.$$

It is simple to check that γ and γ' are related by γ_* in the same way as γ^h and γ'' since $\gamma = \gamma^h$ and $\gamma'' = \gamma'$ if $\rho' \neq \rho$, and $\rho\gamma = (h, \iota)\gamma^h$ and $\rho\gamma' = (h, \iota)\gamma''$. We conclude that

$$(\alpha, \gamma) \preceq^{\mathcal{U}, \Phi_{\theta'}} (\alpha', \gamma').$$

Thus, Algorithm 5 outputs a complete set of Φ_θ-solutions of $\mathcal{U}$.

□

B.2 Saturation Algorithm

Proof (Theorem 5)**.** We first prove that the following properties are invariants of Algorithm 6 when executed on input (Φ, D):

- $dom(\Theta_F) = dom(\Theta_T)$ is a set of D-bindings of Φ;
- for all $\theta \in dom(\Theta_F)$, $(\Phi_\theta)\!\downarrow \subseteq \Theta_F(\theta)$;
- for all $\theta \in dom(\Theta_F)$, $\Theta_T(\theta) \colon dom(\Theta_F(\theta)) \to T(\Phi, \theta)$ is a Φ-translation for $\Theta_T(\theta)$.

This proves that the first requirement of the definition of a saturation is satisfied by the output of the Algorithm whenever it terminates.

The functions Θ_T and Θ_F are initialized together in lines 2—3, and then always updated together, in lines 17—24, and 23—24. We prove that their initialization establishes the invariants and that each update preserves them.

Initialization. After the initialization in lines 2 and 3, we have

$$dom(\Theta_F) = dom(\Theta_T) = \{id(dom(D))\},$$

and $id(dom(D))$ is trivially a D-binding. We have

$$\Theta_F(id(dom(D))) = \Phi\downarrow = \Phi_{id(dom(D))}\downarrow .$$

Finally, for all $h \in dom(D)$,

$$h\Theta_T(id(dom(D))) = (h, \emptyset) \in T(\Phi, id(dom(D))),$$

and

$$h\Theta_T(id(dom(D)))\Phi = h\Phi = h\Theta_F(id(dom(D))).$$

Thus, the initialization of Θ_F and Θ_T establishes all three invariants.

Update in lines 17-18. This update does not change the domains of Θ_F or Θ_T, and thus the first invariant is trivially preserved. It extends the function $\Theta_F(\theta')$, and thus it preserves the second invariant. To prove the preservation of the third invariant it is sufficient to prove:

(1) $h_*\Theta_T(\theta') \in T(\Phi, \theta')$, and

(2) $h_*\Theta_T(\theta')\Phi \approx_{\mathcal{R}} h_*\Theta_F(\theta')$.

Property (1) holds since, by the induction hypothesis, $\Theta_T(\theta')((h, \gamma)) \in T(\Phi, \theta')$. Similarly, using the induction hypothesis to prove property (2), we have

$$\begin{aligned} h_*\Theta_T(\theta')\Phi &\approx_{\mathcal{R}} (h, \gamma)\Theta_T(\theta')\Phi \\ &\approx_{\mathcal{R}} (h, \gamma)\Theta_F(\theta') \\ &\approx_{\mathcal{R}} t \\ &= h_*\Theta_F(\theta'). \end{aligned}$$

Update in lines 23-24. Since γ is a $(\Theta_F(\theta), \Theta_T(\theta)(D_\theta))$-recipe substitution, we have that $\theta_{\Theta_F(\theta)(D_\theta),\gamma}$ is a $\Theta_F(\theta)(D_\theta)$-binding for $\Theta_F(\theta)$; therefore,

$$\Theta_T(\theta)(\theta_{\Theta_F(\theta)(D_\theta),\gamma})$$

is a D_θ-binding for Φ_θ. Because θ is a D-binding of Φ, it follows that

$$\theta' = \Theta_T(\theta)(\theta_{\Theta_F(\theta)(D_\theta),\gamma}) \circ \theta$$

is also a D-binding of Φ. The domain of both Θ_F and Θ_T are extended by adding θ' to them, and thus they remain equal. This proves the preservation of the first invariant.

To prove that the second invariant is preserved we must prove that $\Phi_{\theta'}\downarrow\subseteq \Theta_F(\theta')$. We have

$$\begin{aligned} dom(D_\theta) &= dom(\Theta_T(\theta)(D_\theta)) \\ &= dom(\theta_{\Theta_T(\theta)(D_\theta),\gamma}) \end{aligned}$$

and, for each $\rho \in dom(D_\theta)$, the induction hypothesis implies that

$$\rho\theta_{\Theta_T(\theta)(D_\theta),\gamma}\Theta_F(\theta) \approx_{\mathcal{R}} \rho\theta_{\Theta_T(\theta)(D_\theta),\gamma}\Theta_T(\theta)\Phi,$$

i.e.,

$$\Theta_F(\theta) \circ \theta_{\Theta_T(\theta)(D_\theta),\gamma} \approx_{\mathcal{R}} \Phi \circ \Theta_T(\theta) \circ \theta_{\Theta_T(\theta)(D_\theta),\gamma}.$$

Now, for each $h \in dom(\Phi)$, we have

$$\begin{aligned} h\Phi_{\theta'} &= h\Phi_{\theta\circ(\Theta_T(\theta)\circ\theta_{\Theta_T(\theta)(D_\theta),\gamma})} \\ &= h\Phi_\theta(\Phi \circ \Theta_T(\theta) \circ \theta_{\Theta_T(\theta)(D_\theta),\gamma}) \\ &\approx_{\mathcal{R}} h\Theta_F(\theta)(\Theta_F(\theta) \circ \theta_{\Theta_T(\theta)(D_\theta),\gamma}) \\ &\approx_{\mathcal{R}} h\Theta_F(\theta'). \end{aligned}$$

Since line 23 enforces that $h\Theta_F(\theta')$ is in normal form, it follows that

$$h\Phi_{\theta'}\downarrow = h\Theta_F(\theta'),$$

Thus, $\Phi_{\theta'}\downarrow\subseteq \Theta_F(\theta')$, and the second invariant is preserved.

To prove the preservation of the third invariant, we must prove, for each $h \in dom(\Theta_F(\theta'))$:

(1) $h\Theta_T(\theta') \in T(\Phi, \theta')$, and

(2) $h\Theta_T(\theta')\Phi \approx_{\mathcal{R}} h\Theta_F(\theta')$.

Suppose first that $h \in dom(\Theta_F(\theta')) \setminus \{h_*\}$. Then, we have

$$\begin{aligned} h\Theta_T(\theta') &= hT' \\ &= h\Theta_T(\theta)(\Theta_T(\theta) \circ \theta_{\Theta_T(\theta)(D_\theta),\gamma}). \end{aligned}$$

For each $\rho \in dom(D)$, we have

$$\rho\theta' = \rho\theta(\Theta_T(\theta) \circ \theta_{\Theta_T(\theta)(D_\theta),\gamma}).$$

Furthermore,

$$h\Theta_T(\theta)[\rho] = \{\rho\theta\};$$

thus, it follows that

$$h\Theta_T(\theta)(\Theta_T(\theta) \circ \theta_{\Theta_T(\theta)(D_\theta),\gamma}) = \left\{\rho\theta(\Theta_T(\theta) \circ \theta_{\Theta_T(\theta)(D_\theta),\gamma})\right\} = \left\{\rho\theta'\right\}.$$

The induction hypothesis implies that $h\Theta_T(\theta')$ is a $T(\Phi)$-recipe. Therefore, we have $h\Theta_F(\theta') \in T(\Phi, \theta')$, proving (1).

To prove (2), we compute:

$$\begin{aligned} h\Theta_T(\theta')\Phi &= h\Theta_T(\theta)(\Theta_T(\theta) \circ \theta_{\Theta_T(\theta)(D_\theta),\gamma})\Phi \\ &= h\Theta_T(\theta)\Phi(\Phi \circ \Theta_T(\theta) \circ \theta_{\Theta_T(\theta)(D_\theta),\gamma}) \\ &\approx_{\mathcal{R}} h\Theta_F(\theta)(\Theta_F(\theta) \circ \theta_{\Theta_T(\theta)(D_\theta),\gamma}) \\ &\approx_{\mathcal{R}} hF' \\ &= h\Theta_F(\theta'). \end{aligned}$$

Let us then consider the case that $h = h_*$. We have seen above that $hT'\Phi \approx_{\mathcal{R}} h\Theta_F(\theta)$ for all $h \in dom(\Theta_F(\theta))$. Thus,

$$\begin{aligned} h_*\Theta_T(\theta')\Phi &= (h,\gamma)T'\Phi \\ &\approx_{\mathcal{R}} (h,\gamma)\Theta_F(\theta) \\ &\approx_{\mathcal{R}} t \\ &= h_*\Theta_F(\theta'). \end{aligned}$$

Therefore, the third invariant is preserved.

Saturation. Now we prove the second requirement of the definition of saturation.

For each term t, denote by $|t|_{\mathcal{R}}$ the length of the longest $\mathcal{R}$-rewriting sequence starting from t. We begin by proving that, for each D-binding θ' of Φ and each $n \in \mathbb{N}$, there exist $\theta_n \in dom(\Theta_F)$ and a $(\Theta_F(\theta_n), \Theta_T(\theta_n)(D_{\theta_n}))$-recipe substitution γ_n satisfying the two properties to prove, i.e.:

- $\theta_n\gamma_n\Theta_F(\theta_n) = \theta'\Phi$;
- for all recipes $\zeta' \in T(\Phi_{\theta'})$, there exists a recipe

$$\zeta \in T(\Theta_F(\theta_n), id(dom(D_{\theta_n})))$$

  satisfying

$$\zeta\gamma_n\Theta_F(\theta_n) = (\zeta'\Phi_{\theta_n})\downarrow .$$

This proof is by induction on $|\zeta'\Phi_{\theta'}|_{\mathcal{R}}$.

For the base case, suppose that $|\zeta'\Phi_{\theta'}|_{\mathcal{R}} = 0$, i.e., $\zeta'\Phi_{\theta'}$ is in normal form. Take $\theta_0 = id(dom(D))$ and $\gamma_0 = \theta'$. We have $\theta_0 \in dom(\Theta_F)$ (by the initialization of Θ_F in line 2). Moreover, $D_{\theta_0} = D$, $\Theta_F(\theta_0) = \Phi\downarrow$, and $\Theta_T(\theta_0) = id(dom(\Phi))$; thus, γ_0 is a $(\Theta_F(\theta_0), \Theta_F(\theta_0)(D_{\theta_0}))$-recipe substitution, and

$$\theta_0\gamma_0\Theta_F(\theta_0) \approx_{\mathcal{R}} \theta'\Phi.$$

Letting ζ_* be the Φ-recipe such that $\zeta_*\theta' = \zeta'$, we have

$$\begin{aligned} \zeta_*\gamma_0\Theta_F(\theta_0) &= \zeta_*\theta'(\Phi\downarrow) \\ &= \zeta'\Phi = (\zeta'\Phi)\downarrow, \end{aligned}$$

using the hypothesis that $\zeta'\Phi$ is in normal form. Thus, the result is proved by taking $\zeta = \zeta_*$.

Let $\zeta'_{n,1}, \ldots, \zeta'_{n,i}, \ldots$ be a sequence of $\Phi_{\theta'}$-recipes such that $|\zeta'_{n,j}|_{\mathcal{R}} = n$ for all $j \in \mathbb{N}$ and $\left\{\zeta'_{n,j} \mid j \in \mathbb{N}\right\}$ contains precisely the $\Phi_{\theta'}$-recipes ζ' satisfying $|\zeta'\Phi_{\theta'}|_{\mathcal{R}} = n$.

Suppose then that the result is valid for all integers up to $n-1$. We prove that, for all $k \in \mathbb{N}$, there exist:

(1) $\theta_{n,k} \in dom(\Theta_F)$, and

(2) a $(\Theta_F(\theta_{n,k}), \Theta_T(\theta_{n,k})(D_{\theta_{n,k}}))$-recipe substitution $\gamma_{n,k}$ such that, whenever $\zeta' \in T(\Phi_{\theta'})$ satisfies

$$|\zeta'\Phi_{\theta'}|_{\mathcal{R}} \leq n \qquad \text{or} \qquad \zeta' \in \left\{\zeta'_1, \ldots, \zeta'_k\right\},$$

there exists $\zeta \in T(\Theta_F(\theta_{n,k}), id(dom(D_{\theta_{n,k}})))$ satisfying

$$\zeta\gamma_{n,k}\Theta_F(\theta_{n,k}) = (\zeta'\Phi_{\theta'})\downarrow .$$

For $k = 0$ it is sufficient to take $\theta_{n,0} = \theta_{n-1}$ and $\gamma_{n,0} = \gamma_{n-1}$.

Then, suppose that this property is valid for all integers up to $k-1$. Let $h_{n,k}, \zeta'_1, \ldots, \zeta'_m$ be such that

$$\zeta'_{n,k} = h_{n,k}(\zeta'_1 \ldots, \zeta'_m).$$

Let $\rho_1, \ldots, \rho_m$ be such that

$$\{\rho_1, \ldots, \rho_m\} = rvars(h_k\Phi_{\theta'}) \qquad \text{and} \qquad \rho_1 \prec_{\mathcal{X}_R} \ldots \prec_{\mathcal{X}_R} \rho_k,$$

and let $\delta_{n,k}$ be the substitution given by

$$\delta_{n,k} = \left\{\rho_j \mapsto (\zeta'_j\Phi_{\theta'})\downarrow \mid j \in \{1, \ldots, m\}\right\}.$$

We consider first the case that $h_k\Phi_{\theta'}\delta_{n,k}$ is in normal form. Note that this implies that $h_k\Phi_{\theta'}$ is in normal form. It is sufficient to prove that, for each $j \in \{1, \ldots, m\}$, there exists

$$\zeta_j \in T(\Theta_F(\theta_{n,k-1}), id(dom(D_{\theta_{n,k-1}})))$$

such that

$$\zeta_j\gamma_{n,k-1}\Theta_F(\theta_{n,k-1}) = (\zeta'_j\Phi_{\theta'})\downarrow = \rho_j\delta_{n,k}.$$

In this case, it is sufficient to take $\zeta = h_k(\zeta_1, \ldots, \zeta_m)$, since

$$\begin{aligned}
&h_k(\zeta_1, \ldots, \zeta_m)\gamma'_{n,k}\Theta_F(\theta'_{n,k}) \\
&= h_k\Theta_F(\theta'_{n,k})\delta_{n,k} \\
&= (h_k\Phi_{\theta'})\downarrow \delta_{n,k} \\
&= h_k\Phi_{\theta'}\delta_{n,k}.
\end{aligned}$$

If $|\zeta'_j\Phi_{\theta'}|_{\mathcal{R}} < n$ for all $j \in \{1,\ldots,m\}$, then the result follows from the induction hypothesis. Otherwise, there exists $i \in \{1,\ldots,m\}$ such that $|\zeta'_i\Phi_{\theta'}|_{\mathcal{R}} = n$, and it is sufficient to prove that there exist $\theta'_{n,k}$, $\gamma'_{n,k}$ and ζ_i satisfying properties (1) and (2) when $\zeta' = \zeta'_i$. Iterating this reasoning, we are reduced to the problem of proving the result when $h_k\Phi_{\theta'}\delta_{n,k}$ is not in normal form.

Suppose then that this is the case. Then, we have $|\zeta'_j\Phi_{\theta'}|_{\mathcal{R}} < n$ for all $j \in \{1,\ldots,m\}$, and the induction hypothesis implies that there exists

$$\zeta_j \in T(\Theta_F(\theta_{n,k-1}), id(dom(D_{\theta_{n,k-1}})))$$

such that

$$\zeta_j\gamma_{n,k-1}\Theta_F(\theta_{n,k-1}) = (\zeta'_j\Phi_{\theta'})\downarrow .$$

Let

$$\zeta_{n,k} = h_k(\zeta_1,\ldots,\zeta_n).$$

Then, we have

$$\zeta_{n,k} \in T(\Theta_F(\theta_{n,k-1}), id(dom(D_{\theta_{n,k-1}})))$$

and

$$\begin{array}{l} h_k\Phi_{\theta'}\delta_{n,k} \\ \rightarrow^*_{\mathcal{R}} (h_k\Phi_{\theta'}\downarrow)\delta_{n,k} \\ = h_k\Theta_F(\theta_{n,k-1})\delta_{n,k} \\ = h_k(\zeta_1,\ldots,\zeta_m)\Theta_F(\theta_{n,k-1}) \\ = \zeta_{n,k}\gamma_{n,k-1}\Theta_F(\theta_{n,k-1}). \end{array}$$

Now we prove that there exist

- $\theta_{n,k} \in dom(\Theta_F)$,
- a $(\Theta_F(\theta_{n,k}), \Theta_T(\theta_{n,k})(D_{\theta_{n,k}}))$-recipe substitution $\gamma_{n,k}$, and
- a recipe $\zeta \in T(\Theta_F(\theta_{n,k}), id(dom(D_{\theta_{n,k}})))$

such that

$$\begin{array}{l} \theta_{n,k}\gamma_{n,k}\Theta_F(\theta_{n,k}) \\ \approx_{\mathcal{R}} \theta_{n,k-1}\gamma_{n,k-1}\Theta_F(\theta_{n,k-1}) \\ \approx_{\mathcal{R}} \theta'\Phi \end{array}$$

and

$$\begin{array}{l} \zeta\gamma_{n,k}\Theta_F(\theta_{n,k}) = (\zeta_{n,k}\gamma_{n,k-1}\Theta_F(\theta_{n,k-1}))\downarrow \\ = (h_k\Phi_{\theta'}\delta_{n,k})\downarrow \\ = \zeta'_{n,k}\phi_{\theta'}\downarrow . \end{array}$$

This proof is by induction on $q = |\zeta_{n,k}\gamma_{n,k-1}\Theta_F(\theta_{n,k-1})|_{\mathcal{R}}$. If $q = 0$, then $\zeta_{n,k}\gamma_{n,k-1}\Theta_F(\theta_{n,k-1})$ is in normal form, and it is sufficient to take

$$\theta_{n,k} = \theta_{n,k-1}, \qquad \gamma_{n,k} = \gamma_{n,k-1} \qquad \text{and} \qquad \zeta = \zeta_{n,k}.$$

Otherwise, there are

- $h_s \in sub(h_k\Theta_F(\theta_{n,k-1})) \setminus \mathcal{X}_R$,
- $l \in \mathcal{R}_L$,
- a term substitution α,
- and a $(\Theta_F(\theta_{n,k-1}), \Theta_T(\theta_{n,k-1})(D_{\theta_{n,k-1}}))$-recipe substitution γ

such that

$$h_s\gamma = l\alpha.$$

Thus, there exists

$$(\alpha_*, \gamma_*) \in \mathsf{genUnif}(\Theta_F(\theta_{n,k-1}), \Theta_T(\theta_{n,k-1})(D_{\theta_{n,k-1}}), \mathcal{U}),$$

where $\mathcal{U} = \left\{ l \stackrel{?}{=} h_s \right\}$, such that

$$(\alpha_*, \gamma_*) \prec^{\mathcal{U}, \Theta_F(\theta_{n,k-1})} (\alpha, \gamma).$$

Consider the substitution

$$\theta^q_{n,k} = (\Theta_T(\theta_{n,k-1}) \circ \theta_{\Theta_T(\theta_{n,k-1})(D_{\theta_{n,k-1}}),\gamma_*}) \circ \theta_{n,k-1}.$$

Since the algorithm terminates, the loop in lines 5—24 must terminate. Because $\theta_{n,k-1} \in dom(\Theta_F)$, the last execution of the loop computes

$$\mathsf{genUnif}(\Theta_F(\theta_{n,k-1}), \Theta_T(\theta_{n,k-1})(D_{\theta_{n,k-1}}), \mathcal{U}),$$

finding (α_*, γ_*). Thus, we have $\theta^q_{n,k} \in dom(\Theta_F)$, and there exists

$$\zeta^q \in T(\Theta_F(\theta_{n,k-1}), id(dom(D_{\theta_{n,k-1}})))$$

such that

$$\zeta^q\Theta_F(\theta^q_{n,k}) = (h_k, \gamma_*)\Theta_F(\theta_{n,k-1})\downarrow .$$

For all $\rho \in rvars(h_s\Theta_F(\theta_{n,k-1}))$ and all $p \in pos(\rho\gamma_*)$, we have $p \in pos(\rho\gamma)$. Furthermore, there exists a replacement function γ_{**} such that, for all $p \in pos(\rho\gamma_*)$, either:

- $head(\rho\gamma_* \mid_p) = head(\rho\gamma \mid_p)$;
- or $\rho\gamma_* \mid_p \in dom(\gamma_{**})$ and $\rho\gamma \mid_p \Theta_F(\theta_{n,k-1}) = \rho\gamma_{**}$.

Now, for each position p such that the second condition holds, we have

$$\rho\gamma \mid_p \in T(\Theta_F(\theta_{n,k-1}), \Theta_T(\theta_{n,k-1})(D_{\theta_{n,k-1}}))$$

and

$$\rho\gamma \mid_p \Theta_F(\theta_{n,k-1}) = \rho\gamma_{**}.$$

For each such ρ, choose one such position p, and define $\zeta_\rho = \rho\gamma \mid_p$. Consider the $(\Theta_F(\theta_{n,k-1}), \Theta_T(\theta_{n,k-1})(D_{\theta_{n,k-1}}))$-recipe substitution $\gamma^q_{n,k}$ given by $\rho\gamma_{n,k} = \zeta_\rho$ for all such recipe variables ρ. We have

$$\begin{aligned}
&\theta^q_{n,k}\gamma^q_{n,k}\Theta_F(\theta^q_{n,k}) \\
&= \theta^q_{n,k}\gamma^q_{n,k}\Theta_F(\theta_{n,k-1}) \\
&= \theta_{n,k-1}\gamma_{n,k-1}\Theta_F(\theta_{n,k-1}) \\
&\approx_{\mathcal{R}} \theta'\Phi.
\end{aligned}$$

Furthermore, letting h_* be the new element added to the domain of $F' = \Theta_F(\theta^q_{n,k})$ as in the description of the algorithm, we have

$$\begin{aligned}
&h_*\gamma^q_{n,k}\Theta_F(\theta^q_{n,k}) \\
&= h_*\Theta_F(\theta^q_{n,k})(\Theta_F(\theta^q_{n,k}) \circ \gamma^q_{n,k}) \\
&= (\zeta_{n,k}\Theta_F(\theta_{n,k-1}))\downarrow (\Theta_F(\theta^q_{n,k}) \circ \gamma^q_{n,k}).
\end{aligned}$$

Thus,

$$\begin{aligned}
&\zeta_{n,k}\gamma^q_{n,k}\Theta_F(\theta^q_{n,k}) \\
&\zeta_{n,k}\Theta_F(\theta_{n,k-1})(\Theta_F(\theta^q_{n,k}) \circ \gamma^q_{n,k}) \\
&\rightarrow^+_{\mathcal{R}} (\zeta_{n,k}\Theta_F(\theta_{n,k-1}))\downarrow (\Theta_F(\theta^q_{n,k}) \circ \gamma^q_{n,k}) \\
&h_*\gamma^q_{n,k}\Theta_F(\theta^q_{n,k}),
\end{aligned}$$

using the fact that $\zeta_{n,k}\Theta_F(\theta_{n,k-1})$ is not in normal form by hypothesis. We conclude that

$$|h_*\gamma^q_{n,k}\Theta_F(\theta^q_{n,k})|_{\mathcal{R}} < |\zeta_{n,k}\gamma^q_{n,k}\Theta_F(\theta^q_{n,k})|_{\mathcal{R}},$$

and we can use the induction hypothesis to conclude that there exist $\theta_{n,k}$, $\gamma_{n,k}$ and ζ satisfying properties (1) and (2).

We have thus proved that, for all $k \in \mathbb{N}$, there exist

- $\theta_{n,k} \in dom(\Theta_F)$,
- a $(\Theta_F(\theta_{n,k}), \Theta_T(\theta_{n,k})(D_{\theta_{n,k}}))$-recipe substitution $\gamma_{n,k}$, and
- a recipe ζ

such that

$$\theta_{n,k}\gamma_{n,k}\Theta_F(\theta_{n,k}) \approx_r s\theta'\Phi$$

and

$$\zeta\gamma_{n,k}\Theta_F(\theta_{n,k}) = \zeta'_{n,k}\Phi_{\theta'}.$$

Now, each element of the set

$$\{(\theta_{n,k}, \gamma_{n,k}) \mid k \in \mathbb{N}\}$$

corresponds to a different function $\theta_{n,k}$, and adding this function to the domain of Θ_F requires an algorithm operation. Since the algorithm terminates, this set must be finite. Note further that the set of terms whose normal form can be obtained

constructively from the frames $\Theta_F(\theta_{n,k})$ by composing recipes with $\gamma_{n,k}$ increases monotonically with k: That is, whenever $k < k'$ and

$$\zeta \in \Theta_F(\theta_{n,k}, id(dom(D_{\theta_{n,k}})))$$

is such that $\zeta\gamma_{n,k}\Theta_F(\theta_{n,k})$ is in normal form, there exists

$$\zeta' \in \Theta_F(\theta_{n,k'}, id(dom(D_{\theta_{n,k'}})))$$

such that

$$\zeta'\gamma_{n,k'}\Theta_F(\theta_{n,k'}) = \zeta\gamma_{n,k}\Theta_F(\theta_{n,k}).$$

Thus, there exists $k_\infty \in \mathbb{N}$ such that, for all $k \in \mathbb{N}$, there exists

$$\zeta \in T(\Theta_F(\theta_{n,k_\infty}), id(dom(D_{\theta_{n,k_\infty}})))$$

such that

$$(\zeta'_{n,k}\Phi_{\theta'})\downarrow = \zeta\Theta_F(\theta_{n,k_\infty}).$$

Therefore, we can choose $\theta_n = \theta_{n,k_\infty}$.

An entirely analogous algorithm now allows us to conclude that there must exist $n_\infty \in \mathbb{N}$ such that, for all $\zeta' \in T(\Phi_{\theta'})$, there exists

$$\zeta \in T(\Theta_F(\theta_{n_\infty}), id(dom(D_{\theta_{n_\infty}})))$$

such that

$$(\zeta'\Phi_{\theta'})\downarrow = \zeta\Theta_F(\theta_{n_\infty}).$$

Thus, Θ is a saturation of Φ.

□

B.3 D-Static Equivalence Algorithm

In the following Lemma, we consider unification problems containing no variables. It is clear that, if (α, γ) is a (Φ, D)-solution of such problems, then so is $(\emptyset, \gamma)$. Therefore, we will use simply the recipe substitution γ to refer to (Φ, D)-solutions of such problems.

Lemma 15. *Let Φ^1 and Φ^2 be frames such that $T(\Phi^1) = T(\Phi^2)$, D be a DCS for Φ^1 (equivalently, for Φ^2), Θ be a D-saturation of Φ^1, and $\theta \in dom(\Theta)$. Suppose that:*

- *for all $h \in dom(\Theta_F(\theta))$, $\Delta(h)$ is a finite complete set of $(\Theta_F(\theta), D_\theta)$-solutions of $\left\{\rho \stackrel{?}{=} h\Theta_F(\theta)\right\}$, where ρ is a fresh recipe variable;*
- *for all $\gamma \in \Delta(h)$, $\rho\gamma\Theta_T(\theta)\Phi^2 \approx_{\mathcal{R}} (h, \gamma)\Theta_T(\theta)\Phi^2$.*

If $\zeta, \zeta' \in T(\Theta_F(\theta), \theta')$ *for some* $(\Theta_T(\theta)(D_\theta))$*-binding* θ'*, then*

$$\zeta\Theta_F(\theta) = \zeta'\Theta_F(\theta) \quad \Rightarrow \quad \zeta\Theta_T(\theta)\Phi^2 \approx_{\mathcal{R}} \zeta'\Theta_T(\theta)\Phi^2.$$

Proof We define the notion of *depth* $depth(\zeta)$ of a recipe ζ as expected: if $\zeta \in \mathcal{N} \cup \mathcal{X}_R$, then $depth(\zeta) = 0$ and, if $\zeta = h(\zeta_1, \ldots, \zeta_n)$, then $depth(\zeta) = 1$ if $n = 0$ and $depth(\zeta) = 1 + \max_{i\in\{1,\ldots,n\}} depth(\zeta_i)$ otherwise.

Let $<$ be the order relation on

$$T(\Theta_F(\theta), \Theta_T(\theta)(D_\theta)) \times T(\Theta_F(\theta), \Theta_T(\theta)(D_\theta))$$

such that

$$(\zeta_1, \zeta_2) < (\zeta_1', \zeta_2') \qquad \text{iff} \qquad \max_{i\in\{1,2\}} depth(\zeta_i) < \max_{i\in\{1,2\}} depth(\zeta_i').$$

It is simple to check that $<$ is well-founded. The proof of this Lemma is by induction on (ζ, ζ') with the order relation $<$.

In the base case, we have $depth(\zeta) = depth(\zeta') = 0$, and thus, $\zeta = t$ and $\zeta' = t'$ for some $t, t' \in \mathcal{N} \cup \mathcal{X}_R$. In this case, we have $\zeta\Theta_T(\theta) = \zeta$, $\zeta'\Theta_T(\theta) = \zeta'$, and

$$\begin{aligned} \zeta\Theta_F(\theta)\Phi^2 &= t \\ &= t' \\ &= \zeta'\Theta_F(\theta)\Phi^2. \end{aligned}$$

Thus, the result holds.

Suppose now that $\max(depth(\zeta), depth(\zeta')) > 0$, and assume without loss of generality that $depth(\zeta') > depth(\zeta)$. Then, we have $\zeta' = (h, \gamma)$ for some $h \in dom(\Theta_F(\theta))$ and some recipe substitution $(\Theta_F(\theta), \Theta_T(\theta)(D_\theta))$-recipe substitution γ More precisely, suppose that γ is a $(\Theta_F(\theta), \theta')$-recipe substitution, so that $\zeta' \in T(\Theta_F(\theta), \theta')$.

Let ρ_ζ be a recipe variable that does not occur in $h\Theta_F(\theta)$, and define $\gamma' = \gamma \cup \{\rho_\zeta \mapsto \zeta\}$, so that $\zeta = \rho_\zeta\gamma'$ and $\zeta' = (h, \gamma')$. We have

$$\rho_\zeta\gamma'\Theta_F(\theta) = \zeta\Theta_F(\theta) = \zeta'\Theta_F(\theta) = (h, \gamma')\Theta_F(\theta).$$

In other words, γ' is a $(\Theta_F(\theta), \Theta_T(\theta)(D_\theta))$-solution for $\left\{\rho_\zeta \stackrel{?}{=} h\Theta_F(\theta)\right\}$. Thus, there is $\gamma_\Delta \in \Delta(h)$ and a function γ_{**} such that:

- $dom(\gamma_{**}) = rvars(\rho_\zeta\gamma_\Delta) \cup rvars((h, \gamma_\Delta))$
- for all $\rho \in \{\rho_\zeta\} \cup rvars((h, \gamma_\Delta))$ and all $p \in pos(\rho\gamma_\Delta)$, we have $p \in pos(\rho\gamma)$ and either

  $$head(\rho\gamma_\Delta\,|_p) = head(\rho\gamma\,|_p)$$

  or

  $$\gamma\,|_p \in dom(\gamma_{**}) \quad \wedge \quad \gamma_\Delta\,|_p\,\Theta_F(\theta) = \gamma_{**}(\gamma\,|_p).$$

Let us define a recipe substitution

$$\gamma_* \colon rvars(\zeta\gamma_\Delta) \to T(\Theta_F(\theta), \theta')$$

as follows: For each recipe variable $\rho \in dom(\gamma_{**})$, we choose $\rho' \in \{\rho_\zeta\} \cup rvars((h, \gamma_\Delta))$ and a position $p \in pos(\rho'\gamma_\Delta)$ such that $\rho'\gamma_\Delta \mid_p = \rho$, and define $\rho\gamma_* = \rho'\gamma \mid_p$. By definition, we have

$$\rho\gamma_*\Theta_F(\theta) = \rho\gamma_{**}$$

for all $\rho \in dom(\gamma_{**})$. Letting ζ_Δ and ζ'_Δ be the $(\Theta_F(\theta), \Theta_T(\theta)(D_\theta))$-recipes $\rho_\zeta\gamma_\Delta$ and (h, γ_Δ), respectively, it is simple to check that

$$\begin{aligned} \zeta_\Delta\gamma_*\Theta_F(\theta) &= \rho_\zeta\gamma\Theta_F(\theta) \\ &= \zeta\Theta_F(\theta). \end{aligned}$$

and

$$\begin{aligned} \zeta'_\Delta\gamma_*\Theta_F(\theta) &= (h, \gamma)\Theta_F(\theta) \\ &= \zeta'\Theta_F(\theta); \end{aligned}$$

it follows that

$$\zeta_\Delta\gamma_*\Theta_F(\theta) = \zeta'_\Delta\gamma_*\Theta_F(\theta).$$

By hypothesis,

$$\zeta_\Delta\Theta_T(\theta)\Phi^2 \approx_{\mathcal{R}} \zeta'_\Delta\Theta_T(\theta)\Phi^2.$$

Now, let $\rho \in dom(\gamma_{**})$, and suppose that $\zeta_1 \in \{\rho_\zeta, (h, \emptyset)\}$. Let ζ_2 and p_2 be such that $\zeta_2 \in \{\rho_\zeta, (h, \emptyset)\}$, $p_2 \in pos(\zeta_2\gamma_\Delta)$, and

$$\zeta_2\gamma_\Delta \mid_{p_2} = \rho \qquad \text{and} \qquad \rho\gamma_* = \zeta_2\gamma \mid_{p_2}.$$

We have $p_1 \in pos(\zeta_1\gamma)$, $p_2 \in pos(\zeta_2\gamma)$, and thus

$$(\zeta_1 \mid_{p_1}, \zeta_2 \mid_{p_2}) < (\zeta, \zeta').$$

Moreover, we have

$$\begin{aligned} \zeta_1\gamma \mid_{p_1} \Theta_F(\theta) &= \rho\gamma_{**} \\ &= \zeta_2\gamma \mid_{p_2} \Theta_F(\theta) \\ &= \rho\gamma_*\Theta_F(\theta). \end{aligned}$$

The induction hypothesis then implies that

$$\zeta_1\gamma \mid_{p_1} \Theta_T(\theta)\Phi^2 \approx_{\mathcal{R}} \rho\gamma_*\Theta_T(\theta)\Phi^2.$$

We conclude that

$$\begin{aligned} \zeta\Theta_T(\theta)\Phi^2 &\approx_{\mathcal{R}} (\zeta_\Delta\gamma_*)\Theta_T(\theta)\Phi^2 \\ &\approx_{\mathcal{R}} (\zeta_\Delta\Theta_T(\theta))(\Theta_T(\theta)\gamma_*)\Phi^2 \\ &\approx_{\mathcal{R}} (\zeta'_\Delta\Theta_T(\theta))(\Theta_T(\theta)\gamma_*)\Phi^2 \\ &\approx_{\mathcal{R}} (\zeta'_\Delta\gamma_*)\Theta_T(\theta)\Phi^2 \\ &\approx_{\mathcal{R}} \zeta'\Theta_T(\theta)\Phi^2, \end{aligned}$$

proving the result.

□

Lemma 16. *Let Φ^1 and Φ^2 be frames such that $T(\Phi^1) = T(\Phi^2)$, D be a DCS for Φ^1 (or equivalently, for Φ^2), and Θ be a D-saturation of Φ^1. Suppose that:*

- *for all $\theta \in dom(\Theta)$ and all $h \in dom(\Theta_F(\theta))$, $\Delta(h, \theta)$ is a finite complete set of $(\Theta_F(\theta), \Theta_T(\theta)(D_\theta))$-solutions of $\left\{\rho \stackrel{?}{=} h\Theta_F(\theta)\right\}$, where ρ is a fresh recipe variable;*
- *for all $\theta \in dom(\Theta)$, all $h \in dom(\Theta_F(\theta))$, all $h_s \in sub(h\Theta_F(\theta))$, and all $l \in \mathcal{R}_L$, $\Delta(l, h, h_s, \theta)$ is a finite complete set of $(\Theta_F(\theta), \Theta_T(\theta)(D_\theta))$-solutions of $\left\{l \stackrel{?}{=} h_s\right\}$;*
- *for all $h \in dom(\Theta_F(\theta))$ and all $\gamma \in \Delta(h, \theta)$,*

$$\rho\gamma\Theta_T(\theta)\Phi^2 \approx_{\mathcal{R}} (h, \gamma)\Theta_T(\theta)\Phi^2;$$

- *for all $(\alpha, \gamma) \in \Delta(l, h, h_s, \theta)$, there exists*

$$\zeta \in T(\Theta_F(\theta), \theta_{\Theta_T(\theta)(D_\theta),\gamma})$$

such that

$$\zeta\Theta_F(\theta) = ((h, \gamma)\Theta_F(\theta))\downarrow$$

and

$$(h, \gamma)\Theta_T(\theta)\Phi^2 \approx_{\mathcal{R}} \zeta\Theta_T(\theta')\Phi^2.$$

Then, for each D-binding θ' of Φ^1, there exist $\theta \in dom(\Theta)$ and

$$\gamma\colon dom(D_\theta) \to T(\Theta_F(\theta), \Theta_T(\theta)(D_\theta))$$

such that:

- *$\theta\gamma\Theta_F(\theta) \approx_{\mathcal{R}} \theta'\Phi^1$,*
- *for all $\zeta' \in T(\Phi^1, \theta')$, there exists*

$$\zeta \in T(\Theta_F(\theta), id(dom(D_\theta)))$$

such that

$$(\zeta\gamma)\Theta_F(\theta) = (\zeta'\Phi^1)\downarrow,$$

and

- *for all*

$$\zeta \in T(\Theta_F(\theta), id(dom(D_\theta))),$$

there exists

$$\zeta_{nf} \in T(\Theta_F(\theta), id(dom(D_\theta)))$$

such that:

- $\zeta_{nf}\gamma\Theta_F(\theta) = (\zeta\gamma\Theta_F(\theta))\!\downarrow$, *and*
- $\zeta_{nf}\gamma\Theta_T(\theta)\Phi^2 \approx_{\mathcal{R}} \zeta\gamma\Theta_T(\theta)\Phi^2$.

Proof. Let

$$\iota\colon dom(D_{\theta'}) \rightarrowtail \mathcal{N} \setminus (names(\Phi^1) \cup names(\Phi^2))$$

be an injective mapping of the recipe variables in $dom(D_{\theta'})$ to fresh names. Then, $\theta'\iota$ is a ground D-binding of Φ^1 and, by the definition of saturation, there exist $\theta \in dom(\Theta)$ and a $(\Theta_F(\theta), \Theta_T(\theta)(D_\theta))$-recipe substitution γ such that:

- $\theta\gamma\Theta_F(\theta) \approx_{\mathcal{R}} \theta'\iota\Phi^1$, and
- for all $\zeta'' \in T(\Phi^1_{\theta'\iota})$, there exists $\zeta \in T(\Theta_F(\theta), id(dom(D_\theta)))$ such that

$$\zeta\gamma\Theta_F(\theta) = \zeta''\Phi_{\theta'\iota}.$$

Suppose that $\zeta' \in T(\Phi^1, \theta')$, and let ζ'' be the $(\Phi^1, id(dom(D)))$-recipe such that $\zeta''\theta' = \zeta'$. Then, we have $\zeta''\iota \in T(\Phi^1_{\theta'\iota})$, and there is

$$\zeta \in T(\Theta_F(\theta), id(dom(D_\theta)))$$

such that

$$\zeta\gamma\Theta_F(\theta) = (\zeta''\Phi^1_{\theta'\iota})\!\downarrow .$$

Because ι is injective and all elements of its domain are fresh names, we also have

$$\begin{aligned}\theta\gamma\iota^{-1}\Theta_F(\theta) &\approx_{\mathcal{R}} \theta'\iota\iota^{-1}\Phi^1\\ &= \theta'\Phi^1\end{aligned}$$

and

$$\begin{aligned}\zeta\gamma\iota^{-1}\Theta_F(\theta) &= (\zeta''\Phi^1_{\theta'\iota})\!\downarrow(\iota^{-1})\\ &= (\zeta''\Phi^1_{\theta'}\!\downarrow\\ &= \zeta'\Phi^1\!\downarrow,\end{aligned}$$

These equations show that $\gamma\iota^{-1}$ satisfies the first two properties. It remains to show that the third property also holds.

For $\zeta \in T(\Theta_F(\theta), id(dom(\theta)))$, let $|\zeta|_{\mathcal{R}}$ be the length of the longest $\mathcal{R}$-rewriting sequence starting at $\zeta\gamma\Theta_F(\theta)$. The proof is by induction on $|\zeta|_{\mathcal{R}}$. If $|\zeta|_{\mathcal{R}} = 0$, then $\zeta\gamma\Theta_F(\theta)$ is in normal form, and it is sufficient to take $\zeta_{nf} = \zeta$.

Suppose then that $|\zeta|_{\mathcal{R}} = n$, and that the result is valid for all ζ such that $|\zeta|_{\mathcal{R}} < n$. Let

$$h \in dom(\Theta_F(\theta)) \qquad \text{and} \qquad \zeta_1, \ldots, \zeta_n \in T(\Theta_F(\theta), id(dom(D_\theta)))$$

be such that $\zeta = h(\zeta_1, \ldots, \zeta_n)$.

If there is $i \in \{1,\ldots,n\}$ such that $|\zeta_i|_{\mathcal{R}} = n$, then $\zeta_j\gamma\Theta_F(\theta)$ is in normal form for all $j \in \{1,\ldots,n\}\setminus\{i\}$. Suppose that $\zeta_{i,nf}$ satisfies the two properties we want to prove for ζ_i, i.e., if

$$\zeta_{i,nf}\gamma\Theta_F(\theta) = (\zeta_i\gamma\Theta_F(\theta)\downarrow \qquad \text{and} \qquad \zeta_{i,nf}\gamma\Theta_T(\theta)\Phi^2 \approx_{\mathcal{R}} \zeta\gamma\Theta_T(\theta)\Phi^2,$$

and let

$$\zeta_{nf} = h(\zeta_1,\ldots,\zeta_{i-1},\zeta_{i,nf},\zeta_{i+1,nf},\ldots,\zeta_n).$$

Then, $\zeta\gamma\Theta_F(\theta)$ rewrites to $\zeta_{nf}\gamma\Theta_F(\theta)$ in exactly n steps, and thus $\zeta'_{nf}\gamma\Theta_F(\theta)$ is in normal form (since $|\zeta|_{\mathcal{R}} = n$)). It is straightforward to check that ζ_{nf} is a witness of the result, since

$$\begin{aligned}\zeta_{nf}\gamma\Theta_F(\theta) &= h(\zeta_1,\ldots,\zeta_{i,nf},\ldots,\zeta_n)\Theta_F(\theta)(\Theta_F(\theta)\circ\gamma)\\ &\approx_{\mathcal{R}} h(\zeta_1,\ldots,\zeta_i,\ldots,\zeta_n)\Theta_F(\theta)(\Theta_F(\theta)\circ\gamma)\\ &= \zeta\gamma\Theta_F(\theta)\end{aligned}$$

and $\zeta_{nf}\gamma\Theta_F(\theta)$ is in normal form, and

$$\begin{aligned}\zeta_{nf}\gamma\Theta_T(\theta)\Phi^2 &= h(\zeta_1,\ldots,\zeta_{i,nf},\ldots,\zeta_n)\Theta_T(\theta)\Phi^2(\Phi^2\circ\Theta_T(\theta)\circ\gamma)\\ &\approx_{\mathcal{R}} h(\zeta_1,\ldots,\zeta_i,\ldots,\zeta_n)\Theta_T(\theta)\Phi^2(\Phi^2\circ\Theta_T(\theta)\circ\gamma)\\ &= \zeta\gamma\Theta_T(\theta)\Phi^2.\end{aligned}$$

Thus, it is sufficient to prove that the result holds for ζ_i. Iterating this reasoning, we conclude that it is sufficient to prove the result in the case that there is no $i \in \{1,\ldots,n\}$ such that $|\zeta_i|_{\mathcal{R}} = n$.

In the case that $|\zeta_i| < n$ for all $i \in \{1,\ldots,n\}$. For each $i \in \{1,\ldots,n\}$, the induction hypothesis implies that there exists

$$\zeta_{i,nf} \in T(\Theta_F(\theta), id(dom(D_\theta)))$$

such that

$$\zeta_{i,nf}\gamma\Theta_F(\theta) = (\zeta_i\gamma\Theta_F(\theta)\downarrow \qquad \text{and} \qquad \zeta_{i,nf}\gamma\Theta_T(\theta)\Phi^2 \approx_{\mathcal{R}} \zeta\gamma\Theta_T(\theta)\Phi^2.$$

Let

$$\zeta_{nf} = h(\zeta_{1,nf},\ldots,\zeta_{n,nf}).$$

We consider two cases: either (1) $\zeta_{nf}\gamma\Theta_F(\theta)$ is in normal form, or (2) it is not. In case (1) it is again straightforward to check that ζ_{nf} is a witness of the result, since

$$\begin{aligned}\zeta_{nf}\gamma\Theta_F(\theta) &= h(\zeta_{1,nf},\ldots,\zeta_{i,nf},\ldots,\zeta_{n,nf})\Theta_F(\theta)(\Theta_F(\theta)\circ\gamma)\\ &\approx_{\mathcal{R}} h(\zeta_1,\ldots,\zeta_i,\ldots,\zeta_n)\Theta_F(\theta)(\Theta_F(\theta)\circ\gamma)\\ &= (\zeta\gamma\Theta_F(\theta))\downarrow,\end{aligned} \tag{B.5}$$

using the fact that $\zeta_{nf}\gamma\Theta_F(\theta)$ is in normal form, and

$$\begin{aligned}\zeta_{nf}\gamma\Theta_T(\theta)\Phi^2 &= h(\zeta_{1,nf},\ldots,\zeta_{n,nf})\Theta_T(\theta)\Phi^2(\Phi^2\circ\Theta_T(\theta)\circ\gamma)\\ &\approx_{\mathcal{R}} h(\zeta_1,\ldots,\zeta_n)\Theta_T(\theta)\Phi^2(\Phi^2\circ\Theta_T(\theta)\circ\gamma)\\ &= \zeta\gamma\Theta_T(\theta)\Phi^2.\end{aligned} \tag{B.6}$$

It remains to consider case (2). In this case, $\zeta_{nf}\gamma\Theta_F(\theta)$ is not in normal form, and there must exist $h_s \in sub(h\Theta_F(\theta)) \setminus \mathcal{X}_R$, $l \in \mathcal{R}_L$, and a term substitution α such that

$$h_s\Theta_F(\theta)(\Theta_F(\theta) \circ \gamma) = l\alpha.$$

Note that $h_s \notin \mathcal{X}_R$ since that would imply that $\zeta_{i,nf}\gamma\Theta_F(\theta)$ is not in normal form for some $i \in \{1, \ldots, n\}$, contradicting our induction hypothesis. Let

$$\gamma^+ \colon rvars(h\Theta_F(\theta)) \to T(\Theta_F(\theta), \theta_{\Theta_T(\theta)(D_\theta),\gamma})$$

be the recipe variable substitution such that $\zeta_{nf}\gamma = (h, \gamma^+)$. Then, (α, γ^+) is a $(\Theta_F(\theta), \Theta_T(\theta)(D_\theta))$-solution of $\left\{l \stackrel{?}{=} h_s\right\}$, and there exist $(\alpha', \gamma') \in \Delta(l, h, h_s, \theta)$ and a function δ such that, for all positions $p \in pos((h, \gamma'))$ such that $(h, \gamma') \mid_p \in \mathcal{X}_R$, we have $p \in pos((h, \gamma^+))$ and

$$(h, \gamma^+) \mid_p \Theta_F(\theta) = (h, \gamma') \mid_p \delta.$$

By hypothesis, there exists $\zeta'_{nf} \in T(\Theta_F(\theta), \Theta_T(\theta)(D_\theta))$ such that

$$\zeta'_{nf}\Theta_F(\theta) = (h, \gamma')\Theta_F(\theta)\downarrow \tag{B.7}$$

and

$$\zeta'_{nf}\Theta_T(\theta)\Phi^2 \approx_{\mathcal{R}} (h, \gamma')\Theta_T(\theta)\Phi^2. \tag{B.8}$$

We define a recipe substitution

$$\gamma_* \colon rvars((h, \gamma')) \to T(\Theta_F(\theta), \theta_{\Theta_T(\theta)(D_\theta),\gamma})$$

by choosing, for each recipe variable $\rho \in rvars((h, \gamma'))$, a position $p \in pos((h, \gamma'))$ such that $(h, \gamma') \mid_p = \rho$, and defining $\rho\gamma_* = (h, \gamma^+) \mid_p$.

Note that, for each position $p \in pos((h, \gamma'))$ such that $(h, \gamma') \mid_p \in \mathcal{X}_R$, we have

$$\begin{aligned}(h, \gamma') \mid_p \gamma_*\Theta_F(\theta) &= (h, \gamma') \mid_p \delta \\ &= (h, \gamma^+) \mid_p \Theta_F(\theta) \\ &= (\zeta_{nf}\gamma) \mid_p \Theta_F(\theta).\end{aligned} \tag{B.9}$$

Noting that $head((h, \gamma') \mid_{p'}) = head((h, \gamma^+) \mid_{p'})$ for all positions $p' \in pos((h, \gamma')))$ such that $(h, \gamma') \mid_p \notin \mathcal{X}_R$, we may extend these identities to (h, γ'), obtaining

$$\begin{aligned}(h, \gamma')\gamma_*\Theta_F(\theta) &= (h, \gamma^+)\Theta_F(\theta) \\ &= \zeta_{nf}\gamma\Theta_F(\theta).\end{aligned}$$

By Lemma 15, this implies

$$\zeta_{nf}\gamma\Theta_T(\theta)\Phi^2 \approx_{\mathcal{R}} (h, \gamma')\gamma_*\Theta_T(\theta)\Phi^2$$

Using equation (B.8), we may thus obtain

$$\begin{array}{ll} \zeta_{nf}\gamma\Theta_T(\theta)\Phi^2 & \approx_{\mathcal{R}} (h,\gamma^+)\Theta_T(\theta)\Phi^2 \\ & \approx_{\mathcal{R}} (h,\gamma')\gamma_*\Theta_T(\theta)\Phi^2 \\ & = (h,\gamma')\Theta_T(\theta)\Phi^2(\Phi^2 \circ \Theta_T(\theta) \circ \gamma_*) \\ & \approx_{\mathcal{R}} (\zeta'_{nf}\Theta_T(\theta)\Phi^2)(\Phi^2 \circ \Theta_T(\theta) \circ \gamma_*) \\ & = \zeta'_{nf}\gamma_*\Theta_T(\theta)\Phi^2. \end{array} \tag{B.10}$$

Since $(h,\gamma')\Theta_F(\theta) \to^+_{\mathcal{R}} \zeta'_{nf}\Theta_F(\theta)$, we also have

$$\begin{array}{ll} \zeta_{nf}\gamma\Theta_F(\theta) & = (h,\gamma')\gamma_*\Theta_F(\theta) \\ & \to^+_{\mathcal{R}} \zeta'_{nf}\gamma_*\Theta_F(\theta). \end{array} \tag{B.11}$$

Now, we observe that there exists $\zeta' \in T(\Theta_F(\theta), id(dom(D_\theta)))$ such that

$$(h,\gamma') = \zeta'\theta_{\Theta_T(\theta)(D_\theta),\gamma'}.$$

By our choice of θ, there exists $\zeta'' \in T(\Theta_F(\theta), id(dom(D_\theta)))$ such that $\zeta''\gamma\Theta_F(\theta) = \zeta'\gamma\Theta_F(\theta)$. Moreover, we have

$$\begin{array}{ll} \gamma & = \theta_{\Theta_T(\theta)(D_\theta),\gamma^+} \\ & = \theta_{\Theta_T(\theta)(D_\theta),\gamma_*\circ\gamma'} \\ & = \theta_{\Theta_T(\theta)(D_\theta),\gamma_*} \circ \theta_{\Theta_T(\theta)(D_\theta),\gamma'}. \end{array}$$

It follows that

$$\begin{array}{ll} \zeta''\theta_{\Theta_T(\theta)(D_\theta),\gamma'}\Theta_F(\theta) & = (\zeta'\theta_{\Theta_F(\theta)(D_\theta),\gamma'}\Theta_F(\theta))\downarrow \\ & = \zeta'_{nf}\Theta_F(\theta), \end{array}$$

and

$$\zeta''\theta_{\Theta_T(\theta)(D_\theta),\gamma'}\gamma_*\Theta_F(\theta) = \zeta'_{nf}\gamma_*\Theta_F(\theta);$$

therefore, there exists $\zeta^* \in T(\Theta_F(\theta), id(dom(D_\theta)))$ such that

$$\begin{array}{ll} \zeta^*\gamma & = \zeta^*\theta_{\Theta_T(\theta)(D_\theta),\gamma'}\theta_{\Theta_T(\theta)(D_\theta),\gamma_*}\Theta_F(\theta) \\ & = \zeta'_{nf}\gamma_*\Theta_F(\theta), \end{array}$$

and Lemma 15 implies that

$$\zeta^*\gamma\Theta_T(\theta)\Phi^2 \approx_{\mathcal{R}} \zeta'_{nf}\gamma_*\Theta_T(\theta)\Phi^2.$$

Using equation (B.11), we have

$$\begin{array}{ll} \zeta_{nf}\gamma\Theta_F(\theta) & \to^+_{\mathcal{R}} \zeta'_{nf}\gamma_*\Theta_F(\theta) \\ & = \zeta^*\gamma\Theta_F(\theta); \end{array} \tag{B.12}$$

combining this equality with equation (B.10) and Lemma 15, it follows that

$$\zeta^*\gamma\Theta_T(\theta)\Phi^2 \quad \approx_{\mathcal{R}} \zeta'_{nf}\gamma_*\Theta_T(\theta)\Phi^2. \tag{B.13}$$

By equation (B.12), we can apply our induction hypothesis to the recipe ζ^*. This guarantees that there exists a recipe ζ^*_{nf} such that

$$\begin{aligned}(\zeta^*_{nf}\gamma)\Theta_F(\theta) &= (\zeta^*\gamma\Theta_F(\theta))\downarrow \\ &= (\zeta_{nf}\gamma\Theta_F(\theta))\downarrow \\ &= (\zeta\gamma\Theta_F(\theta))\downarrow,\end{aligned}$$

where we use equations (B.12) and (B.5), and

$$\begin{aligned}(\zeta^*_{nf}\gamma)\Theta_T(\theta)\Phi^2 &\approx_{\mathcal{R}} \zeta^*\gamma\Theta_T(\theta)\Phi^2 \\ &\approx_{\mathcal{R}} \zeta'_{nf}\gamma_*\Theta_T(\theta)\Phi^2 \\ &\approx_{\mathcal{R}} \zeta_{nf}\gamma\Theta_T(\theta)\Phi^2 \\ &\approx_{\mathcal{R}} \zeta\gamma\Theta_T(\theta)\Phi^2,\end{aligned}$$

using equations (B.13), (B.10), and (B.6). Thus, ζ^*_{nf} is a witness of the result.

□

Lemma 17. *Suppose that Φ^1, Φ^2, D, Θ, θ', θ and γ are as in the conditions of Lemma 16. Then, for all $\rho \in dom(D)$,*

$$\rho\theta'\Phi^2 \approx_{\mathcal{R}} \rho(\Theta_T(\theta) \circ \gamma)\Phi^2.$$

Proof . Let $D = (\rho_1, K_1), \ldots, (\rho_n, K_n)$. It is sufficient to prove that, for all $i \in \{1, \ldots, n\}$,

$$\rho_i\theta'\Phi^2 \approx_{\mathcal{R}} \rho_i(\Theta_T(\theta) \circ \gamma)\Phi^2.$$

We proceed by induction on i.

If $i = 1$, then $\rho_i\theta' \in T(\Phi\mid_{K_1})$ and, for all handles h occurring in $\rho_i\theta'$, we have

$$rvars(h\Phi) \cap dom(D) = rvars(h\Phi') \cap dom(D) = \emptyset.$$

This means that $(h, \theta) = (h, \emptyset)$ for all such h, and thus

$$\rho_i\theta' \in T(\Theta_F(\theta)) \qquad \text{and} \qquad \rho_i\theta'\Theta_F(\theta) \approx_{\mathcal{R}} \rho_i\theta'\Phi^1.$$

Lemma 16 implies that there exists

$$\zeta \in T(\Theta_F(\theta), id(dom(D_\theta)))$$

such that

$$\zeta\gamma\Theta_F(\theta) = (\rho_i\theta'\Phi^1)\downarrow \qquad \text{and} \qquad \zeta\gamma\Theta_T(\theta)\Phi^2 \approx_{\mathcal{R}} \rho_i\theta'\Phi^2.$$

Similarly, there exists

$$\zeta' \in T(\Theta_F(\theta), id(dom(D_\theta)))$$

such that

$$\zeta'\gamma\Theta_F(\theta) = (\rho_i\theta(\Theta_T(\theta)\circ\gamma)\Phi^1 \qquad \text{and} \qquad \zeta'\gamma\Theta_T(\theta)\Phi^2 \approx_{\mathcal{R}} \rho_i\theta(\Theta_T(\theta)\circ\gamma)\Phi^2.$$

Since

$$\rho_i\theta(\Theta_T(\theta)\circ\gamma)\Phi^1 \approx_{\mathcal{R}} \rho_i\theta'\Phi^1,$$

we also have

$$\begin{aligned}\zeta\gamma\Theta_F(\theta) &= (\rho_i\theta'\Phi^1)\downarrow \\ &= (\rho_i\theta(\Theta_T(\theta)\circ\gamma)\Phi^1)\downarrow \\ &= \zeta'\gamma\Theta_F(\theta).\end{aligned}$$

Lemma 15 then implies that

$$\begin{aligned}\rho_i\theta'\Phi^2 &\approx_{\mathcal{R}} \zeta_i'\gamma\Theta_T(\theta)\Phi^2 \\ &\approx_{\mathcal{R}} \zeta_i\gamma\Theta_T(\theta)\Phi^2 \\ &\approx_{\mathcal{R}} \rho_i\theta(\Theta_T(\theta)\circ\gamma)\Phi^2,\end{aligned}$$

proving the result.

Suppose then that the result is valid for all $i \in \{1,\ldots,k\}$ for some k, and consider the case that $i = k+1$. Only handles in $\Phi^1 \mid_{K_i}$ occur in $\rho_i\theta'$ and $\rho_i\theta$; thus, there exist $(\Phi^1 \mid_{K_i}, id(dom(D)))$-recipes ζ_i and ζ_i' such that

$$\zeta_i\theta(\Theta_T(\theta)\circ\gamma) = \rho_i\theta(\Theta_T(\theta)\circ\gamma) \qquad \text{and} \qquad \zeta_i'\theta' = \rho_i\theta'.$$

Consider the image of $\zeta_i\gamma$ and $\zeta_i'\gamma$ under $\Theta_F(\theta)$. We have

$$\begin{aligned}\zeta_i\gamma\Theta_F(\theta) &= \zeta_i\Theta_F(\theta)(\Theta_F(\theta)\circ\gamma) \\ &\approx_{\mathcal{R}} \zeta_i\theta\Phi^1(\Phi^1\circ\Theta_T(\theta)\circ\gamma) \\ &= \zeta_i\theta(\Theta_T(\theta)\circ\gamma)\Phi^1 \\ &= \rho_i\theta(\Theta_T(\theta)\circ\gamma)\Phi^1\end{aligned}$$

and, similarly,

$$\begin{aligned}\zeta_i'\gamma\Theta_F(\theta) &= \zeta_i'\Theta_F(\theta)(\Theta_F(\theta)\circ\gamma) \\ &\approx_{\mathcal{R}} \zeta_i'\theta\Phi^1(\Phi^1\circ\Theta_T(\theta)\circ\gamma) \\ &\approx_{\mathcal{R}} \zeta_i'\theta'\Phi^1 \\ &= \rho_i\theta'\Phi^1.\end{aligned}$$

Using the two equations above and the property that

$$\theta(\Theta_T(\theta)\circ\gamma)\Phi^1 \approx_{\mathcal{R}} \theta'\Phi^1,$$

we obtain

$$\begin{aligned}\zeta_i\gamma\Theta_F(\theta) &\approx_{\mathcal{R}} \rho_i\theta(\Theta_T(\theta)\circ\gamma)\Phi^1 \\ &\approx_{\mathcal{R}} \rho_i\theta'\Phi^1 \\ &\approx_{\mathcal{R}} \zeta_i'\gamma\Theta_F(\theta).\end{aligned}$$

By Lemma 16, there exist

$$\zeta_{i,nf}, \zeta_{i,nf}' \in T(\Theta_F(\theta), id(dom(D_\theta)))$$

such that

$$(\zeta_{i,nf}\gamma)\Theta_T(\theta)\Phi^2 \approx_{\mathcal{R}} \zeta_i\gamma\Theta_T(\theta)\Phi^2, \tag{B.14}$$

$$(\zeta'_{i,nf}\gamma\Theta_T(\theta))\Phi^2 \approx_{\mathcal{R}} \zeta'_i\gamma\Theta_T(\theta)\Phi^2 \tag{B.15}$$

and

$$\begin{aligned}\zeta_{i,nf}\gamma\Theta_F(\theta) &= (\zeta_i\gamma\Theta_F(\theta))\downarrow \\ &= (\zeta' i_\gamma\Theta_F(\theta))\downarrow \\ &= \zeta_{i,nf}\gamma\Theta_F(\theta).\end{aligned}$$

By Lemma 15, the above equality also implies that

$$\zeta_{i,nf}\gamma\Theta_T(\theta)\Phi^2 \approx_{\mathcal{R}} \zeta'_{i,nf}\gamma\Theta_T(\theta)\Phi^2;$$

combining this with equalities (B.14) and (B.15), it follows that

$$\zeta_i\gamma\Theta_T(\theta)\Phi^2 \approx_{\mathcal{R}} \zeta'_i\gamma\Theta_T(\theta)\Phi^2. \tag{B.16}$$

Now, we also have

$$\zeta'_i\gamma\Theta_T(\theta) = \zeta'_i\theta(\Theta_T(\theta)\circ\gamma),$$

and since $\zeta'_i \in T(\Phi^2 \mid_{K_i})$, the only recipes in $dom(D)$ occurring in ζ_i are in $\{\rho_1,\ldots,\rho_k\}$. By the induction hypothesis,

$$\rho_j\theta(\Theta_T(\theta)\circ\gamma)\Phi^2 \approx_{\mathcal{R}} \rho_j\theta'\Phi^2$$

for all $j \in \{1,\ldots,k\}$. This implies that

$$\begin{aligned}\zeta'_i\gamma\Theta_T(\theta)\Phi^2 &\approx_{\mathcal{R}} \zeta'_i\theta(\Theta_T(\theta)\circ\gamma)\Phi^2 \\ &\approx_{\mathcal{R}} \zeta'_i\theta'\Phi^2 \\ &\approx_{\mathcal{R}} \rho_i\theta'\Phi^2.\end{aligned} \quad .$$

Moreover,

$$\begin{aligned}\zeta_i\gamma\Theta_T(\theta)\Phi^2 &= \zeta_i\theta(\Theta_T(\theta)\circ\gamma)\Phi^2 \\ &= \rho_i\theta(\Theta_T(\theta)\circ\gamma)\Phi^2.\end{aligned}$$

Combining the two equations above with equation (B.16), we obtain

$$\rho_i\theta(\Theta_T(\theta)\circ\gamma)\Phi^2 \approx_{\mathcal{R}} \rho_i\theta'\Phi^2,$$

proving the result.

□

Lemma 18. *Suppose that Φ^1, Φ^2, D, and Θ satisfy the conditions of Lemma 16. Then, for each D-binding θ' of Φ^1, there exist $\theta \in dom(\Theta)$ and*

$$\gamma\colon dom(D_\theta) \to T(\Theta_F(\theta), \Theta_T(\theta)(D_\theta))$$

such that:

- $\theta\gamma\Theta_F(\theta) \approx_{\mathcal{R}} \theta'\Phi^1$, *and*

- *for all* $\zeta' \in T(\Phi^1, \theta')$*, there exists*

$$\zeta \in T(\Theta_F(\theta), id(dom(D_\theta)))$$

such that:

  - $(\zeta\gamma)\Theta_F(\theta) = (\zeta'\Phi^1)\downarrow$*, and*
  - $(\zeta\gamma)\Theta_T(\theta)\Phi^2 \approx_{\mathcal{R}} \zeta'\Phi^2$.

Proof. Lemma 16 guarantees that θ and γ satisfy all the properties in the conclusion of the Lemma except that, for all $\zeta' \in T(\Phi^1, \theta')$, there exists

$$\zeta \in T(\Theta_F(\theta), id(dom(D_\theta)))$$

such that

$$(\zeta\gamma)\Theta_T(\theta)\Phi^2 \approx_{\mathcal{R}} \zeta'\Phi^2.$$

To prove this last property we proceed by induction on ζ'. If $\zeta' \in \mathcal{N} \cup \mathcal{X}_R$, then taking $\zeta = \zeta'$ yields the desired result (note that $dom(\gamma) = dom(\Theta_T(\theta)(D_\theta))$, and thus these variables may always be renamed).

Suppose then that $\zeta' = (h, \gamma')$ for some $h \in dom(\Phi^1)$ and some (Φ^1, θ')-recipe substitution γ'. For $\rho \in rvars(h\Phi^1) \cap dom(D)$, we have $\rho\gamma' = \rho\theta'$; thus, letting

$$\gamma'' = \gamma' \mid_{rvars(h\Phi^1) \setminus dom(D)},$$

we have

$$(h, \gamma') = (h, \theta')\gamma''.$$

Let $\rho_1, \ldots, \rho_n$ be such that

- $\{\rho_1, \ldots, \rho_n\} = rvars(h\Phi^1) \setminus dom(D)$, and
- $\rho_1 \prec_{\mathcal{X}_R} \ldots \prec_{\mathcal{X}_R} \rho_n$.

For each $i \in \{1, \ldots, n\}$, we will write ζ_i' for the recipe $\rho_i\gamma''$. Note that $\zeta_i' \in T(\Phi^1, \theta')$ for all $i \in \{1, \ldots, n\}$. By the induction hypothesis, there exist

$$\zeta_1^*, \ldots, \zeta_n^* \in T(\Theta_F(\theta), id(dom(D_\theta)))$$

such that

$$(\zeta_i^*\gamma)\Theta_F(\theta) = (\zeta_i'\Phi^1)\downarrow \qquad \text{and} \qquad (\zeta_i^*\gamma)\Theta_T(\theta)\Phi^2 \approx_{\mathcal{R}} \zeta_i'\Phi^2.$$

We also have

$$\begin{array}{ll} (h, \gamma)\Theta_F(\theta) & = h\Theta_F(\theta)(\Theta_F(\theta) \circ \gamma) \\ & \approx_{\mathcal{R}} (h, \theta)\Phi^1(\Theta_F(\theta) \circ \gamma) \\ & \approx_{\mathcal{R}} (h, \emptyset)\theta\gamma\Theta_F(\theta) \\ & \approx_{\mathcal{R}} (h, \theta')\Phi^1. \end{array} \tag{B.17}$$

Consider the recipe substitution

$$\gamma^*\colon rvars(h\Phi^1) \setminus dom(D) \to T(\Theta_F(\theta), id(dom(D_\theta)))$$

given by

$$i \in \{1, \ldots, n\} \quad \Rightarrow \quad \rho_i\gamma^* = \zeta_i^*.$$

It is clear that $(h, \gamma^*) \in (\Theta_F(\theta), id(dom(D_\theta)))$, and by the definition of ζ_i^* we have

$$\begin{array}{ll} \rho\gamma^*\gamma\Theta_T(\theta)\Phi^1 & \approx_{\mathcal{R}} \rho\gamma^*\gamma\Theta_F(\theta) \\ & \approx_{\mathcal{R}} \rho\gamma'\Phi^1 \end{array} \tag{B.18}$$

and

$$\begin{array}{ll} \rho\gamma^*\gamma\Theta_T(\theta)\Phi^2 & \approx_{\mathcal{R}} \rho\gamma^*\gamma\Theta_T(\theta)\Phi^2 \\ & \approx_{\mathcal{R}} \rho\gamma'\Phi^2 \end{array} \tag{B.19}$$

for all $\rho \in rvars(h\Phi^1) \setminus dom(D)$ (note that $rvars(h\Phi^2) = rvars(h\Phi^1)$ since $T(\Phi^1) = T(\Phi^2)$). By Lemma 16, there exists $\zeta_{nf} \in T(\Theta_F(\theta), id(dom(D_\theta)))$ such that

$$\zeta_{nf}\gamma\Theta_F(\theta) = ((h, \gamma^*)\gamma\Theta_F(\theta))\downarrow$$

and

$$\zeta_{nf}\gamma\Theta_T(\theta)\Phi^2 \approx_{\mathcal{R}} (h, \gamma^*)\gamma\Theta_T(\theta)\Phi^2.$$

Thus,

$$\begin{array}{ll} \zeta_{nf}\gamma\Theta_F(\theta) & = ((h, \gamma^*)\gamma\Theta_F(\theta))\downarrow \\ & = (h, \gamma)\Theta_F(\theta)(\Theta_F(\theta) \circ \gamma \circ \gamma^*)\downarrow \\ & \approx_{\mathcal{R}} (h, \theta')\Phi^1(\Phi^1 \circ \Theta_T(\theta) \circ \gamma \circ \gamma^*)\downarrow \\ & \approx_{\mathcal{R}} (h, \theta')\Phi^1(\Phi^1 \circ \gamma')\downarrow \\ & = (h, \gamma')\Phi^1\downarrow, \end{array} \tag{B.20}$$

where we use equation (B.18) on the fourth step.

Moreover, since $T(\Phi^1) = T(\Phi^2)$, we have $rvars(h\Phi^1) = rvars(h\Phi^2)$, and thus, for all $\rho \in rvars(h\Phi^2) \setminus dom(D)$, equation (B.17) carries over to the frame Φ^2, i.e.:

$$\begin{array}{ll} (h, \gamma)\Theta_T(\theta)\Phi^2 & = h\Theta_T(\theta)\Phi^2(\Phi^2 \circ \Theta_T(\theta) \circ \gamma) \\ & \approx_{\mathcal{R}} (h, \theta)\Phi^2(\Phi^2 \circ \Theta_T(\theta) \circ \gamma) \\ & \approx_{\mathcal{R}} (h, \emptyset)\theta\gamma\Theta_T(\theta)\Phi^2 \\ & \approx_{\mathcal{R}} (h, \theta')\Phi^2, \end{array} \tag{B.21}$$

using Lemma 17.

$$\begin{array}{ll} \zeta_{nf}\gamma\Theta_T(\theta)\Phi^2 & \approx_{\mathcal{R}} (h, \gamma^*)\gamma\Theta_T(\theta)\Phi^2 \\ & = (h, \gamma)\Theta_T(\theta)\Phi^2(\Phi^2 \circ \Theta_T(\theta) \circ \gamma \circ \gamma^*) \\ & \approx_{\mathcal{R}} (h, \theta')\Phi^2(\Phi^2 \circ \gamma'') \\ & \approx_{\mathcal{R}} (h, \theta')\gamma''\Phi^2 \\ & = (h, \gamma')\Phi^2, \end{array} \tag{B.22}$$

using equations (B.19) and (B.21) on the third step. Equations (B.20) and (B.22) show that ζ_{nf} is a witness of the result.

□

Proof (Theorem 6)**.** First we prove the "only if" side of the theorem. If there are $i \in \{1,2\}$, $\theta \in dom(\Theta^i)$, and $\gamma \in \Delta(h,\theta)$ such that

$$\rho\gamma\Theta_T(\theta)\Phi^{3-i} \not\approx_{\mathcal{R}} (h,\gamma)\Theta_T(\theta)\Phi^{3-i},$$

then the recipes $\rho\gamma\Theta_T(\theta)$ and $(h,\gamma)\Theta_T(\theta)$ are witnesses that $\Phi^1 \not\approx^s_{\mathcal{R},D} \Phi^2$, since

$$\begin{aligned}\rho\gamma\Theta_T(\theta)\Phi^i &\approx_{\mathcal{R}} \rho\gamma\Theta_F(\theta)\\ &= (h,\gamma)\Theta_F(\theta)\\ &\approx_{\mathcal{R}} (h,\gamma)\Theta_T(\theta)\Phi^i.\end{aligned}$$

Similarly, if there is $i \in \{1,2\}$, $\theta \in dom(\Theta^i)$, and $(\alpha,\gamma) \in \Delta(h,\theta)$ such that

$$(h,\gamma)\Theta_T(\theta)\Phi^{3-i} \not\approx_{\mathcal{R}} \zeta\Theta_T(\theta')\Phi^{3-i},$$

then the recipes $(h,\gamma)\Theta_T(\theta)$ and $\zeta\Theta_T(\theta')$ are witnesses that $\Phi^1 \not\approx^s_{\mathcal{R},D} \Phi^2$, since we have

- $(h,\gamma)\Theta_T(\theta) \in T(\Phi^i,\theta')$,
- $\zeta\Theta_T(\theta) \in T(\Phi^i,\theta')$, and
- $(h,\gamma)\Theta_T(\theta)\Phi^i \approx_{\mathcal{R}} \zeta\Theta_T(\theta')\Phi^i$,

by construction.

To prove the "if" side of the theorem, let θ' be some D-binding of Φ^i, and $\zeta,\zeta' \in T(\Phi^i,\theta')$ be such that $\zeta\Phi^i \approx_{\mathcal{R}} \zeta'\Phi^i$. Then, by Lemma 18, there exist:

- $\theta_1,\theta_2 \in dom(\Theta)$,
- $\gamma_1\colon dom(\Theta_T(\theta_1)(D_{\theta_1})) \to T(\Theta_F(\theta_1),\Theta_T(\theta_1)(D_{\theta_1}))$,
- $\gamma_2\colon dom(\Theta_T(\theta_2)(D_{\theta_2})) \to T(\Theta_F(\theta_2),\Theta_T(\theta_2)(D_{\theta_2}))$,
- $\zeta_1^* \in T(\Theta_F(\theta_1), id(dom(D_{\theta_1})))$, and
- $\zeta_2^* \in T(\Theta_F(\theta_2), id(dom(D_{\theta_2})))$

such that:

- $\zeta_1^*\gamma_1\Theta_F(\theta_1) = \zeta\Phi^i\!\downarrow$;
- $\zeta_2^*\gamma_2\Theta_F(\theta_2) = \zeta'\Phi^i\!\downarrow$;
- $\zeta_1^*\Theta_T(\theta_1)\Phi^{3-i} \approx_{\mathcal{R}} \zeta\Phi^{3-i}$;

- $\zeta_2^* \Theta_T(\theta_2) \Phi^{3-i} \approx_{\mathcal{R}} \zeta' \Phi^{3-i}$.

Because $\zeta \Phi^i \approx_{\mathcal{R}} \zeta' \Phi^i$, we have $\zeta \Phi^i \downarrow = \zeta' \Phi^i \downarrow$, and thus also

$$\zeta_1^* \gamma_1 \Theta_F(\theta_1) = \zeta_2^* \gamma_2 \Theta_F(\theta_2).$$

Lemma 15 then yields

$$\zeta_1^* \Theta_T(\theta_1) \Phi^{3-i} \approx_{\mathcal{R}} \zeta_2^* \Theta_T(\theta_2) \Phi^{3-i}.$$

Combining these equalities, we obtain

$$\begin{array}{ll} \zeta \Phi^{3-i} & \approx_{\mathcal{R}} \zeta_1^* \Theta_T(\theta_1) \Phi^{3-i} \\ & \approx_{\mathcal{R}} \zeta_2^* \Theta_T(\theta_2) \Phi^{3-i} \\ & \approx_{\mathcal{R}} \zeta' \Phi^{3-i}, \end{array}$$

proving the result.

□

B.4 Constraints Systems

Proof (Theorem 7)**.** We first show that Θ^+ is a set of D-bindings of Φ that satisfy C^+. If $\theta^+ \in \Theta^+$, there is $\theta \in \Theta$, $\varrho \in \Upsilon_\theta$, and $(\alpha, \gamma) \in \Delta_{\theta,\varrho}$ such that

- $\theta^+ = (\gamma \Theta_T(\theta)) \circ \theta$,
- and (α, γ) is a $(\Theta_F(\theta), \Theta_T(\theta)(D_\theta))$-solution of

$$\bigcup_{\rho \in dom(D_\theta)} \left\{ \rho \stackrel{?}{=} (\rho\varrho)\downarrow \right\}.$$

Because $ran(\Theta_T(\theta)) \subseteq T(\Phi, \theta)$, we have

$$(\rho \in dom(D_\theta) \quad \wedge \quad h \in dom(\Theta_T(\theta))) \quad \Rightarrow \quad (h\Theta_T(\theta))[\rho] = \{\rho\}.$$

Since γ is a $(\Theta_F(\theta), D_\theta)$-recipe, we also have that

$$\rho \in dom(D_\theta) \quad \Rightarrow \quad (\gamma \Theta_T(\theta))[\rho] = \Theta_T(\theta)[\gamma[\rho]]$$

is a set with a single element in $T(\Phi \mid_{D_\theta(\rho)})$. Therefore, $\gamma \Theta_T(\theta)$ is a (Φ, D_θ)-substitution, and it follows that $(\gamma \Theta_T(\theta)) \circ \theta$ is a D-binding of Φ.

By definition of $\Theta_T(\theta)$, we have, for all $\rho \in dom(D_\theta)$

$$\begin{array}{ll} \rho\gamma\Theta_T(\theta)\Phi & \approx_{\mathcal{R}} \rho\gamma\Theta_F(\theta) \\ & = (\rho\varrho)\downarrow \alpha. \end{array}$$

It follows that, for all $(t \overset{?}{\approx}_{\mathcal{R}} t') \in C^+$, we have

$$\begin{array}{ll} t(\gamma\Theta_T(\theta) \circ \theta)\Phi & = t(\theta\Phi)(\Phi \circ \Theta_T(\theta) \circ \gamma) \\ & \approx_{\mathcal{R}} t(\theta\Phi)(\Theta_F(\theta) \circ \gamma) \\ & = t(\theta\Phi)(\varrho\alpha), \end{array}$$

and similarly for t':

$$\begin{array}{ll} t'(\gamma\Theta_T(\theta) \circ \theta)\Phi & = t'(\theta\Phi)(\Phi \circ \Theta_T(\theta) \circ \gamma) \\ & \approx_{\mathcal{R}} t'(\theta\Phi)(\Theta_F(\theta) \circ \gamma) \\ & = t'(\theta\Phi)(\varrho\alpha). \end{array}$$

Since

$$t(\theta\Phi)\varrho\alpha \approx_{\mathcal{R}} t'(\theta\Phi)\varrho\alpha$$

(because $\varrho \in \Upsilon$ and $(t \overset{?}{\approx}_{\mathcal{R}} t') \in C^+$), we conclude that

$$t(\gamma\Theta_T(\theta) \circ \theta)\Phi \approx_{\mathcal{R}} t'(\gamma\Theta_T(\theta) \circ \theta)\Phi$$

for all $(t \overset{?}{\approx}_{\mathcal{R}} t') \in C^+$; in other words, $\gamma\Theta_T(\theta) \circ \theta$ is a D-binding of Φ satisfying C^+.

Completeness. To prove that Θ^+ is complete, suppose that θ' is a D-binding of Φ that satisfies C^+. Because Θ is a D-saturation of Φ, there exist $\theta \in dom(\Theta)$ and a $(\Theta_F(\theta), \Theta_F(\theta)(D_\theta))$-recipe substitution γ such that:

- $\theta\gamma\Theta_F(\theta) \approx_{\mathcal{R}} \theta'\Phi$,
- and, for all $\zeta' \in \Phi_{\theta'}$, there exists $\zeta \in T(\Theta_F(\theta), id(dom(D_\theta)))$ such that

$$\zeta\gamma\Theta_F(\theta) = (\zeta'\Phi_{\theta'})\downarrow .$$

We have

$$(t \overset{?}{\approx}_{\mathcal{R}} t' \in C^+) \quad \Rightarrow \quad t(\Phi \circ \theta') \approx_{\mathcal{R}} t'(\Phi \circ \theta').$$

Moreover, we have $\Phi_\theta \downarrow\subseteq \Theta_F(\theta)$, and thus, for all $h \in dom(\Phi_\theta)$, $h\Phi_\theta \approx_{\mathcal{R}} h\Theta_F(\theta)$; therefore,

$$\begin{array}{ll} t(\theta\Phi)\gamma\Theta_F(\theta) & \approx_{\mathcal{R}} t(\theta\Theta_F(\theta))(\gamma\Theta_F(\theta)) \\ & = t\theta\gamma\Theta_F(\theta) \\ & \approx_{\mathcal{R}} t(\Phi \circ \theta'). \end{array}$$

The same reasoning holds replacing t by t', and it follows that

$$\begin{array}{ll} t(\theta\Phi)\gamma\Theta_F(\theta) & \approx_{\mathcal{R}} t(\Phi \circ \theta') \\ & \approx_{\mathcal{R}} t'(\Phi \circ \theta') \\ & \approx_{\mathcal{R}} t'(\theta\Phi)\gamma\Theta_F(\theta). \end{array}$$

In particular, $\Theta_F(\theta) \circ \gamma$ is an $\approx_{\mathcal{R}}$-unifier for

$$\bigcup_{(t \stackrel{?}{\approx}_{\mathcal{R}} t') \in C^+} \left\{ t\theta\Phi \stackrel{?}{\approx}_{\mathcal{R}} t'\theta\Phi \right\}.$$

Therefore, there exists $\varrho \in \Upsilon_\theta$ and a term substitution α such that

$$\rho \in dom(D_\theta) \cap (rvars(t) \cup rvars(t')) \Rightarrow \rho\gamma\Theta_F(\theta) \approx_{\mathcal{R}} \rho\varrho\alpha. \tag{B.23}$$

Now, we note that γ can be chosen such that $\rho\gamma\Theta_F(\theta)$ is in normal form for all $\rho \in dom(D_\theta)$. Indeed, let

$$\theta'' = (\Theta_T(\theta) \circ \theta_{\Theta_T(\theta)(D_\theta),\gamma}) \circ \theta.$$

For all $\rho \in dom(D_\theta)$,

$$\rho\gamma\Theta_T(\theta) \in T(\Phi_{\theta''}, D_{\theta''}),$$

and

$$\rho\gamma\Theta_T(\theta)\Phi_{\theta''} \approx_{\mathcal{R}} \rho\gamma\Theta_F(\theta).$$

Now, we note that, for all $\rho \in dom(D)$,

$$\begin{aligned} \rho\theta'' &= \rho\theta(\gamma\Theta_T(\theta)) \\ &\approx_{\mathcal{R}} \rho\theta'\Phi. \end{aligned}$$

Combining this with the fact (stated above) that $\rho\gamma\Theta_T(\theta)\Phi_{\theta''} \approx_{\mathcal{R}} \rho\gamma\Theta_F(\theta)$, we conclude that there exists $\zeta' \in T(\Phi_{\theta'})$ such that

$$\zeta'\Phi'_\theta \approx_{\mathcal{R}} \rho\gamma\Theta_F(\theta).$$

By the definition of saturation, there exist $\zeta \in T(\Theta_F(\theta), id(dom(D_\theta)))$ and a $(\Theta_F(\theta), \Theta_T(\theta)(D_\theta))$-recipe substitution γ' such that

$$\begin{aligned} \zeta\gamma'\Theta_F(\theta) &= (\zeta'\Phi'_\theta)\downarrow \\ &= \rho\gamma\Theta_F(\theta)\downarrow . \end{aligned}$$

Since such recipes exist for all $\rho \in dom(D)$, we conclude that we can choose γ such that $\rho\gamma\Theta_F(\theta)$ is in normal form for all $\rho \in dom(D_\theta)$.

Combining this result with equation (B.23), we conclude that $(\alpha\downarrow, \gamma)$ is a $(\Theta_F(\theta), \Theta_T(\theta)(D_\theta))$-solution of the unification problem

$$\mathcal{U}_{\theta,\varrho} = \bigcup_{\rho \in dom(D_\theta)} \left\{ \rho \stackrel{?}{=} (\rho\varrho)\downarrow \right\}.$$

Since $\Delta_{\theta,\varrho}$ is a complete set of $(\Theta_F(\theta), \Theta_T(\theta)(D_\theta))$-solutions of this problem, there is $(\alpha', \gamma') \in \Delta_{\theta,\varrho}$ such that

$$(\alpha', \gamma') \prec^{\mathcal{U}_{\theta,\varrho}, \Theta_F(\theta)} (\alpha\downarrow, \gamma),$$

i.e., there exist α_* and $\gamma_* \colon \mathcal{X}_R \to T_\Sigma(\mathcal{N} \cup \mathcal{X}_R)$ such that:

- $\alpha\downarrow = \alpha_* \circ \alpha'$,
- and, for all $\rho \in rvars(\mathcal{U}_{\theta,\varrho})$ and all $p \in pos(\rho\gamma)$, we have $p \in pos(\rho\gamma')$ and either:
  - $head(\rho\gamma\mid_p) = head(\rho\gamma'\mid_p)$;
  - or $\rho\gamma'\mid_p \in dom(\gamma_*)$ and $\rho\gamma\mid_p \Theta_F(\theta) = \rho\gamma'\mid_p \gamma_*$.

Consider a $(\Theta_F(\theta), D_\theta)$-substitution γ_{**} such that, for each $\rho \in rvars(\mathcal{U}_{\theta,\varrho})$ and each $p \in pos(\rho\gamma')$ such that $\rho\gamma'\mid_p \in dom(\gamma_*)$, there exist $\rho' \in rvars(\mathcal{U}_{\theta,\varrho})$ and $p' \in pos(\rho'\gamma')$ such that

$$(\rho\gamma')\mid_p = (\rho'\gamma')\mid_{p'}$$

(implying that $(\rho\gamma)\mid_p \Theta_F(\theta) = (\rho'\gamma)\mid_{p'} \Theta_F(\theta)$), and

$$(\rho\gamma')\mid_p \gamma_{**} = (\rho'\gamma)\mid_{p'} .$$

Note that $\gamma_{**} \circ \gamma'$ can be obtained from γ by a series of transformations, each of which replacing $\rho\gamma\mid_p$ by $(\rho'\gamma'\mid_{p'} \gamma_{**}$ for some $\rho' \in rvars(\mathcal{U}_{\theta,\varrho})$ and some position $p' \in pos(\rho'\gamma)$ such that $\rho\gamma'\mid_p = \rho'\gamma'\mid_{p'}$, and thus

$$\rho\gamma'\mid_p \Theta_F(\theta) = \rho'\gamma'\mid_{p'} \Theta_F(\theta).$$

We have

$$\begin{aligned}(\rho\gamma\mid_p)\Theta_F(\theta) &= \rho'\gamma\mid_{p'} \Theta_F(\theta)\\ &= \rho\gamma'\gamma_{**}\Theta_F(\theta).\end{aligned}$$

Therefore, we have

$$\Theta_F(\theta) \circ \gamma = \Theta_F(\theta) \circ \gamma_* \circ \gamma'.$$

We show that taking

$$\Theta_T(\theta) \circ \gamma_* \circ \gamma'$$

as the γ in the statement of the theorem yields the desired result. Because of the way we choose θ, we have

$$\theta\gamma\Theta_F(\theta) \approx_{\mathcal{R}} \theta'\Phi.$$

Moreover, we have seen above that

$$\begin{aligned}\gamma\Theta_F(\theta) &\approx_{\mathcal{R}} \gamma'\gamma_*\Theta_F(\theta)\\ &\approx_{\mathcal{R}} \gamma'\gamma_*\Theta_T(\theta)\Phi;\end{aligned}$$

combining these results, it follows that

$$\theta(\Theta_T(\theta) \circ \gamma_* \circ \gamma')\Phi \approx_{\mathcal{R}} \theta'\Phi.$$

To prove the second property, let $\zeta' \in T(\Phi, \theta')$. Because of the way we choose θ and γ, there exists $\zeta'' \in T(\Theta_F(\theta), id(dom(D_\theta)))$ such that

$$\zeta''\gamma\Theta_F(\theta) = (\zeta'\Phi)\downarrow .$$

Choosing $\zeta = \zeta\Theta_T(\theta)$, we have

$$\begin{aligned}(\zeta'\Phi)\downarrow &= \zeta''\gamma\Theta_F(\theta)\\ &= (\zeta''\Theta_F(\theta))(\gamma\Theta_F(\theta))\\ &= (\zeta''\Theta_F(\theta))\downarrow\,(\gamma\Theta_F(\theta))\downarrow\\ &= (\zeta\Theta_T(\theta)\Phi)\downarrow\,((\Theta_T(\theta)\circ\gamma_*\circ\gamma')\Phi)\downarrow,\end{aligned}$$

with the third equality following from the fact that $\zeta''\gamma\Theta_F(\theta)$ is in normal form, and thus so is $(\zeta''\Theta_F(\theta))(\gamma''\Theta_F(\theta))$. Since $\theta \in \Theta^+$, we conclude that Θ^+ is a complete set of D-bindings of Φ satisfying C^+.

□

Proof (Theorem 8)**.** We first prove that, if the Algorithm returns `false` , then $C^1 \not\sim C^2$. For simplicity, we assume that `false` is output when executing lines 1—8 for the first time.

Correction of `false` output. The case that `false` is returned when executing these lines for the second time is entirely analogous. If `false` is returned in line 4 then either $\theta\iota \in \mathsf{sol}(\mathsf{C}^1) \setminus \mathsf{sol}(C^2)$, or $\Phi^1_\theta \not\approx^s_{\mathcal{R},D} \Phi^2_\theta$. In the first case it is clear that $\theta\iota$ is a witness of the fact that $C^1 \not\sim C^2$. In the second case, there exists a D_θ-binding θ' of Φ^1_θ (and Φ^2_θ) and $\zeta, \zeta' \in T(\Phi^1_\theta, \theta')$ such that either

$$\zeta\Phi^1_\theta \approx_{\mathcal{R}} \zeta'\Phi^1_\theta \qquad \text{and} \qquad \zeta\Phi^2_\theta \not\approx_{\mathcal{R}} \zeta'\Phi^2_\theta$$

or

$$\zeta\Phi^2_\theta \approx_{\mathcal{R}} \zeta'\Phi^2_\theta \qquad \text{and} \qquad \zeta\Phi^1_\theta \not\approx_{\mathcal{R}} \zeta'\Phi^1_\theta.$$

There exist $\zeta_*, \zeta'_* \in T(\Phi^1, id(dom(D)))$ such that

$$\zeta_*\theta\theta'\Phi^1 = \zeta\Phi^1_\theta \qquad \text{and} \qquad \zeta'_*\theta\theta'\Phi^1 = \zeta'\Phi^1_\theta$$

and, analogously,

$$\zeta_*\theta\theta'\Phi^2 = \zeta_*\Phi^2_\theta \qquad \text{and} \qquad \zeta'_*\theta\theta'\Phi^2 = \zeta'\Phi^2_\theta.$$

Assume without loss of generality that the first case holds. Then, we have

$$\begin{aligned}\zeta_*(\theta'\circ\theta)\Phi^1 &= \zeta_*\theta\theta'\Phi^1\\ &= \zeta\Phi^1_\theta\\ &\approx_{\mathcal{R}} \zeta'\Phi^1_\theta\\ &= \zeta'_*\theta\theta'\Phi^1\\ &= \zeta'_*(\theta'\circ\theta)\Phi^1.\end{aligned}$$

On the other hand,

$$\begin{aligned}\zeta_*(\theta'\circ\theta)\Phi^2 &= \zeta_*\theta\theta'\Phi^2\\ &= \zeta\Phi^2\theta\end{aligned}$$

and

$$\begin{aligned}\zeta'_*(\theta'\circ\theta)\Phi^2 &= \zeta'_*\theta\theta'\Phi^2\\ &= \zeta'\Phi^2\theta,\end{aligned}$$

implying that

$$\zeta_*(\theta' \circ \theta)\Phi^2 \not\approx_{\mathcal{R}} \zeta'_*(\theta' \circ \theta)\Phi^2.$$

Letting

$$\iota \colon dom(D_{\theta' \circ \theta}) \rightarrowtail \mathcal{N} \setminus (names(\Phi^1_{\theta' \circ \theta}) \cup names(\Phi^2_{\theta' \circ \theta}))$$

be an injective renaming of the parameters in $dom(D_{\theta' \circ \theta})$ into fresh names, it follows that $\theta\theta'\iota$ is a solution of C^1 such that

$$\Phi^1_{\theta\theta'\iota} \not\approx^s_{\mathcal{R},D} \Phi^2_{\theta\theta'\iota};$$

thus, $\theta\theta'\iota$ is a witness of the fact that $C^1 \not\sim C^2$.

If `false` is returned in line 8, then θ' is a $(\Phi^2_\theta, D_\theta)$-solution of $\left\{t \stackrel{?}{=} t'\right\}$; thus, $\theta\theta'$ is a (Φ^2, D)-solution of $\left\{t \stackrel{?}{=} t'\right\}$. Since $(t \stackrel{?}{\neq} t') \in N^2$, it follows that $\theta' \circ \theta$ is not a solution of C^2, but it is a solution of C^1. Again taking a ground, injective, fresh renaming of the parameters in $D_{\theta\theta'}$ into names

$$\iota \colon dom(D_{\theta' \circ \theta}) \rightarrowtail \mathcal{N} \setminus (names(\Phi^1_{\theta' \circ \theta}) \cup names(\Phi^2_{\theta' \circ \theta})),$$

it follows that $\theta\theta'\iota$ is a witness that $C^1 \not\sim C^2$.

Correction of `true` output. Suppose then that the algorithm outputs `true` . For simplicity, we prove that, for all solutions θ' of C^1, θ' is a solution of C^2 and $\Phi^1_{\theta'} \approx^s_{\mathcal{R}} \Phi^2_{\theta'}$. The proof of the reciprocal statement is analogous, since line 9 ensures that the algorithm is executed exchanging the roles of Φ^1 and Φ^2.

By Theorem 7, there exists $\theta \in \mathsf{genUnifP}(\Phi^1, D^1, P^1)$ and a recipe variable substitution

$$\gamma \colon dom(D_\theta) \to T(\Phi^1_\theta, D_\theta)$$

such that

- $\theta'\Phi^1 \approx_{\mathcal{R}} \theta_*\gamma_*\Theta_F(\theta)$, and
- for all $\zeta' \in T(\Phi^1, \theta')$, there is $\zeta \in T(\Phi^1, \theta)$ such that

$$\zeta\gamma\Phi^1 \approx_{\mathcal{R}} \zeta'\Phi^1.$$

We have $\Phi^1_\theta \approx^s_{\mathcal{R},D_\theta} \Phi^2_\theta$; thus, noting that γ is a D_θ-binding of Φ^1_θ, we also have

$$\Phi^1_{\gamma\circ\theta} \approx^s_{\mathcal{R},D_{\gamma\circ\theta}} \Phi^2_{\gamma\circ\theta}.$$

For all $D_{\theta'}$-bindings θ'' of $\Phi^1_{\theta'}$ and all $\zeta \in T(\Phi^1_{\theta'}, \theta'')$, there is

$$\zeta_* \in T(\Phi^1_{\theta'}, id(dom(D_{\theta'})))$$

such that

$$\begin{aligned}\zeta\Phi^1_{\theta'} &= \zeta_*\theta''\Phi^1_{\theta'}\\ &= \zeta_*\Phi^1_{\theta'}(\Phi^1_{\theta'}\circ\theta'')\\ &\approx_{\mathcal{R}} \zeta_*\Phi^1_{\gamma\circ\theta}(\Phi^1_{\gamma\circ\theta}\circ\theta'')\\ &= \zeta_*\theta''\Phi^1_{\gamma\circ\theta}\\ &= \zeta\Phi^1_{\gamma\circ\theta},\end{aligned}$$

with the third step using the fact that

$$\theta\gamma\Phi^1 \approx_{\mathcal{R}} \theta'\Phi^1,$$

and thus also

$$\Phi^1_{\gamma\circ\theta} \approx_{\mathcal{R}} \Phi^1_{\theta'}.$$

We conclude that

$$\Phi^1_{\gamma\circ\theta} \approx^s_{\mathcal{R},D_{\theta'}} \Phi^1_{\theta'}. \tag{B.24}$$

The fact that $\Phi^1_\theta \approx^s_{\mathcal{R},D_\theta} \Phi^2_\theta$ implies that all the conditions of Lemma 17 hold, and thus we have, as above,

$$\theta\gamma\Phi^2 \approx_{\mathcal{R}} \theta'\Phi^2,$$

$$\Phi^2_{\gamma\circ\theta} \approx_{\mathcal{R}} \Phi^2_{\theta'},$$

and

$$\Phi^2_{\gamma\circ\theta} \approx^s_{\mathcal{R},D_{\theta'}} \Phi^2_{\theta'}. \tag{B.25}$$

Combining equations (B.24) and (B.25) with the fact that $\Phi^1_\theta \approx^s_{\mathcal{R},D_\theta} \Phi^2_\theta$, we obtain

$$\begin{aligned}\Phi^1_{\theta'} &\approx_{\mathcal{R}} \Phi^1_{\theta(\Theta_T(\theta)\circ\gamma)}\\ &= (\Phi^1_\theta)_{\Theta_T(\theta)\circ\gamma)}\\ &\approx^s_{\mathcal{R},D_{\theta'}} (\Phi^2_\theta)_{\Theta_T(\theta)\circ\gamma}\\ &= \Phi^2_{\theta(\Theta_T(\theta)\circ\gamma)}\\ &= \Phi^2_{\theta'}.\end{aligned} \tag{B.26}$$

Now, if there is $(t \stackrel{?}{\neq} t') \in N^2$ such that

$$t\Phi^2_{\gamma\circ\theta} \approx_{\mathcal{R}} t'\Phi^2_{\gamma\circ\theta},$$

(i.e., if $\gamma\circ\theta$ does not satisfy the negative constraints of C^2), then by Theorem 7 there exists a D_θ-binding $\theta'' \in \mathsf{genUnifP}(\Phi^2_\theta, D^2, \left\{t \stackrel{?}{=} t'\right\})$ of Φ^2_θ and a recipe variable substitution

$$\gamma'' \colon dom(D_\theta) \to T(\Phi^2, D_\theta)$$

such that

$$\gamma\Phi^2_\theta \approx_{\mathcal{R}} \theta''\gamma''\Phi^2_\theta$$

and, for all $\zeta' \in T(\Phi^2_\theta, \gamma)$, there exists $\zeta \in T(\Phi^2_\theta, \theta'')$ such that

$$\zeta'\Phi^2_\theta \approx_{\mathcal{R}} \zeta\gamma''\Phi^2_\theta.$$

Since the algorithm returns `true` by hypothesis, the test in line 7 implies that $\theta'' \circ \theta$ is not a solution of C^1. However, since θ is a solution of P^1 and $\approx_{\mathcal{R}}$ is stable under substitution of recipe variables by arbitrary terms, $\theta'' \circ \theta$ is also a solution of P^1. It follows that $\theta'' \circ \theta$ is not a solution of N^1, i.e., there is $(t_1 \stackrel{?}{\neq} t_1') \in N^1$ such that

$$t_1\Phi^1_{\theta''\circ\theta} \approx_{\mathcal{R}} t_1'\Phi^1_{\theta''\circ\theta}.$$

Because $\Phi^1_\theta \approx^s_{\mathcal{R},D_\theta} \Phi^2_\theta$, Lemma 17 implies that

$$(\gamma'' \circ \theta'')\Phi^1_\theta \approx_{\mathcal{R}} \gamma\Phi^1_\theta.$$

Therefore,

$$\begin{aligned} t_1\Phi^1_{\theta'} &\approx_{\mathcal{R}} t_1\Phi^1_{\gamma\circ\theta} \\ &\approx_{\mathcal{R}} t_1\Phi^1_{\gamma''\circ\theta''\circ\theta} \\ &= t_1\Phi^1_{\theta''\circ\theta}(\Phi^1 \circ \gamma'') \\ &\approx_{\mathcal{R}} t_1'\Phi^1_{\theta''\circ\theta}(\Phi^1 \circ \gamma'') \\ &= t_1'\Phi^1_{\gamma''\circ\theta''\circ\theta} \\ &\approx_{\mathcal{R}} t_1'\Phi^1_{\gamma\circ\theta} \\ &\approx_{\mathcal{R}} t_1'\Phi^1_{\theta'}. \end{aligned}$$

This implies that θ' is not a solution of C^1, contradicting our hypothesis. We conclude that $t\Phi^2_{\theta'} \not\approx_{\mathcal{R}} t'\Phi^2_{\theta'}$ for all θ' that are solutions of C^1 and all $(t \stackrel{?}{\neq} t') \in N^2$. Moreover, Φ^2_θ satisfies all equations in P^2; Since $\approx_{\mathcal{R}}$ is stable under the substitution of recipe variables by arbitrary terms,

$$\Phi^2_\theta\gamma = \Phi^2_{\gamma\circ\theta}$$

also does. This shows that θ' is a solution of Φ^2.

Combining this result with equation (B.26), we conclude that, for all solutions θ' of C^1, θ' is a solution of C^2 and $\Phi^1_{\theta'} \approx_{\mathcal{R}} \Phi^2_{\theta'}$, concluding the proof.

□

Appendix C

Estimated Probability Measures

In this Appendix we prove the results of Chapter 5 concerning the probability measures $\mu^{K,\prec}$, μ_{ROM} and μ_C.

C.1 Well-definedness

In this section we show that $\mu^{K,\prec}$ and μ_{ROM} are well-defined and relate them by proving Theorem 12.

Proof (Theorem 9)**.** We first note that if $\lambda, \lambda' \in \Lambda_{sub[K]}$ are distinct, then $\Omega_\lambda \neq \Omega_{\lambda'}$. Therefore, $\mu^{K,\prec}$ is well-defined.

Letting $\lambda_{\mathcal{B}^*_\perp} = \{t \mapsto \mathcal{B}^*_\perp \mid t \in sub[K]\}$, we have $\Omega_{\lambda_{\mathcal{B}^*_\perp}} \in \mathcal{F}_K$ and $\mathbf{\Omega} = \Omega_{\lambda_{\mathcal{B}^*_\perp}}$; thus, $\mu^{K,\prec}(\Omega_{\lambda_{\mathcal{B}^*_\perp}}) = 1$.

Now, each $\Omega \in \mathcal{F}_K$ can be written as $\Omega = \biguplus_{\psi\in\Psi} \Omega_\psi$ for some countable subset Ψ of Ψ_K. Sets of the form $\biguplus_{\psi\in\Psi'} \Omega_\psi$ for some countable subset Ψ' of Ψ_K are closed under complementation and countable unions. Conversely, for each $\psi \in \Psi_{sub[K]}$, let $\lambda_\psi \colon sub[K] \to \mathcal{P}(\mathcal{B}^*_\perp)$ be defined by $\lambda_\psi(t) = \{\psi(t)\}$ for each $t \in K$. It is clear that $\Omega_{\lambda_\psi} = \Omega_\psi$, and thus $\Omega_\psi \in \Omega_\Lambda \subseteq \mathcal{F}$ for all $\psi \in \Psi_K$.

We conclude that, for all $\Omega \in \mathbf{\Omega}$, $\Omega \in \mathcal{F}_K$ if and only if there exists a sequence of *distinct* $\psi_1, \ldots, \psi_k \in \Psi_{sub[K]}$ (with $k \in \mathbb{N} \cup \{\infty\}$) such that $\Omega = \biguplus_{i=1}^{k} \Omega_{\psi_i}$. Thus, for any sequence $\{\Omega_i\}_{i\in\mathbb{N}}$ such that $\Omega_i \in \mathcal{F}_K$ for all $i \in \mathbb{N}$ and $\Omega_i \cap \Omega_j = \emptyset$ whenever $i \neq j$, there exist $k_i \in \mathbb{N} \cup \{\infty\}$ and $\psi_{i,j} \in \Psi_{sub[K]}$ (with $j \in \{1, \ldots, k_i\}$) such that $\Omega_i = \biguplus_{j=1}^{k_i} \Omega_{\psi_{i,j}}$ and $\biguplus_{i\in\mathbb{N}} \Omega_i = \biguplus_{i\in\mathbb{N}} \biguplus_{j=1}^{k_i} \Omega_{\psi_{i,j}}$. The definition of $\mu^{K,\prec}$ implies that

$$\mu^{K,\prec}\left(\biguplus_{i\in\mathbb{N}} \Omega_i\right) = \mu^{K,\prec}\left(\biguplus_{i\in\mathbb{N}} \biguplus_{j=1}^{k_i} \Omega_{\psi_{i,j}}\right) = \sum_{i\in\mathbb{N}} \sum_{j=1}^{k_i} \mu(\Omega_{\psi_{i,j}}),$$

and, for each $i \in \mathbb{N}$,

$$\mu^{K,\prec}(\Omega_i) = \mu^{K,\prec}\left(\biguplus_{j=1}^{k_i} \Omega_{\psi_{i,j}}\right) = \sum_{j=1}^{k_i} \mu(\Omega_{\psi_{i,j}}).$$

Thus, $\mu^{K,\prec}$ is σ-additive:

$$\mu^{K,\prec}\left(\biguplus_{i\in\mathbb{N}} \Omega_i\right) = \sum_{i\in\mathbb{N}} \sum_{j=1}^{k_i} \mu(\Omega_{\psi_{i,j}}) = \sum_{i\in\mathbb{N}} \mu(\Omega_i),$$

concluding the proof.

□

If K is a finite set of terms and $\iota \in I(K)$, we will write $\omega \models \iota$ to denote that ω satisfies ι. If $\psi \in \Psi_{sub[K]}$ for some finite set of terms K, we say that ψ satisfies ι, and write $\psi \models \iota$, if, for all $t = f(t_1, \ldots, t_n) \in K$, we have

$$(\psi(t_1), \ldots, \psi(t_n)) \in [\![dom(\iota(t))]\!] \qquad \text{and} \qquad \psi(t) \in [\![ran(\iota(t))]\!].$$

Lemma 19. *Let K be a finite set of terms and $\psi \in \Psi_{sub[K]}$. Then, there exists at most one $\iota \in I(K)$ such that $\psi \models \iota$.*

Proof. For each $t \in sub[K]$, letting $t = f(t_1, \ldots, t_n)$, there is at most one $ps \in PS_f$ such that $(\psi(t_1), \ldots, \psi(t_n)) \in [\![dom(ps)]\!]$. If $\psi \models \iota$, we must have $\iota(t) = ps$ in this case, and $\iota(t) = \bot$ if no such ps exists; thus, there is at most one ι that is satisfied by ψ.

□

In light of Lemma 19, if $K = sub[K]$ and $\psi \in \Psi_K$, we will write $\iota(\psi)$ for the only $\iota \in I(K)$ such that $\psi \models \iota$, and set $\iota(\psi) = \bot$ if no such ι exists. Moreover, if $\psi \colon sub[K] \to \mathcal{B}^*_\bot$, we say that ψ satisfies $\approx_{W(\psi)}$ if, whenever $t, t' \in sub[K]$ are such that $t \approx_{W(\psi)} t'$, we have

$$\psi(t) = \psi(t') \quad \text{or} \quad \psi(t) = \bot \quad \text{or} \quad \psi(t') = \bot.$$

Lemma 20. *At any point of any execution of Algorithm 9, there is a finite set K' such that $dom(\psi_{ROM}) = sub[K']$. Furthermore, ψ_{ROM} satisfies $\approx_{W(\psi_{ROM})}$ and exactly one selection function $\iota \in I_{\mathcal{S}}(K')$.*

Proof. Consider an execution of Algorithm 9 for input K, using a subterm-compatible order $\prec$. Let $t_1, \ldots, t_n$ be such that

$$sub[K] = \{t_1, \ldots, t_n\} \qquad \text{and} \qquad t_1 \prec \ldots \prec t_n.$$

Algorithm 9 samples $\psi_{ROM}(t_1), \ldots, \psi_{ROM}(t_n)$, in this order. For all $i \in \{0, \ldots, n\}$, let $K_i = \{t_1, \ldots, t_i\}$ and let $\psi_{ROM,i}$ be the function ψ_{ROM} used in the algorithm after the i-th execution of the cycle in lines 3-13, so that $dom(\psi_{ROM,i}) = K_i$.

In the beginning of the execution, we have

$$K_0 = dom(\psi_{ROM}) = \emptyset \qquad \text{and} \qquad I_{\mathcal{S}}(K_0) = \{\emptyset\}\,.$$

Thus, the result is clear. Suppose then that the result holds for $j \in \{1, \ldots, i\}$, and consider the $i+1$-th execution of the cycle.

To prove that $\psi_{ROM,i+1}$ satisfies $\approx_{W(\psi_{ROM,i+1})}$, let k, k' be indexes such that $t_k \approx_{W(\psi_{ROM,i+1})} t_{k'}$. Note that, if $i+1 \in \{k, k'\}$, then lines 5 and 9 of the algorithm description imply the result. There are two cases:

(1) $t_k \not\approx_{W(\psi_{ROM,i})} t_{k'}$;

(2) $t_k \approx_{W(\psi_{ROM,i})} t_{k'}$.

In case (1), Lemmas 7 and 8 imply that $i+1 \in \{k, k'\}$, by noting that $\tau^*(t_k)$ and $\tau^*(t_{k'})$ do not depend on t_{i+1}. Therefore, the result holds.

In case (2), we have either $i+1 \in \{k, k'\}$, and the result holds as before, or $i+1 \notin \{k, k'\}$, and the result is implied by the induction hypothesis (since $\psi_{ROM,i+1}(t_m) = \psi_{ROM,i}(t_m)$ whenever $1 \leq m \leq i$).

Furthermore, we have $psub(t_{i+1}) \subseteq K_i$ (because $\prec$ is a subterm-compatible order), and the induction hypothesis yields

$$sub[K_{i+1}] = sub[K_i] \cup \{t_{i+1}\} = K_i \cup \{t_{i+1}\} = K_{i+1}.$$

It remains to prove that $\psi_{ROM,i+1}$ satisfies exactly one selection function $\iota \in I_{\mathcal{S}}(K_{i+1})$. By Lemma 19, it is sufficient to show that $\psi_{ROM,i+1}$ satisfies some $\iota \in I_{\mathcal{S}}(K_{i+1})$. By the induction hypothesis, there exists $\iota \in I_{\mathcal{S}}(K_i)$ such that ψ_{ROM} satisfies ι. Let

$$t_{i+1} = f_{i+1}(t'_{i+1,1}, \ldots, t'_{i+1,k}).$$

If

$$(\psi_{ROM,i}(t'_{i+1,1}), \ldots, \psi_{ROM,i}(t'_{i+1,k})) \notin dom_{\mathcal{S}}(f_{i+1}),$$

then $\psi_{ROM,i+1}(t_{i+1})$ will be sampled to $\perp$, and thus $\psi_{ROM,i+1}$ satisfies the selection function $\iota' \in I_{\mathcal{S}}(K_{i+1})$ given by

$$\iota' = \iota \cup \{t_{i+1} \mapsto \perp\}\,.$$

Otherwise, there is $ps \in PS_{f_{i+1}}$ such that

$$(\psi_{ROM,i}(t'_{i+1,1}), \ldots, \psi_{ROM,i}(t'_{i+1,k})) \in [\![dom(ps)]\!]\,.$$

Let $\iota' = \iota \cup \{t_{i+1} \mapsto ps\}$; we have $\iota' \in I_{\mathcal{S}}(K_{i+1})$, and it is sufficient to show that $\psi_{ROM,i+1}$ satisfies ι'. Let $t' = f'(t'_1, \ldots, t'_{k'})$ be the minimal (with respect to $\prec$) term in K_{i+1} such that $t' \approx_{P(\psi_{ROM,i})} t$ and

$$(\psi_{ROM,i}(t'_1), \ldots, \psi_{ROM,i}(t'_{k'})) \in dom_{\mathcal{S}}(f')$$

(note that we may have $t' = t_{i+1}$). Then, we have

$$\psi_{ROM,i+1}(t_{i+1}) = \psi_{ROM,i}(t'),$$

and $\psi_{ROM,i}(t')$ is sampled from $[\![ran(\iota(t'))]\!]$. The compatibility condition implies that, for all $\iota'' \in I_{\mathcal{S}}(K_{i+1})$ and all $t'' \in K_{i+1}$,

$$[\![ran(\iota''(t'))]\!] \subseteq [\![ran(\iota''(t''))]\!] .$$

It follows that

$$[\![ran(\iota'(t'))]\!] \subseteq [\![ran(\iota'(t_{i+1})]\!] ;$$

therefore,

$$\psi_{ROM,i+1}(t_{i+1}) = \psi_{ROM,i}(t') \in [\![ran(\iota'(t'))]\!] \subseteq [\![ran(\iota'(t_{i+1}))]\!] ,$$

and we conclude that $\psi_{ROM,i+1}$ satisfies ι'.

□

In the following, if $t \in sub[K]$ and $\prec$ is a subterm-compatible order, we denote by $\widehat{t}^{K,\prec}$ the random variable representing the output $\psi_{ROM}(t)$ of Algorithm 9 when executed on input K using the order $\prec$.

Lemma 21. *Assume the following:*

- K *is a finite set of terms;*
- $\prec$ *is a subterm-compatible order;*
- $\psi \colon sub[K] \to \mathcal{B}^*_\perp$ *satisfies* $\approx_{W(\psi)}$*;*
- $\iota \in I_{\mathcal{S}}(K)$ *and* ψ *satisfies* ι*;*
- $t' = f(t'_1, \ldots, t'_n)$ *is a term such that* $t' \notin K$*,* $psub(t') \subseteq sub[K]$*, and* $t \prec t'$ *for all* $t \in K$*;*
- $b \in \mathcal{B}^*_\perp$*;*
- $K' = K \cup \{t'\}$*;*
- $\psi' = \psi \cup \{t' \mapsto b\}$*;*
- τ *is some* $W(\psi')$*-renaming;*
- τ^+ *is the function output by Algorithm 10 on the input* $(K', W(\psi'), \prec, \tau)$*.*

Define

$$P[K, \psi', t'] = P[\widehat{t'}^{K',\prec} = b \mid \widehat{t}^{K',\prec} = \psi(t) \text{ for all } t \in sub[K]].$$

If there is no $ps \in PS$ *such that*

$$head(ps) = head(t') \quad \wedge \quad (\psi'(t'_1), \ldots, \psi'(t'_n)) \in dom(ps), \tag{C.1}$$

then

$$P[K, \psi', t'] = \begin{cases} 1 & \textit{if } b = \bot \\ 0 & \textit{otherwise} \end{cases} .$$

Otherwise, letting ps be the only property statement satisfying condition (C.1), we have:

- *if there is $t \in sub[K]$ such that either (1) $tau^+(t) = \tau^+(t')$, or (2) $\tau^+(t') \in \mathcal{N}^+$ and $\tau^*(t) = \tau^+(t')$, then*

$$P[K, \psi', t'] = \begin{cases} 1 & \textit{if } \psi(t) = b \\ 0 & \textit{otherwise} \end{cases} .$$

- *if such a t does not exist, then*

$$P[K, \psi', t'] = \begin{cases} 1/|\,[\![ran(ps)]\!]\,| & \textit{if } b \in [\![ran(ps)]\!] \\ 0 & \textit{otherwise} \end{cases} .$$

Proof. Consider an execution of Algorithm 9 on input $K \cup \{t'\}$ using the order $\prec$. Then, $P[K, \psi', t']$ corresponds to the probability that such an execution outputs a function $\psi_{ROM} = \psi'$ given that, before the last step of the execution (when $\psi_{ROM}(t')$ is sampled), we have $\psi_{ROM} = \psi$ (corresponding to the sampling of the terms in $sub[K]$). It follows from the definition of the algorithm that, if there is no $ps \in PS$ such that

$$head(ps) = head(t') \qquad \text{and} \qquad (\psi'(t'_1), \ldots, \psi'(t'_n)) \in dom(ps),$$

then $\psi_{ROM}(t')$ is sampled to $\bot$, and therefore

$$P[K, \psi', t'] = \begin{cases} 1 & \text{if } b = \bot \\ 0 & \text{otherwise.} \end{cases} .$$

Suppose then that there is $ps \in PS$ such that condition (C.1) is satisfied. If there is $t \in sub[K]$ such that either

(1) $\tau^+(t) = \tau^+(t')$, or

(2) $\tau^+(t') \in \mathcal{N}^+$ and $\tau^*(t) = \tau^+(t')$,

we have $t \approx_{W(\psi)} t'$. Therefore, $\psi_{ROM}(t')$ is sampled as $\psi_{ROM}(t') = \psi_{ROM}(t)$, and we have

$$P[K, \psi', t'] = \begin{cases} 1 & \text{if } \psi(t) = b \\ 0 & \text{otherwise} \end{cases} .$$

Now we prove that, for any t, if $t \approx_{W(\psi)} t'$, then either (1) or (2) holds. If $\tau^+(t') \in T^W_\Sigma$, then there exists a subterm $s \in sub[K]$ of t' such that

$$s \in T^W_\Sigma, \qquad t' \approx_{W(\psi)} s \quad \text{and} \quad \tau^*(s) = \tau^*(t');$$

thus, (1) holds. On the other hand, if $\tau^+(t') \notin T_\Sigma^W$, then $\tau^*(t') = \tau^+(t')$ (since we have $head(t'\downarrow) = head(t)$ whenever $t \approx_P t'$ and $t \in T_\Sigma^W$). By Lemma 7, if $t \approx_{W(\psi)} t'$, we have $\tau^*(t) = \tau^*(t')$. It follows that $\tau^*(t) \notin T_\Sigma^W$, and (2) holds since

$$\tau^+(t) = \tau^*(t) = \tau^*(t') = \tau^+(t').$$

We conclude that if neither (1) nor (2) hold for t, then $t \not\approx_{W(\psi)} t'$, and if neither condition hold for any $t \in sub[K]$, then there is no $t \in dom(\psi)$ such that $t \approx_{W(\psi)} t'$. In this case, $\psi_{ROM}(t')$ is sampled with uniform probability distribution from $[\![ran(ps)]\!]$, according to the algorithm description, and it follows that

$$P[K, \psi', t'] = \begin{cases} 1/|\,[\![ran(ps)]\!]\,| & \text{if } b \in [\![ran(ps)]\!] \\ 0 & \text{otherwise} \end{cases}.$$

This concludes the proof of the lemma.

□

If ι is any selection function, we adopt the following conventions:

- $ran(\bot) = \bot$;
- $[\![\bot]\!] = \{\bot\}$;
- $\iota(\bot) = \bot$.

Note that $[\![ran(\iota(\bot))]\!] = \{\bot\}$.

Corollary 2. *Let K, $\prec$, ψ and ι be as in the statement of Lemma 21. Let τ be a $W(\psi)$-renaming and τ^+ be as output by Algorithm 10 on the input $(K, W(\psi), \prec, \tau)$.*

Let

$$\tau_K \colon \tau^+[sub[K]] \to sub[K]$$

be such that, for each $t^+ \in \tau^+[sub[K]]$, $\tau_K(t^+)$ is the least $t \in sub[K]$ such that $\tau^+(t) = t^+$ and $\iota(t) \neq \bot$ if such a t exists, and $\bot$ otherwise.

Let ψ_{ROM} be the function output by Algorithm 9 on input K and using the subterm-compatible order $\prec$.

Then,

$$P[\psi_{ROM} = \psi] = \prod_{t^+ \in \tau^+[sub[K]] \setminus \mathcal{N}^+} \frac{1}{|\,[\![ran(\iota(\tau_K(t^+)))]\!]\,|}.$$

Proof. Let $t_1, \ldots, t_n$ be such that $dom(\psi) = \{t_1, \ldots, t_n\}$ and $t_1 \prec \ldots \prec t_n$. We have

$$\begin{array}{ll} & P[\psi_{ROM} = \psi] \\ = & P[\psi_{ROM}(t_1) = \psi(t_1), \ldots, \psi_{ROM}(t_n) = \psi(t_n)] \\ = & \prod_{j=1}^n P[\widehat{t_j}^{K,\prec} = \psi(t_j) \mid \widehat{t_1}^{K,\prec} = \psi(t_1), \ldots, \widehat{t_{j-1}}^{K,\prec} = \psi(t_{j-1})] \end{array} \tag{C.2}$$

By Lemma 21, if ψ does not satisfy $\approx_{\mathcal{R}}$ or there is no $\iota \in I_{\mathcal{S}}(K)$ such that ψ satisfies ι, then $P[\psi_{\mathit{ROM}} = \psi] = 0$ (because one of the factors in the product is 0). Therefore, it is sufficient to consider the case that ψ satisfies $\approx_R$ and there is $\iota \in I_{\mathcal{S}}(K)$ such that ψ satisfies ι. Then, for each $j \in \{1, \ldots, n\}$, we have either

(1) there is $k < j$ such that either

   (a) $\tau^+(t_j) = \tau^+(t_k)$, or
   (b) $\tau^+(t_j) \in \mathcal{N}^+$ and $\tau^*(t_k) = \tau^+(t_j)$,

   or

(2) there is no such k.

In case (1), $\psi_{\mathit{ROM}}(t_j)$ is sampled as $\psi_{\mathit{ROM}}(t_j)$, and

$$P[\widehat{t_j}^K = \psi(t_j) \mid \widehat{t_1}^K = \psi_{\mathit{ROM}}(t_1), \ldots, \widehat{t_{j-1}}^K = \psi_{\mathit{ROM}}(t_{j-1})] = 1.$$

In case (2), we have

$$P[\widehat{t_j}^K = \psi(t_j) \mid \widehat{t_1}^K = \psi_{\mathit{ROM}}(t_1), \ldots, \widehat{t_{j-1}}^K = \psi_{\mathit{ROM}}(t_{j-1})]$$

$$= \frac{1}{|\left[\!\left[ran(\iota(t_{i(\prec,j)}))\right]\!\right]|}.$$

The result then follows by combining equation (C.2) and Lemma 21.

□

Lemma 22. *Let K, $\prec$, ψ, ι, τ and τ^+ be as in the conditions of Corollary 2, and let $\prec'$ be another subterm-compatible order. Define the functions*

$$\tau_K, \tau'_K \colon \tau^+[sub[K]] \to T_\Sigma(\mathcal{N}^+)$$

such that, for each $t^+ \in \tau^+[sub[K]]$:

- *$\tau_K[t^+]$ is the least term t with respect to $\prec$ such that $\tau^+(t) = t^+$ and $\iota(t) \neq \bot$ if such a t exists, and $\bot$ otherwise;*
- *analogously, $\tau'_K[t^+]$ is the least term t with respect to $\prec'$ such that $\tau^+(t) = t^+$ and $\iota(t) \neq \bot$ if such a t exists, and $\bot$ otherwise.*

Then, for all $t^+ \in \tau^+[sub[K]] \setminus \mathcal{N}^+$,

$$\left[\!\left[ran(\iota(\tau_K(t^+)))\right]\!\right] = \left[\!\left[ran(\iota(\tau_{K'}(t^+)))\right]\!\right].$$

Proof. Let $t^+ \in \tau^+[sub[K]] \setminus \mathcal{N}^+$. If $\iota(t) = \bot$ for all t such that $\tau^+(t) = t^+$, the result holds since

$$[\![ran(\iota(\tau_K(t^+)))]\!] = [\![ran(\iota(\tau_{K'}(t^+)))]\!] = \{\bot\}\,.$$

Suppose then that this is not the case. By the compatibility condition, there are $t^K, t^{K'} \in sub(t^+)$ such that $t^K \approx_{P|_{psub(t^+)}} t^+$, $t^{K'} \approx_{P|_{psub(t^+)}} t^+$, and, for all t' such that $t' \approx_{P|_{psub(t^+)}} t^+$, we have

$$[\![ran(\iota(t^K))]\!] \subseteq [\![ran(\iota(t'))]\!] \quad \text{and} \quad \left[\!\!\left[ran(\iota(t^{K'}))\right]\!\!\right] \subseteq [\![ran(\iota(t'))]\!]\,.$$

Because $t^K \approx_{P|_{psub(t^+)}} \tau_K(t^+)$ and $t^+ \notin \mathcal{N}^+$, we have $\tau^+(t^K) = t^+$, and thus $t^K = \tau_K(t^+)$ (since $\prec, \prec'$ are subterm-compatible orders). Analogously, we have $t^{K'} = \tau_{K'}(t^+)$. We conclude that

$$[\![ran(\iota(\tau_{K'}(t^+)))]\!] \subseteq [\![ran(\iota(\tau_K(t^+)))]\!]$$

and

$$[\![ran(\iota(\tau_K(t^+)))]\!] \subseteq [\![ran(\iota(\tau_{K'}(t^+)))]\!]\,,$$

and the result follows.

□

Proof (Theorem 10)**.** Let $K = dom(\lambda)$, $K' = dom(\lambda')$. Let $t_1, \ldots, t_n$ be such that $sub[K \cap K'] = \{t_1, \ldots, t_n\}$. Because $\Omega_\lambda = \Omega_{\lambda'}$, we must have

- $\lambda(t) = \lambda'(t)$ for all $t \in K \cap K'$,
- $\lambda(t) = \mathcal{B}^*_\bot$ for all $t \in K \setminus K'$, and
- $\lambda'(t) = \mathcal{B}^*_\bot$ for all $t \in K' \setminus K$.

For simplicity, we will write $\widehat{t}^K$ (respectively $\widehat{t}^{K'}$) for the random variable $\widehat{t}^{K,\prec}$ (respectively $\widehat{t}^{K',\prec'}$). The probability that Algorithm 9 on input K and $\prec$ outputs a function $\psi_{\textit{ROM}}$ such that $\psi_{\textit{ROM}}(t) \in \lambda(t)$ for all $t \in K$ depends only on the values of $\psi_{\textit{ROM}}(t')$ for $t' \in sub[K \cap K']$, and the same statement is valid for executing Algorithm 9 on input K' and $\prec'$. Therefore, it is sufficient to prove that, whenever $b_1, \ldots, b_n \in \mathcal{B}^*_\bot$,

$$P[\widehat{t_1}^K = b_1, \ldots, \widehat{t_n}^K = b_n] = P[\widehat{t_1}^{K'} = b_1, \ldots, \widehat{t_n}^{K'} = b_n].$$

Let $\tau_K \colon \tau^+[sub[K]] \to sub[K]$ be such that, for each $t^+ \in \tau^+[sub[K]]$, $\tau_K(t^+)$ is the least $t \in sub[K]$ such that $\tau^+(t) = t^+$, and define $\tau_{K'}$ analogously.

Corollary 2 implies that

$$P[\widehat{t_1}^K = b_1, \ldots, \widehat{t_n}^K = b_n] = \prod_{t^+ \in \tau^+[sub[K]] \setminus \mathcal{N}^+} \frac{1}{|\,[\![ran(\iota(\tau_K(t^+)))]\!]\,|}$$

and

$$P[\widehat{t_1}^{K'} = b_1, \ldots, \widehat{t_n}^{K'} = b_n] = \prod_{t^+ \in \tau^+[sub[K]] \setminus \mathcal{N}^+} \frac{1}{|\,[\![ran(\iota(\tau_{K'}(t^+)))]\!]\,|}.$$

Lemma 22 then implies the result.

□

Lemma 23. *Let K be a finite set of terms and $\prec$ be a subterm-compatible order. For each $t \in sub[K]$, there exists a finite set $supp_{PS}(t)$ such that, if ψ_{ROM} is a possible output of Algorithm 9) on input K and using the order $\prec$, then $\psi_{ROM}(t) \in supp_{PS}(t)$.*

Proof. Let $t_1, \ldots, t_n$ be such that $sub[K] = \{t_1, \ldots, t_n\}$ and $t_1 \prec \ldots \prec t_n$. We prove the result for all t_i such that $i \in \{1, \ldots, n\}$ by induction on i. For $i = 0$, we have $t_i \in \Sigma_0$ and, at the point of the execution of the Algorithm in which $\psi_{ROM}(t_0)$ is sampled, we have $dom(\psi_{ROM}) = \emptyset$. Thus, t_0 is either sampled as $\perp$ or sampled from $[\![ran(ps)]\!]$, which is finite by our definition of property statement and interpretation function. Now, suppose that terms $t_1, \ldots, t_k$ have been sampled, so that $dom(\psi_{ROM}) = \{t_1, \ldots, t_k\}$, and consider the point of the execution of the algorithm in which $\psi_{ROM}(t_{k+1})$ is sampled. Let $f \in \Sigma_{n'}$ and $t'_1, \ldots, t'_{n'}$ be such that $t_{k+1} = f(t'_1, \ldots, t'_{n'})$. By the induction hypothesis, for each $i \in \{1, \ldots, k\}$, there exists a finite set $supp_{PS}(t_i)$ such that, for any execution of the algorithm, $\psi_{ROM}(t_i) \in supp_{PS}(t_i)$. We have

$$(\psi_{ROM}(t'_1), \ldots, \psi_{ROM}(t'_{n'})) \in supp_{PS}(t'_1), \ldots, supp_{PS}(t'_{n'}));$$

therefore, there exists a finite set $PS_{k+1} \subseteq PS_f$ of property statements such that, for each possible ψ_{ROM}, there is exactly one $ps \in PS_{k+1}$ such that

$$(\psi_{ROM}(t'_1), \ldots, \psi_{ROM}(t'_{n'})) \in [\![dom(ps)]\!].$$

Then, t_i is either sampled to $\perp$, or to $\psi_{ROM}(\psi_{ROM}(t'_i))$ for some $i \in \{1, \ldots, k\}$, or to some element of $\bigcup_{ps \in PS_{k+1}} [\![ran(ps)]\!]$. This is a union of finitely many finite sets; thus, we can choose $supp_{PS}(t_{k+1})$ to be this set.

□

Lemma 24. *The set Ω_Λ is a semi-ring of sets.*

Proof. If t_1 is any term and $\lambda = \{t_1 \mapsto \emptyset\}$, then $\Omega_\lambda = \emptyset$. Thus, $\emptyset \in \Omega_\Lambda$.

Let

$$\lambda = \{t_i \mapsto B_i \mid i \in \{1, \ldots, n\}\},$$
$$\lambda' = \{t'_i \mapsto B'_i \mid i \in \{1, \ldots, n'\}\},$$

and let $t''_1, \ldots, t''_{n''}$ be such that

$$dom(\lambda) \cup dom(\lambda') = \{t''_1, \ldots, t''_{n''}\}.$$

There are sets $C_1, \ldots, C_{n''}, C'_1, \ldots, C'_{n''}$ such that, choosing

$$\lambda_* = \left\{t''_i \mapsto C_i \mid i \in \left\{1, \ldots, n''\right\}\right\}$$

and

$$\lambda'_* = \left\{t''_i \mapsto C'_i \mid i \in \left\{1, \ldots, n''\right\}\right\},$$

we have $\Omega_{\lambda_*} = \Omega_{\lambda'_*}$. These sets can be obtained by simply choosing $C_i = \lambda(t_i)$ if $t_i \in dom(\lambda)$ and $C_i = \mathcal{B}^*_\perp$ otherwise and, analogously, $C'_i = \lambda'(t_i)$ if $t_i \in dom(\lambda')$ and $C'_i = \mathcal{B}^*_\perp$ otherwise. We have that

$$\Omega_\lambda \cap \Omega_{\lambda'} = \Omega_{\lambda_*} \cap \Omega_{\lambda'_*} = \Omega_{\lambda''_*},$$

where

$$\lambda'' = \left\{t_i \mapsto C_i \cap C'_i \mid i \in \left\{1, \ldots, n''\right\}\right\}.$$

Thus, Ω_Λ is closed for intersections.

For each $i \in \{1, \ldots, n''\}$, let $C^0_i = C_i \cap C'_i$ and $C^1_i = C_i \setminus C'_i$, and consider the set $\Lambda(\lambda, \lambda')$ of functions $\lambda''\colon \left\{t''_1, \ldots, t''_{n''}\right\} \to \mathcal{B}^*_\perp$ such that, for each $i \in \{1, \ldots, n''\}$, $\lambda''(t''_i)$ is either C^0_i or C^1_i. Let λ''_0 be the element of $\Lambda(\lambda, \lambda')$ such that, for each $i \in \{1, \ldots, n''\}$, $\lambda''_0(t''_i) = C^0_i$.

We have $\Omega_{\lambda''} \in \Omega_\Lambda$ for all $\lambda'' \in \Lambda(\lambda, \lambda')$, $\Omega_\lambda = \biguplus_{\lambda'' \in \Lambda(\lambda,\lambda')} \Omega_{\lambda''}$, and $\Omega_\lambda \cap \Omega_{\lambda'} = \Omega_{\lambda''_0}$.

We conclude that

$$\Omega_\lambda \setminus \Omega_{\lambda'} = \biguplus_{\lambda'' \in \Lambda(\lambda,\lambda') \setminus \left\{\lambda''_0\right\}} \Omega_{\lambda''}$$

is a finite, disjoint union of elements in Ω_Λ. Thus, Ω_Λ is a semi-ring of sets.

□

Lemma 25. *Suppose that $\mu(\emptyset) = 0$, $\mu(\mathbf{\Omega}) = 1$ and, whenever $\lambda_1, \ldots, \lambda_n \in \Lambda$ are such that $\Omega_{\lambda_1}, \ldots, \Omega_{\lambda_n}$ are disjoint sets such that $\biguplus_{i=1}^n \Omega_{\lambda_i} = \Omega_\lambda$ for some $\lambda \in \Lambda$, we have*

$$\mu(\Omega_\lambda) = \sum_{i=1}^{n} \mu(\Omega_{\lambda_i}).$$

Then, there is a unique extension of μ to $\mathcal{F}$ that is a probability measure.

Proof. By Lemma 24 and Caratheodory's extension theorem, it is sufficient to show that, if $\lambda_i \in \Lambda$ for each $i \in \mathbb{N}$ are such that $\Omega_{\lambda_i} \cap \Omega_{\lambda_j} = \emptyset$ whenever $i \neq j$ and there is $\lambda \in \Lambda$ such that $\biguplus_{i \in \mathbb{N}} \Omega_{\lambda_i} = \Omega_\lambda$, then there are finitely many indexes $i_1, \ldots, i_n \in \mathbb{N}$ such that

$$\Omega_\lambda = \biguplus_{j=1}^{n} \Omega_{\lambda_{i_j}}.$$

For each $i \in \mathbb{N}$, let us consider the number k_i and, for each $j \in \{1, \ldots, k_i\}$, the terms t_i^j and the sets of bitstrings B_i^j such that

$$\lambda_i = \left\{ t_i^j \mapsto B_i^j \mid j \in \{1, \ldots, k_i\} \right\}.$$

For each $i \in \mathbb{N}$ and each $j \in \{1, \ldots, k_i\}$, let

$$\lambda_i^{PS} = \left\{ t_i^j \mapsto B_i^j \cap supp_{PS}(t_i^j) \right\},$$

where $supp_{PS}$ is as in Lemma 23. Analogously, let $k \in \mathbb{N}$, $t^1, \ldots, t^k \in T_\Sigma$, and $B^1, \ldots, B^k$ be such that

$$\lambda = \left\{ t^j \mapsto B^j \mid j \in \{1, \ldots, k\} \right\}.$$

Finally, define

$$\lambda^{PS} = \left\{ t^j \mapsto B^j \cap supp_{PS}(t^j) \right\}.$$

Lemma 23 implies that $B_i^j \cap supp_{PS}(t_i^j)$ is finite for all i and all j, and (together with the definition of μ),

$$\mu(\Omega_{\lambda_i}) = \mu(\Omega_{\lambda_i^{PS}})$$

and

$$\mu(\Omega_\lambda) = \mu(\Omega_{\lambda^{PS}}).$$

Thus, we may assume without loss of generality that all the sets B_i^j are such that

$$B_i^j \subseteq supp_{PS}(t_i^j).$$

For each $t \in T_\Sigma$, consider the topological space $supp_{PS}(t)$ where all subsets are open. This space is finite and, therefore, it is trivially compact. Now, consider the topological space

$$F = \{\omega : T_\Sigma \to \mathcal{B}_\perp^* \mid \omega(t) \in supp_{PS}(t) \text{ for all } t\}.$$

F is the Cartesian product of the topological spaces associated to each term t. The open sets in this topological space with the product topology are precisely the sets Ω_λ for functions $\lambda \in \Omega_\Lambda$ such that, for each $t \in dom(\lambda)$, $\lambda(t) \subseteq supp_{PS}(t)$. By Tychonoff's theorem, F with the product topology is also a compact space.

Because the open sets of F form a semi-ring (by an argument entirely analogous to the one we used in the proof of Lemma 24), we know that $F \setminus r$ is a finite union of open sets, and thus is also open. We conclude that r is closed. Because F is compact, it follows that r is also compact. Since $\{\Omega_{\lambda_i} \mid i \in \mathbb{N}\}$ is a open cover of Ω_λ, there must be a finite sub-cover — that is, there must be indexes $i_1, \ldots, i_m$ such that

$$\Omega_\lambda = \biguplus_{k=1}^{m} \Omega_{\lambda_{i_k}}.$$

Because the Ω_{λ_i} are disjoint, we conclude that $\Omega_{\lambda_i} = \emptyset$ for all $i \notin \{i_1, \ldots, i_m\}$. The result then follows from

$$\mu(\Omega_\lambda) = \mu(\biguplus_{i\in\mathbb{N}} \Omega_{\lambda_i}) = \mu(\biguplus_{k=1}^{m} \Omega_{\lambda_{i_k}}) = \sum_{k=1}^{m} \mu(\Omega_{\lambda_{j_k}}) = \sum_{i\in\mathbb{N}} \mu(\Omega_{\lambda_j}).$$

□

Proof (Theorem 11)**.** It is trivial to check that $\mu(\emptyset) = 0$ and $\mu(\mathbf{\Omega}) = 1$. By Lemma 25, it is sufficient to prove that, whenever $\Omega_1, \ldots, \Omega_n \in \Omega_\Lambda$ are pairwise disjoint and there is $\lambda \in \Lambda$ such that $\Omega_\lambda = \biguplus_{i=1}^{n} \Omega_i$, then

$$\mu(\Omega_\lambda) = \sum_{i=1}^{n} \mu(\Omega_{\lambda_i}).$$

For each $i \in \{1, \ldots, n\}$, we have $\Omega_{\lambda_i} = \biguplus_{\psi\in\Psi(\lambda_i)} \Omega_\psi$. Whenever $i \neq j$, we have $\Omega_{\lambda_i} \cap \Omega_{\lambda_j} = \emptyset$, and thus also $\Omega_{\psi_i} \neq \Omega_{\psi_j}$ whenever $\psi_i \in \Psi(\lambda_i)$ and $\psi_j \in \Psi(\lambda_j)$. It follows that

$$\biguplus_{i=1}^{n} \biguplus_{\psi\in\Psi(\lambda_i)} \Omega_\psi = \biguplus_{\psi\in\Psi(\lambda)} \Omega_\psi.$$

Now, if λ' is such that $\Omega_{\lambda'} = \biguplus_{\psi\in\Psi(\lambda')} \Omega_\psi$, it is clear that the function ψ_{ROM} output by Algorithm 9 on input $dom(\lambda')$ is such that $\psi_{ROM}(t) \in \lambda'(t)$ for all $t \in K$ if and only if $\psi_{ROM} \in \Psi(\lambda')$. Therefore, it follows that, for all λ,

$$\mu(\Omega_\lambda) = \sum_{\psi\in\Psi(\lambda)} \mu(\Omega_\psi).$$

We obtain

$$\mu(\Omega_\lambda) = \mu(\biguplus_{\psi\in\Psi(\lambda)} \Omega_\psi) = \sum_{\psi\in\Psi(\lambda)} \mu(\Omega_\psi)$$

$$= \sum_{i=1}^{n} \left(\sum_{\psi\in\Psi(\lambda_i)} \mu(\Omega_\psi) \right) = \sum_{i=1}^{n} \mu(\Omega_{\lambda_i}),$$

which proves that μ is a probability measure. It remains to prove that

$$\mu(\Omega_{PS,[\![\cdot]\!],\approx_{\mathcal{R}}}) = 1.$$

Let $f \in \Sigma$ and $t_{n+1} = f(t_1, \ldots, t_n)$. By Lemma 20, any execution of Algorithm 9 on input $\{t_{n+1}\}$ yields a function ψ_{ROM} which satisfies some $\iota \in I_{\mathcal{S}}(sub(t_{n+1}))$. Thus, if

$$(\psi_{ROM}(t_1), \ldots, \psi_{ROM}(t_n)) \notin dom_{\mathcal{S}}(f),$$

then $\psi_{ROM}(t_{n+1}) = \perp$; otherwise, there is some $ps \in PS_f$ such that

$$(\psi_{ROM}(t_1), \ldots, \psi_{ROM}(t_n)) \in [\![dom(ps)]\!],$$

in which case we must have

$$\psi_{ROM}(t_{n+1}) \in [\![ran(ps)]\!] .$$

If $f \in \Sigma_n$, $t_1, \ldots, t_n$ are terms, and $ps \in PS_f$, let us write

$$\Omega_{f,t_1,\ldots,t_n,ps}$$

for the set of $\omega \in \mathbf{\Omega}$ such that

$$(\omega(t_1), \ldots, \omega(t_n)) \in [\![dom(ps)]\!] \qquad \text{and} \qquad \omega(t_{n+1}) \notin [\![ran(ps)]\!] .$$

Analogously, we write

$$\Omega_{f,t_1,\ldots,t_n,\perp}$$

for the set of $\omega \in \mathbf{\Omega}$ such that

$$(\omega(t_1), \ldots, \omega(t_n)) \notin dom_{\mathcal{S}}(f) \qquad \text{and} \qquad \omega(t_{n+1}) \neq \perp.$$

If $\omega \not\models PS$, we must have

$$\omega \in \Omega_{f,t_1,\ldots,t_n,ps} \cup \Omega_{f,t_1,\ldots,t_n,\perp}$$

for some $f \in \Sigma_n$, some terms $t_1, \ldots, t_n$ and some $ps \in PS_f$. We have seen above that

$$\mu(\Omega_{f,t_1,\ldots,t_n,ps}) = \mu(\Omega_{f,t_1,\ldots,t_n,\perp}) = 0$$

for all such $f, t_1, \ldots, t_n, ps$. Since there are only countably many possible choices for $f, t_1, \ldots, t_n, ps$, it follows that

$$\mu\left(\left\{\omega \mid \omega \models_{[\![\cdot]\!]} PS\right\}\right) = 1. \tag{C.3}$$

On the other hand, suppose that $t_1, t_2 \in T_\Sigma$ are terms such that $t_1 \approx_{\mathcal{R}} t_2$, and assume without loss of generality that $t_1 \prec t_2$. Any execution of Algorithm 9 on input $\{t_1, t_2\}$ will sample $\psi_{ROM}(t_2)$ as $\psi_{ROM}(t_1)$ on line 9. For each $b_1, b_2 \in \mathcal{B}^*_\perp$, let

$$\psi(b_1, b_2) = \{t_1 \mapsto b_1, t_2 \mapsto b_2\} .$$

We have that

$$\{\omega \in \mathbf{\Omega} \mid \omega(t_1) \neq \omega(t_2)\} = \bigcup_{b_1 \in \mathcal{B}^*_\perp} \left(\bigcup_{b_2 \in \mathcal{B}^*_\perp \setminus \{b_1\}} \Omega_{\psi(b_1,b_2)} \right)$$

is a set in $\mathcal{F}$, and

$$\mu(\omega \in \mathbf{\Omega} \mid \omega(t_1) \neq \omega(t_2)) = 0.$$

Since there are only countably many choices for t_1 and t_2, it follows that

$$\mu\left(\{\omega \in \mathbf{\Omega} \mid \omega \models \approx_R\}\right) = 1. \tag{C.4}$$

Combining equations (C.3) and (C.4), we conclude that $\mu(\Omega_{\mathcal{S}}) = 1$, as desired.

□

Let K be a set of terms and $\psi \in \Psi_K$. As in Algorithm 9, we will denote by $W(\psi)$ the partition on $sub[dom(\psi)] \cap T^W$ induced by ψ: thus,

$$W(\psi) = P(\psi) \mid_{T^W}, \text{ where } P(\psi) = \left\{\psi^{-1}(b) \mid b \in ran(\psi)\right\}.$$

We say that ψ is a *colliding instantiation* of K if there exist terms $t_1, t_2 \in sub[K]$ such that:

- t_1 is strong (i.e., $t_1 \notin T^W$),
- $t_1 \not\approx_{W(\psi)} t_2$,
- $(\iota(\psi))(t_1) \neq \bot$,
- and $\psi(t_1) = \psi(t_2)$.

We write Ψ_K^{col} for the set of $\psi \in \Psi_K$ that are colliding instantiations of K, and define

$$\Omega^{col}(K) = \bigcup_{\psi \in \Psi_K^{col}} \Omega_\psi, \qquad \overline{\Omega^{col}}(K) = \mathbf{\Omega} \setminus \Omega^{col}(K).$$

Theorem 15. *If K is a finite set of terms and $\prec$ is a subterm-compatible order, then*

(1) for any $\lambda \in \Lambda_K$, $\mu_\eta^{K,\prec}(\Omega_\lambda \cap \overline{\Omega^{col}}(K)) = \mu_{ROM,\eta}(\Omega_\lambda \cap \overline{\Omega^{col}}(K))$;

(2) $\mu_\eta^{K,\prec}(\Omega^{col}(K)) = \mu_{ROM,\eta}(\Omega^{col}(K)) \leq |K|^2 \cdot |I_{\mathcal{S}}(K)| \cdot (1/L)$.

Proof. We prove the two properties separately.

Proof of (1). Let $t_1, \ldots, t_n$ be such that $sub[K] = \{t_1, \ldots, t_n\}$ and $t_1 \prec \ldots \prec t_n$. Let $\widehat{t}^{K,\prec}$ be the random variable representing the output of Algorithm 8 when executed on input K and using the order $\prec$. Let $\widehat{t}$ be the random variable representing the output of Algorithm 9 on input K. It is sufficient to prove the property for the sets Ω_ψ for all $\psi \in \Psi_K$. For all $\psi \in \Psi_K$, we have

$$\mu_{ROM,\eta}^{K,\prec}(\Omega_\psi) = \textstyle\prod_{i=1}^n P[\widehat{t_i}^{K,\prec} = \psi(t_i) \mid j < i \Rightarrow \widehat{t_j}^{K,\prec} = \psi(t_j)] \tag{C.5}$$

and

$$\mu_{ROM,\eta}(\Omega_\psi) = \textstyle\prod_{i=1}^n P[\widehat{t_i} = \psi(t_i) \mid j < i \Rightarrow \widehat{t_j} = \psi(t_j)]. \tag{C.6}$$

If ψ is a colliding instantiation of K, then $\Omega_\psi \cap \overline{\Omega^{col}}(K) = \emptyset$, and the result is true since

$$\mu_\eta^{K,\preceq}(\emptyset) = \mu_{ROM,\eta}(\emptyset) = 0.$$

Suppose then that ψ is not a colliding instantiation of K. Let $i \in \{1, \ldots, n\}$, and suppose that $\psi_{ROM}(t_j)$ has been sampled to b_j for all $j < i$. Because ψ is not a colliding instantiation of K, we have

$$W(\psi \mid_{\{t_1,\ldots,t_{i-1}\}}) = P(\psi \mid_{\{t_1,\ldots,t_{i-1}\}}).$$

Therefore, the steps executed by the two algorithms when sampling $\psi_{ROM}(t_i)$ are exactly the same. It follows that

$$\begin{aligned} P[\widehat{t_i}^{K,\prec} = \psi(t_i) \mid j < i \Rightarrow \widehat{t_j}^{K,\prec} = \psi(t_j)] \\ = P[\widehat{t_i} = \psi(t_i) \mid j < i \Rightarrow \widehat{t_j} = \psi(t_j)] \end{aligned}$$

for all $i \in \{1, \ldots, n\}$. Thus, equations (C.5) and (C.6) imply (1).

Proof of (2). For each $\iota \in I_{\mathcal{S}}(K)$, each $j, k \in \{1, \ldots, n\}$ such that $j \neq k$ and $t_k \in T_\Sigma \setminus T_\Sigma^W$, and each $b \in ran(\iota(t_j))$, let $\Psi^\iota_{j,k,b}$ be the set of ψ such that:

- ψ satisfies ι;
- $t_k \in T_\Sigma \setminus T_\Sigma^W$;
- $ran(\iota(t_k)) \neq \bot$;
- $ran(\iota(t_j)) \neq \bot$;
- and $\psi(t_j) = \psi(t_k)$.

For readability, we will write $r(\iota, i)$ instead of $[\![ran(\iota(t_i))]\!]$ for all $\iota \in I_{\mathcal{S}}(K)$ and all $i \in \{1, \ldots, n\}$. We remark that, for all $\psi \in \Psi_K^{col}$, there are ι, j, k, b such that $\psi \in \Psi^\iota_{j,k,b}$, and $|r(\iota, k)| \geq L$.

Letting $\Psi_{j,k}$ be the set of $\psi \in \Psi_K^{col}$ such that $\psi(t_j) = \psi(t_k)$, we have

$$P[\Psi_{j,k}] \leq \sum_{\iota \in I_{\mathcal{S}}(K)} \left(\sum_{b \in r(\iota,j)} \left(\sum_{\psi \in \Psi^\iota_{j,k,b}} \left(\prod_{i=1}^{n} \frac{1}{|r(\iota, i)|} \right) \right) \right)$$

Note that, for each $\iota \in I_{\mathcal{S}}(K)$, we have

$$|\Psi^\iota_{j,k,b}| \leq \prod_{i=1, i\neq j,k} r(i).$$

Therefore, the previous equation implies

$$P[\Psi_{j,k}] \leq \sum_{\iota \in I_{\mathcal{S}}(K)} \left(|r(\iota, j)| \cdot \left(\prod_{\substack{i=1 \\ i\neq j,k}}^{n} |r(\iota, i)| \right) \cdot \left(\prod_{i=1}^{n} \frac{1}{|r(\iota, i)|} \right) \right) \leq 1/L.$$

It follows that

$$P[\Psi_K^{col}] \leq \sum_{\iota \in I_{\mathcal{S}}(K)} \left(\sum_{j=1}^{n} \left(\sum_{\substack{k=1 \\ k \neq j}}^{n} P[\Psi_{j,k}] \right) \right)$$

$$\leq |K|^2 \sum_{\iota \in I_{\mathcal{S}}(K)} \frac{1}{L} \leq |K|^2 |I_{\mathcal{S}}(K)| \frac{1}{L},$$

concluding the proof.

□

Proof (Theorem 12)**.** Follows from Theorem 15, by taking $\Omega(K) = \overline{\Omega^{col}}(K)$.

□

C.2 Probability Computation

In this section we prove the results of Chapter 5 concerning the probability distribution μ_C. Namely, we show that μ_C indeed yields a probability distribution on $\mathcal{F}$, and that this probability distribution coincides with μ_{ROM}. While μ_{ROM} is defined as the probability distribution of the output of a probabilistic algorithm, the probability distribution μ_C is defined by a mathematical formula, and thus can be used to compute probabilities in our model.

Proof (Theorem 9)**.** Let $t_1, \ldots, t_n$ be such that $sub[K] = \{t_1, \ldots, t_n\}$ and $t_1 \prec \ldots \prec t_n$. For $i \in \{0, \ldots, n\}$, let $K_i = \{t_1, \ldots, t_i\}$. We prove the result for each K_i by induction on i. We have $I_{\mathcal{S}}(K_0) = \emptyset$, and thus the result holds.

Now let $i \in \{1, \ldots, n\}$, and suppose that the result holds for all $j \in \{1, \ldots, i-1\}$. For each $t \in K_{i-1}$, let

$$supp_{i-1}(t) = \bigcup_{\iota \in I_{\mathcal{S}}(K_{i-1})} [\![ran(\iota(t))]\!] .$$

Suppose that $\iota_i \in I_{\mathcal{S}}(K_i)$. For all $\omega \in \mathbf{\Omega}$ such that $\omega \models \iota_i$, we also have $\omega \models \iota_i \mid_{K_{i-1}}$. Thus, $\iota_i \mid_{K_{i-1}} \in I_{\mathcal{S}}(K_i)$ and, letting $t_i = f_i(t'_1, \ldots, t'_k)$, we have $\omega(t'_j) \in supp_{i-1}(t'_j)$ for all $j \in \{1, \ldots, k\}$. The induction hypothesis implies that $supp_{i-1}(t'_j)$ is finite for all j. Therefore, the set $supp_{i-1}(t'_1) \times \ldots \times supp_{i-1}(t'_k)$ is finite and, if $\omega \models \iota_i$, then

$$(\omega(t'_1), \ldots, \omega(t'_k)) \in supp_{i-1}(t'_1) \times \ldots \times supp_{i-1}(t'_k).$$

Since property statements are assumed to be disjoint, it follows that there exists a finite set $P \subseteq PS_{f_i}$ such that, whenever $\omega \models \iota_i$,

$$(\omega(t'_1), \ldots, \omega(t'_k)) \in [\![dom(ps)]\!]$$

for some $ps \in P$. It follows that

$$I_{\mathcal{S}}(K_i) \subseteq I_{\mathcal{S}}(K_{i-1}) \times (P \cup \{\bot\}),$$

and the induction hypothesis implies that $I_{\mathcal{S}}(K_i)$ is finite.

□

For each term $t \in T_\Sigma$, we define the *PS-support of* t $supp_{PS}(t)$ by

$$supp_{PS}(t) = \bigcup_{\iota \in I_{\mathcal{S}}(sub(t))} [\![ran(\iota(t))]\!] .$$

If K is a finite set of terms and $t \in K$, we define

$$supp_K(t) = \bigcup_{\iota \in I_{\mathcal{S}}(K)} [\![ran(\iota(t))]\!] .$$

If $\lambda \in \Lambda$ is such that $dom(\lambda) = K$, we define $\Psi_{\mathcal{S}}(\lambda)$ as the set of all $\psi \colon K \to \mathcal{B}^*_\bot$ such that

$$t \in K \quad \Rightarrow \quad \psi(t) \in \lambda(t) \cap supp_K(t)$$

(contrast with the definition of $\Psi(\lambda)$ in Section 5.1).

Lemma 26. *Suppose that $t \in T_\Sigma$ is some term and K is a finite set of terms such that $t \in K$. $supp_{PS}(t)$ and $supp_K(t)$ are finite.*

If $\lambda \in \Lambda_K$ and $\lambda_{PS} \in \Lambda_K$ is given by $\lambda_{PS}(t) = \lambda(t) \cap supp_{PS}(t)$ for all $t \in K$, we have $\mu_C(\Omega_\lambda) = \mu_C(\Omega_{\lambda_{PS}})$.

Proof. Lemma 9 implies that there are finitely many $\iota \in I_{\mathcal{S}}(K)$; therefore, $supp_{PS}(t)$ is finite.

It is simple to check that, if $\iota \in I_{\mathcal{S}}(K)$, then $\iota \mid_{sub(t)} \in I_{\mathcal{S}}(sub(t))$. Thus, if $\iota \in I_{\mathcal{S}}(K)$, then $[\![ran(\iota(t))]\!] \subseteq supp_{PS}(t)$. For all $\iota \in I_{\mathcal{S}}(K)$ and all $P \in \mathcal{P}^W_{\mathcal{R}}(K)$, we have

$$\left[\!\left[supp_{\lambda,\iota,P}(t)\right]\!\right] \subseteq \left[\!\left[\overline{supp_\iota}(t)\right]\!\right] = \left[\!\left[ran(\iota)\right]\!\right](t) \subseteq supp_{PS}(t),$$

and thus

$$\left[\!\left[supp_{\lambda_{PS},\iota,P}(t)\right]\!\right] = \left[\!\left[supp_{\lambda,\iota,P}(t)\right]\!\right] \cap supp_{PS}(t) = \left[\!\left[supp_{\lambda,\iota,P}(t)\right]\!\right] .$$

It follows that $\Psi^U(\lambda, \iota, P) = \Psi^U(\lambda_{PS}, \iota, P)$ for all $\iota \in I_{\mathcal{S}}(K)$ and all $P \in \mathcal{P}^W_{\mathcal{R}}(K)$. Thus, we have $\mu_C(\Omega_\lambda) = \mu_C(\Omega_{\lambda_{PS}})$, as desired.

□

Lemma 27. *Let $\lambda \in \Lambda_K$, $\psi \in \Psi_{\mathcal{S}}(\lambda)$, $\iota \in I_{\mathcal{S}}(K)$ and $P \in \mathcal{P}^W_{\mathcal{R}}(K)$. If*

- $\iota = \iota(\psi) \neq \bot$;
- $P = W(\psi)$;

- *there exists* $\psi^u \in \Psi^U(\lambda, \iota(\psi), P(\psi))$ *such that, for each* $t \in sub[K] \setminus \mathcal{N}^*$*:*

$$\psi(t) = \begin{cases} \bot & \textit{if } \overline{supp_\iota}(t) = \mathsf{error} \\ \psi^u([t]^*_P) & \textit{otherwise} \end{cases}.$$

then

$$|\Psi^U(\psi, \iota, P)| = 1.$$

Otherwise,

$$|\Psi^U(\psi, \iota, P)| = 0.$$

Proof. We prove the following five propositions:

(1) if $\iota(\psi) = \bot$ or $\iota \neq \iota(\psi)$, then $|\Psi^U(\psi, \iota, P)| = 0$;

(2) if $P \neq W(\psi)$, then $|\Psi^U(\psi, \iota, P)| = 0$;

(3) if there is $\psi^u \in \Psi^U(\psi, \iota, P)$, then $\psi^u \in \Psi^U(\lambda, \iota, P)$ and, for each $t \in sub[K]$:

$$\psi(t) = \begin{cases} \bot & \text{if } \overline{supp_\iota}(t) = \mathsf{error} \\ \psi^u([t]^*_P) & \text{otherwise} \end{cases}.$$

(4) there is at most one $\psi^u \in \Psi^U(\psi, \iota, P)$;

(5) if there is $\psi^u \in \Psi^U(\lambda, \iota, P)$ such that, for each $t \in sub[K]$:

$$\psi(t) = \begin{cases} \bot & \text{if } \overline{supp_\iota}(t) = \mathsf{error} \\ \psi^u([t]^*_P) & \text{otherwise} \end{cases}.$$

then $\psi^u \in \Psi^U(\psi, \iota, P)$.

Properties (4) and (5) show that $|\Psi^U(\psi, \iota, P)| = 1$. Properties (1), (2) and (3) show that $|\Psi^U(\psi, \iota, P)| = 0$ otherwise.

(1) If $\iota \neq \iota(\psi)$ or $\iota(\psi) = \bot$, then one of the following conditions holds:

(1) there is $t \in sub[K]$ such that $\psi(t) \notin [\![ran(\iota(t))]\!]$;

(2) there is $t = f(t_1, \ldots, t_n) \in sub[K]$ such that, letting

$$\iota(t) = (f[T_1, \ldots, T_n] \subseteq T),$$

we have $\psi(t_i) \notin [\![T_i]\!]$ for some $i \in \{1, \ldots, n\}$.

In case (1), we have

$$\{\psi(t)\} \subseteq \left[\!\left[supp_{\psi,\iota,P}([t]^*_P)\right]\!\right] \qquad \text{and} \qquad [\![ran(\iota(t))]\!] \subseteq \left[\!\left[supp_{\psi,\iota,P}([t]^*_P)\right]\!\right].$$

We conclude that $|\left[\!\left[supp_{\psi,\iota,P}(t)\right]\!\right]| = 0$, and thus $|\Psi^U(\psi, \iota, P)| = 0$.

Similarly, in case (2), we have

$$\{\psi(t_i)\} \subseteq \left[\!\left[supp_{\psi,\iota,P}([t_i]^*_P)\right]\!\right] \qquad \text{and} \qquad [\![T_i]\!] \subseteq \left[\!\left[supp_{\psi,\iota,P}([t_i]^*_P)\right]\!\right],$$

and we conclude that $|\left[\!\left[supp_{\psi,\iota,P}(t_i)\right]\!\right]| = 0$, so that $|\Psi^U(\psi, \iota, P)| = 0$.

(2) If $P \neq W(\psi)$, we have $P \mid_{\psi^{-1}(\mathcal{B}^*)} \neq P(\psi) \mid_{\psi^{-1}(\mathcal{B}^*)}$, and there are

$$t, t' \in sub[K] \cap T^W \cap \psi^{-1}(\mathcal{B}^*)$$

such that one of the following two conditions holds:

(1) $t \approx_P t'$ and $\psi(t) \neq \psi(t')$;

(2) $t \not\approx_P t'$ and $\psi(t) = \psi(t')$.

It is sufficient to prove that $\Psi^U(\psi, \iota, P) = \emptyset$.

In case (1), we have

$$\left[\!\left[supp_{\psi,\iota,P}\right]\!\right]([t]^*_P) \subseteq \{\psi(t)\} \qquad \text{and} \qquad \left[\!\left[supp_{\psi,\iota,P}\right]\!\right]([t']^*_P) \subseteq \left\{\psi(t')\right\};$$

since $[t]^*_P = [t']^*_P$ and $\{\psi(t)\} \cap \{\psi(t')\} = \emptyset$, we obtain $\left[\!\left[supp_{\psi,\iota,P}([t']^*_P)\right]\!\right] = \emptyset$, and thus $\Psi^U(\psi, \iota, P) = \emptyset$.

In case (2), we have

$$\left[\!\left[supp_{\psi,\iota,P}\right]\!\right]([t]^*_P) \subseteq \{\psi(t)\} \qquad \text{and} \qquad \left[\!\left[supp_{\psi,\iota,P}\right]\!\right]([t']^*_P) \subseteq \left\{\psi(t')\right\}.$$

Therefore, if $\psi\colon [sub[K]]^*_P \to \mathcal{B}^*_\perp$, we must have $\psi([t]^*_P) = \psi([t']^*_P)$. Since $[t]^*_P, [t']^*_P \in [sub[K] \cap T^W]^*_P$ and $[t]^*_P \neq [t']^*_P$, we again have $\Psi^U(\psi, \iota, P) = \emptyset$.

(3) For each $t \in sub[K]$, we have $supp_{\psi,\iota,P} \subseteq supp_{\lambda,\iota,P}$; therefore,

$$\Psi^U(\psi, \iota, P) \subseteq \Psi^U(\lambda, \iota, P).$$

- We have $\left[\!\left[supp_{\psi,\iota,P}(t)\right]\!\right] \subseteq \left[\!\left[\overline{supp_\iota}(t)\right]\!\right]$; thus,

  $$\overline{supp_\iota}(t) = \mathsf{error} \Rightarrow \left[\!\left[supp_{\psi,\iota,P}\right]\!\right] \subseteq \{\perp\}.$$

  We also have $\left[\!\left[supp_{\psi,\iota,P}(t)\right]\!\right] \subseteq \{\psi(t)\}$; thus,

  $$\psi(t) \neq \perp \Rightarrow \left[\!\left[supp_{\psi,\iota,P}(t)\right]\!\right] = \emptyset.$$

  In this case there can be no function in $\Psi^U(\psi, \iota, P)$, contradicting the hypothesis.

- If $\overline{supp_\iota}(t) \neq \mathsf{error}$, then

  $$\left[\!\left[supp_{\psi,\iota,P}([t]^*_P)\right]\!\right] \subseteq \{\psi(t)\}.$$

  Since $\psi^u \in \Psi^U(\psi, \iota, P)$, we must have $\psi^u([t]^*_P) = \psi_t$.

(4) For each $C \in [sub[K]]^*_P$, either (1) there is $t \in K \cap C$ such that $\overline{supp_\iota}(t) \neq$ error or (2) not. In case (1), we have $[\![supp_{\psi,\iota,P}([t]^*_P)]\!] \subseteq \{\psi(t)\}$. In case (2), then $[\![supp_{\psi,\iota,P}(C)]\!] = \{\bot\}$.

Thus, for all $C \in [sub[K]]^*_P$, $[\![supp_{\psi,\iota,P}(C)]\!]$ is a set with a single element. Since

$$\left(\psi^u \in \Psi^U(\psi,\iota,P) \wedge C \in [sub[K]]^*_P\right) \quad \Rightarrow \quad \psi^u(C) \in [\![supp_{\psi,\iota,P}(C)]\!]\,,$$

it follows that $\Psi^U(\psi,\iota,P)$ has at most one element.

(5) It is sufficient to prove

$$\psi^u([t]^*_P) \in [\![supp_{\psi,\iota,P}([t]^*_P)]\!]$$

for each $t \in sub[K]$. Let $t \in sub[K]$ and $C = [t]^*_P$. We again distinguish two cases: (1) there exists $t' \in C$ such that $\overline{supp_\iota}(t') \neq$ error, or (2) no such t' exists.

In case (1), we have

$$[\![supp_{\psi,\iota,P}(C)]\!] \subseteq [\![supp_{\lambda,\iota,P}(C)]\!] \cap \{\psi(t')\}\,;$$

since $\psi^u(C) = \psi(t')$, it follows that $\psi(t') \in [\![supp_{\lambda,\iota,P}(C)]\!]$. Thus,

$$\psi^u(C) = \{\psi(t')\} \qquad \text{and} \qquad \{\psi(t')\} = [\![supp_{\lambda,\iota,P}(C)]\!]$$

for all $t' \in C$.

In case (2), we have

$$[\![supp_{\lambda,\iota,P}(C)]\!] = [\![supp_{\lambda,\iota,P}(C)]\!] = \{\bot\}\,,$$

and since $\psi^u \in \Psi^U(\lambda,\iota,P)$, we have $\psi^u(C) = \bot$ and $\psi^u(C) \in [\![supp_{\psi,\iota,P}(C)]\!]$.

□

Lemma 28. *Let $\lambda \in \Lambda_K$. We have*

$$\mu_C(\lambda) = \sum_{\psi \in \Psi(\lambda)} \mu_C(\psi).$$

Proof. We have to show that

$$\sum_{\psi \in \Psi(\lambda)} \mu_C(\psi) = \sum_{\iota \in I_{\mathcal{S}}(K)} \left(\sum_{P \in \mathcal{P}^W_{\mathcal{R}}(K)} \frac{|\Psi^U(\lambda,\iota,P)|}{|\Psi^D(\lambda,\iota,P)|} \right) \tag{C.7}$$

We have

$$\mu_C(\psi) = \sum_{\iota \in I_{\mathcal{S}}(K)} \left(\sum_{P \in \mathcal{P}^W_{\mathcal{R}}(K)} \frac{|\Psi^U(\psi,\iota,P)|}{|\Psi^D(\psi,\iota,P)|} \right).$$

Noting that $\Psi^D(\lambda, \iota, P) = \Psi^D(\psi, \iota, P)$, we conclude

$$\sum_{\psi \in \Psi(\lambda)} \mu_C(\psi) = \sum_{\iota \in I(K)} \left(\sum_{P \in \mathcal{P}^W_{\mathcal{R}}(K)} \left(\sum_{\psi \in \Psi(\lambda)} \frac{|\Psi^U(\psi, \iota, P)|}{|\Psi^D(\psi, \iota, P)|} \right) \right). \tag{C.8}$$

By Lemma 27, for each $\psi \in \Psi(\lambda)$, each $\iota \in I_{\mathcal{S}}(K)$, and each $P \in \mathcal{P}^W_{\mathcal{R}}(K)$, we have:

- $|\Psi^U(\psi, \iota, P)| = 1$ if:
  - $\iota = \iota(\psi)$;
  - $P = P(\psi)$;
  - there exists $\psi^u \in \Psi^U(\psi, \iota, P)$ such that, for each $t \in K$:

$$\psi(t) = \begin{cases} \bot & \text{if } \overline{supp_\iota}(t) = \mathsf{error} \\ \psi^u([t]^*_P) & \text{otherwise} \end{cases}.$$

- $|\Psi^U(\psi, \iota, P)| = 0$ otherwise.

It is simple to check that, for each $\psi^u \in \Psi^U(\lambda, \iota, P)$ there is one and only one $\psi \in \Psi(\lambda)$ such that the first condition is satisfied. Thus, for each $\iota \in I(K)$ and each $P \in \mathcal{P}^W_{\mathcal{R}}(K)$, there are $|\Psi^U(\lambda, \iota, P)|$ functions $\psi \in \Psi(\lambda)$ such that $|\Psi^U(\psi, \iota, P)| = 1$, and $|\Psi^U(\psi', \iota, P)| = 0$ for all other $\psi' \in \Psi(\lambda)$. We obtain

$$\sum_{\psi \in \Psi(\lambda)} |\Psi^U(\psi, \iota, P)| = |\Psi^U(\lambda, \iota, P)|.$$

Combining this equality and (C.8), we obtain (C.7), concluding the proof.

□

Lemma 29. *Let $\lambda_i \in \Lambda$ for each $i \in \{1, \ldots, n\}$. Suppose that $\Omega_{\lambda_1}, \ldots, \Omega_{\lambda_n}$ are pairwise disjoint sets, and that $\lambda \in \Lambda$ is such that $\Omega_\lambda = \biguplus_{i=1}^n \Omega_{\lambda_i}$. Then,*

$$\mu_C(\Omega_\lambda) = \sum_{i=1}^n \mu_C(\Omega_{\lambda_i}).$$

Proof. Let $K = \bigcup_{i=1}^n dom(\lambda_i)$. For each i, there is $\lambda'_i \in \Lambda_K$ such that $\Omega_{\lambda'_i} = \Omega_{\lambda_i}$. Furthermore, there is $\lambda' \in \Lambda_K$ such that $\Omega_{\lambda'} = \Omega_\lambda$.

Now we note that

$$\Psi(\lambda') = \biguplus_{i=1}^n \Psi(\lambda'_i),$$

and the result follows from Lemmas 13 and 28:

$$\mu_C(\Omega(\lambda)) = \mu_C(\Omega(\lambda')) = \sum_{\psi \in \Psi(\lambda')} \mu_C(\Omega_\psi)$$

$$= \sum_{i=1}^{n} \left(\sum_{\psi \in \Psi(\lambda'_i)} \mu_C(\Omega_\psi) \right) = \sum_{i=1}^{n} \mu_C(\Omega_{\lambda'_i}) = \sum_{i=1}^{n} \mu_C(\Omega_{\lambda_i}).$$

□

Lemma 30. *Suppose that $\mu_C(\emptyset) = 0$, $\mu_C(\mathbf{\Omega}) = 1$ and, whenever $\lambda_1, \ldots, \lambda_n \in \Lambda$ are such that $\Omega_{\lambda_1}, \ldots, \Omega_{\lambda_n}$ are disjoint sets such that $\biguplus_{i=1}^{n} \Omega_{\lambda_i} = \Omega_\lambda$ for some $\lambda \in \Lambda$, we have*

$$\mu_C(\Omega_\lambda) = \sum_{i=1}^{n} \mu_C(\Omega_{\lambda_i}).$$

Then, there is a unique extension of μ_C to $\mathcal{F}$ that is a probability measure.

Proof. The proof is entirely similar to the proof of Lemma 25. The only difference is that instead of considering $supp_{PS}$ as in Lemma 23, we consider $supp_{PS}$ as in Corollary 26.

□

Proof (Theorem 14) **.** Let $\lambda = \emptyset$. We have $\mathbf{\Omega} = \Omega_\lambda$, and it is trivial to check that $\mu_C(\lambda) = 1$. Moreover, if $\lambda = \{t \mapsto \emptyset\}$ for some term t, we have $\emptyset = \Omega_\lambda$, and $\mu_C(\lambda) = 0$. Therefore, to conclude that μ_C is a probability measure it suffices to combine Lemmas 30 and 29.

Now, μ_C and μ_{ROM} are probability distributions (by Theorems 14 and 11), and thus the set $\Omega_\Psi = \{\Omega_\psi \mid \psi \in \Psi\}$ is a generator of $\mathcal{F}$. Therefore, proving that $\mu_C(\Omega_\psi) = \mu_{ROM}(\Omega_\psi)$ for all $\psi \in \Psi$ is sufficient to prove that $\mu_C = \mu_{ROM}$.

Let $K = sub[dom(\psi)]$ and

$$\psi' = \psi \cup \{t \mapsto \mathcal{B}^*_\perp \mid t \in K \setminus dom(\psi)\}.$$

We have $\Omega_\psi = \Omega_{\psi'}$, and thus $\mu_C(\Omega_\psi) = \mu_C(\Omega_{\psi'})$ and $\mu_{ROM}(\Omega_\psi) = \mu_{ROM}(\Omega_{\psi'})$, by Theorems 13 and 10. Thus, we can assume without loss of generality that $dom(\psi) = sub[dom(\psi)] = K$.

We consider two cases:

(1) either ψ does not satisfy P or there is no $\iota \in I_{\mathcal{S}}(K)$ such that $\psi \models \iota$;

(2) ψ satisfies P and there is $\iota \in I_{\mathcal{S}}(K)$ such that $\psi \models \iota$.

In case (1), Lemmas 20 and 27 imply that $\mu_{ROM}(\psi) = \mu_C(\psi) = 0$ (note that all parcels in the sum that defines μ_C are 0), and the result is proved.

In case (2), we have

$$\mu_C(\Omega_\psi) = \frac{|\Psi^U(\psi, \iota, P)|}{|\Psi^D(\psi, \iota, P)|}$$

and, by Corollary 2,

$$\mu_{ROM}(\Omega_\psi) = \prod_{t^+ \in \tau^+[sub[K]] \setminus \mathcal{N}^+} \frac{1}{|\,[\![ran(\iota(\tau_K(t^+)))]\!]\,|},$$

where τ_K is as in Corollary 2.

By Lemma 27, we have $|\Psi^U(\psi, \iota, P)| = 1$. On the other hand, the definitions of $|\Psi^D(\psi, \iota, P)|$ and $\overline{supp_\iota}$ imply that

$$|\Psi^D(\psi, \iota, P)| = \prod_{C \in [sub[K]]^+_P} |\, [\![\overline{supp_\iota}(C)]\!] \,|$$

$$= \prod_{t^+ \in \tau^+[sub[K]] \setminus \mathcal{N}^+} |\, [\![ran(\iota(\tau_K(t^+)))]\!] \,|.$$

Combining the equations above, we conclude that $\mu_C(\Omega_\psi) = \mu_{ROM}(\Omega_\psi)$, proving the result.

□

Bibliography

[1] *Proceedings of the 23rd IEEE Computer Security Foundations Symposium, CSF 2010, Edinburgh, United Kingdom, July 17-19, 2010*. IEEE Computer Society, 2010.

[2] Implementation of algorithm for computing probabilities, `http://www.infsec.ethz.ch/people/brunoco/prob_rsa.tar.gz`, 2013.

[3] Implementation of the FAST algorithm. `http://www.infsec.ethz.ch/people/brunoco/fast.zip`, 2013.

[4] Martin Abadi, Mathieu Baudet, and Bogdan Warinschi. Guessing attacks and the computational soundness of static equivalence. *Journal of Computer Security*, pages 909–968, December 2010.

[5] Martín Abadi and Véronique Cortier. Deciding knowledge in security protocols under equational theories. *Theor. Comput. Sci.*, 367:2–32, November 2006.

[6] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. POPL '01, pages 104–115, New York, NY, USA, 2001. ACM.

[7] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Inf. Comput.*, 148(1):1–70, 1999.

[8] Martín Abadi and Steve Kremer, editors. *Principles of Security and Trust - Third International Conference, POST 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, volume 8414 of *Lecture Notes in Computer Science*. Springer, 2014.

[9] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *J. Cryptology*, 20(3):395, 2007.

[10] Onur Aciiçmez, Billy Bob Brumley, and Philipp Grabher. New results on instruction cache attacks. In Stefan Mangard and François-Xavier Standaert, editors, *CHES*, volume 6225 of *Lecture Notes in Computer Science*, pages 110–124. Springer, 2010.

[11] Pedro Adão, Gergei Bana, Jonathan Herzog, and Andre Scedrov. Soundness and completeness of formal encryption: The cases of key cycles and partial information leakage. *Journal of Computer Security*, 17(5):737–797, 2009.

[12] Elena Andreeva, Charles Bouillaguet, Pierre-Alain Fouque, Jonathan J. Hoch, John Kelsey, Adi Shamir, and Sébastien Zimmer. Second preimage attacks on dithered hash functions. In Nigel P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 270–288. Springer, 2008.

[13] Myrto Arapinis, Tom Chothia, Eike Ritter, and Mark Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *CSF* [1], pages 107–121.

[14] Myrto Arapinis, Véronique Cortier, Steve Kremer, and Mark Ryan. Practical everlasting privacy. In Basin and Mitchell [33], pages 21–40.

[15] Myrto Arapinis, Jia Liu, Eike Ritter, and Mark Ryan. Stateful applied pi calculus. In Abadi and Kremer [8], pages 22–41.

[16] Alessandro Armando, David Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuellar, Paul Hankes Drielsma, Pierre-Cyrille Heám, Jacopo Mantovani, Sebastian Mödersheim, David von Oheimb, Michaël Rusinowitch, Judson Santiago, Mathieu Turuani, Luca Viganò, and Laurent Vigneron. The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In Kousha Etessami and Sriram K. Rajamani, editors, *Proceedings of the 17th International Conference on Computer Aided Verification (CAV'05)*, volume 3576 of *LNCS*. Springer, 2005.

[17] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, New York, NY, USA, 1998.

[18] Michael Backes, Fabian Bendun, and Dominique Unruh. Computational soundness of symbolic zero-knowledge proofs: Weaker assumptions and mechanized verification. In Basin and Mitchell [33], pages 206–225.

[19] Michael Backes and Peeter Laud. Computationally sound secrecy proofs by mechanized flow analysis. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM Conference on Computer and Communications Security*, pages 370–379. ACM, 2006.

[20] Michael Backes, Ankit Malik, and Dominique Unruh. Computational soundness without protocol restrictions. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM Conference on Computer and Communications Security*, pages 699–711. ACM, 2012.

[21] Michael Backes, Esfandiar Mohammadi, and Tim Ruffing. Computational soundness results for ProVerif - bridging the gap from trace properties to uniformity. In Abadi and Kremer [8], pages 42–62.

[22] Michael Backes and Birgit Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *CSFW*, pages 204–218. IEEE Computer Society, 2004.

[23] Michael Backes, Birgit Pfitzmann, and Andre Scedrov. Key-dependent message security under active attacks - BRSIM/UC-soundness of Dolev-Yao-style encryption with key cycles. *Journal of Computer Security*, 16(5):497–530, 2008.

[24] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A composable cryptographic library with nested operations. In Sushil Jajodia, Vijayalakshmi Atluri, and Trent Jaeger, editors, *ACM Conference on Computer and Communications Security*, pages 220–230. ACM, 2003.

[25] Michael Backes, Birgit Pfitzmann, and Michael Waidner. The reactive simulatability (rsim) framework for asynchronous systems. *Inf. Comput.*, 205(12):1685–1720, 2007.

[26] Gergei Bana, Pedro Adão, and Hideki Sakurada. Computationally complete symbolic attacker in action. In Deepak D'Souza, Telikepalli Kavitha, and Jaikumar Radhakrishnan, editors, *FSTTCS*, volume 18 of *LIPIcs*, pages 546–560. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[27] Gergei Bana and Hubert Comon-Lundh. Towards unconditional soundness: Computationally complete symbolic attacker. In Degano and Guttman [90], pages 189–208.

[28] Gergei Bana, Koji Hasebe, and Mitsuhiro Okada. Computationally complete symbolic attacker and key exchange. In Sadeghi et al. [135], pages 1231–1246.

[29] Gilles Barthe, Juan Manuel Crespo, Benjamin Grégoire, César Kunz, and Santiago Zanella Béguelin. Computer-aided cryptographic proofs. In *3rd International Conference on Interactive Theorem Proving, ITP 2012*, pages 12–27. Springer, 2012.

[30] Gilles Barthe, Marion Daubignard, Bruce M. Kapron, and Yassine Lakhnech. Computational indistinguishability logic. In Ehab Al-Shaer, An-

gelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM Conference on Computer and Communications Security*, pages 375–386. ACM, 2010.

[31] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. Formal certification of code-based cryptographic proofs. In Zhong Shao and Benjamin C. Pierce, editors, *POPL*, pages 90–101. ACM, 2009.

[32] David A. Basin, Cas Cremers, and Simon Meier. Provably repairing the ISO/IEC 9798 standard for entity authentication. *Journal of Computer Security*, 21(6):817–846, 2013.

[33] David A. Basin and John C. Mitchell, editors. *Principles of Security and Trust - Second International Conference, POST 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, volume 7796 of *Lecture Notes in Computer Science*. Springer, 2013.

[34] David A. Basin, Sebastian Mödersheim, and Luca Viganò. An on-the-fly model-checker for security protocol analysis. In Einar Snekkenes and Dieter Gollmann, editors, *ESORICS*, volume 2808 of *Lecture Notes in Computer Science*, pages 253–270. Springer, 2003.

[35] David A. Basin, Sebastian Mödersheim, and Luca Viganò. OFMC: A symbolic model checker for security protocols. *Int. J. Inf. Sec.*, 4(3):181–208, 2005.

[36] Mathieu Baudet. Deciding security of protocols against off-line guessing attacks. In *Proceedings of the 12th ACM conference on Computer and communications security*, CCS '05, pages 16–25, New York, NY, USA, 2005. ACM.

[37] Mathieu Baudet, Véronique Cortier, and Stéphanie Delaune. YAPA: A generic tool for computing intruder knowledge. In Treinen [140], pages 148–163.

[38] Giampaolo Bella. *Formal Correctness of Security Protocols - With 62 Figures and 4 Tables*. Information Security and Cryptography. Springer, 2007.

[39] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *CRYPTO*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, 1993.

[40] Steven M. Bellovin and Michael Merritt. Encrypted Key Exchange: Password-based protocols secure against dictionary attacks. In *IEEE Symposium on Research in Security and Privacy*, pages 72–84, 1992.

[41] Karthikeyan Bhargavan, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, and Pierre-Yves Strub. Implementing TLS with verified cryptographic security. In *IEEE Symposium on Security and Privacy*, pages 445–459. IEEE Computer Society, 2013.

[42] Eli Biham and Adi Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer, 1993.

[43] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In Burton S. Kaliski Jr., editor, *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 1997.

[44] John Black, Phillip Rogaway, and Thomas Shrimpton. Encryption-scheme security in the presence of key-dependent messages. In Kaisa Nyberg and Howard M. Heys, editors, *Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 62–75. Springer, 2002.

[45] Bruno Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Proc. of the 14th IEEE workshop on Computer Security Foundations*, CSFW '01, pages 82–96, Washington, DC, USA, 2001. IEEE Computer Society.

[46] Bruno Blanchet. Automatic proof of strong secrecy for security protocols. In *IEEE Symposium on Security and Privacy*, pages 86–. IEEE Computer Society, 2004.

[47] Bruno Blanchet. A computationally sound mechanized prover for security protocols. *IEEE Trans. Dependable Sec. Comput.*, 5(4):193–207, 2008.

[48] Bruno Blanchet. Security protocol verification: Symbolic and computational models. In Degano and Guttman [90], pages 3–29.

[49] Bruno Blanchet and Cedric Fournet. Automated verification of selected equivalences for security protocols. In *Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science*, pages 331–340, Washington, DC, USA, 2005. IEEE Computer Society.

[50] Bruno Blanchet and David Pointcheval. Automated security proofs with sequences of games. In Cynthia Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 537–554. Springer, 2006.

[51] Florian Böhl, Véronique Cortier, and Bogdan Warinschi. Deduction soundness: prove one, get five for free. In Sadeghi et al. [135], pages 1261–1272.

[52] Dan Boneh, Shai Halevi, Michael Hamburg, and Rafail Ostrovsky. Circular-secure encryption from decision Diffie-Hellman. In David Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 108–125. Springer, 2008.

[53] Stefan Brands and David Chaum. Distance-bounding protocols (extended abstract). In Tor Helleseth, editor, *EUROCRYPT*, volume 765 of *Lecture Notes in Computer Science*, pages 344–359. Springer, 1993.

[54] Emmanuel Bresson, Yassine Lakhnech, Laurent Mazaré, and Bogdan Warinschi. A generalization of ddh with applications to protocol analysis and computational soundness. In Alfred Menezes, editor, *CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 482–499. Springer, 2007.

[55] Mayla Brusò, Konstantinos Chatzikokolakis, and Jerry den Hartog. Formal verification of privacy for RFID systems. In *CSF* [1], pages 75–88.

[56] Sergiu Bursuc and Hubert Comon-Lundh. Protocol security and algebraic properties: Decision results for a bounded number of sessions. In Treinen [140], pages 133–147.

[57] Sergiu Bursuc, Hubert Comon-Lundh, and Stéphanie Delaune. Associative-commutative deducibility constraints. In Wolfgang Thomas and Pascal Weil, editors, *STACS*, volume 4393 of *Lecture Notes in Computer Science*, pages 634–645. Springer, 2007.

[58] Sergiu Bursuc, Hubert Comon-Lundh, and Stéphanie Delaune. Deducibility constraints. In Anupam Datta, editor, *ASIAN*, volume 5913 of *Lecture Notes in Computer Science*, pages 24–38. Springer, 2009.

[59] Jan Camenisch and Ivan Damgard. Verifiable encryption, group encryption, and their applications to separable group signatures and signature sharing schemes. In Tatsuaki Okamoto, editor, *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 331–345. Springer, 2000.

[60] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 93–118. Springer, 2001.

[61] Rohit Chadha, Ştefan Ciobâcă, and Steve Kremer. Automated verification of equivalence properties of cryptographic protocols. In Helmut Seidl, editor, *ESOP*, volume 7211 of *Lecture Notes in Computer Science*, pages 108–127. Springer, 2012.

[62] Vincent Cheval. Apte: An algorithm for proving trace equivalence. In Erika Ábrahám and Klaus Havelund, editors, *TACAS*, volume 8413 of *Lecture Notes in Computer Science*, pages 587–592. Springer, 2014.

[63] Vincent Cheval and Bruno Blanchet. Proving more observational equivalences with ProVerif. In Basin and Mitchell [33], pages 226–246.

[64] Vincent Cheval, Hubert Comon-Lundh, and Stéphanie Delaune. Trace equivalence decision: negative tests and non-determinism. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM Conference on Computer and Communications Security*, pages 321–330. ACM, 2011.

[65] Vincent Cheval, Véronique Cortier, and Stéphanie Delaune. Deciding equivalence-based properties using constraint solving. *Theor. Comput. Sci.*, 492:1–39, 2013.

[66] Yannick Chevalier, Ralf Küsters, Michael Rusinowitch, and Mathieu Turuani. Deciding the security of protocols with Diffie-Hellman exponentiation and products in exponents. pages 124–135. Springer, 2003.

[67] Yannick Chevalier and Michaël Rusinowitch. Decidability of equivalence of symbolic derivations. *J. Autom. Reasoning*, 48(2):263–292, 2012.

[68] Tom Chothia, Simona Orzan, Jun Pang, and Muhammad Torabi Dashti. A framework for automatically checking anonymity with *mu* crl. In Ugo Montanari, Donald Sannella, and Roberto Bruni, editors, *TGC*, volume 4661 of *Lecture Notes in Computer Science*, pages 301–318. Springer, 2006.

[69] Stefan Ciobâca, Stéphanie Delaune, and Steve Kremer. Computing knowledge in security protocols under convergent equational theories. In *CADE*, pages 355–370, 2009.

[70] H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive-or. In *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science*, pages 271–280, Washington, DC, USA, 2003. IEEE Computer Society.

[71] Hubert Comon-Lundh and Véronique Cortier. Computational soundness of observational equivalence. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM Conference on Computer and Communications Security*, pages 109–118. ACM, 2008.

[72] Hubert Comon-Lundh, Véronique Cortier, and Guillaume Scerri. Security proof with dishonest keys. In Degano and Guttman [90], pages 149–168.

[73] Hubert Comon-Lundh, Véronique Cortier, and Guillaume Scerri. Tractable inference systems: An extension with a deducibility predicate. In Maria Paola Bonacina, editor, *CADE*, volume 7898 of *Lecture Notes in Computer Science*, pages 91–108. Springer, 2013.

[74] Hubert Comon-Lundh and Stéphanie Delaune. The finite variant property: How to get rid of some algebraic properties. In Jürgen Giesl, editor, *RTA*, volume 3467 of *Lecture Notes in Computer Science*, pages 294–307. Springer, 2005.

[75] Hubert Comon-Lundh and Ralf Treinen. Easy intruder deductions. In Nachum Dershowitz, editor, *Verification: Theory and Practice*, volume 2772 of *Lecture Notes in Computer Science*, pages 225–242. Springer, 2003.

[76] Bruno Conchinha, David Basin, and Carlos Caleiro. Efficient algorithms for deciding deduction and static equivalence. In *Proc. 7th Int. Workshop on Formal Aspects of Security and Trust (FAST'10)*, 2010.

[77] Bruno Conchinha, David A. Basin, and Carlos Caleiro. FAST: An efficient decision procedure for deduction and static equivalence. In Schmidt-Schauß [137], pages 11–20.

[78] Bruno Conchinha, David A. Basin, and Carlos Caleiro. Symbolic probabilistic analysis of off-line guessing. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS*, volume 8134 of *Lecture Notes in Computer Science*, pages 363–380. Springer, 2013.

[79] Ricardo Corin, Jeroen Doumen, and Sandro Etalle. Analysing password protocol security against off-line dictionary attacks. *Electron. Notes Theor. Comput. Sci.*, 121:47–63, February 2005.

[80] Véronique Cortier and Stéphanie Delaune. Deciding knowledge in security protocols for monoidal equational theories. In *Proceedings of the 14th international conference on Logic for programming, artificial intelligence and reasoning*, LPAR'07, pages 196–210, Berlin, Heidelberg, 2007. Springer-Verlag.

[81] Véronique Cortier and Stéphanie Delaune. A method for proving observational equivalence. In *CSF*, pages 266–276. IEEE Computer Society, 2009.

[82] Véronique Cortier and Stéphanie Delaune. Decidability and combination results for two notions of knowledge in security protocols. *Journal of Automated Reasoning*, 2010. To appear.

[83] Véronique Cortier, Stéphanie Delaune, and Pascal Lafourcade. A survey of algebraic properties used in cryptographic protocols. *J. Comput. Secur.*, 14:1–43, January 2006.

[84] Véronique Cortier, Steve Kremer, Ralf Küsters, and Bogdan Warinschi. Computationally sound symbolic secrecy in the presence of hash functions. In S. Arun-Kumar and Naveen Garg, editors, *FSTTCS*, volume 4337 of *Lecture Notes in Computer Science*, pages 176–187. Springer, 2006.

[85] Véronique Cortier, Steve Kremer, and Bogdan Warinschi. A survey of symbolic methods in computational analysis of cryptographic systems. *J. Autom. Reasoning*, 46(3-4):225–259, 2011.

[86] Véronique Cortier and Bogdan Warinschi. Computationally sound, automated proofs for security protocols. In Shmuel Sagiv, editor, *ESOP*, volume 3444 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2005.

[87] Judicaël Courant, Marion Daubignard, Cristian Ene, Pascal Lafourcade, and Yassine Lakhnech. Automated proofs for asymmetric encryption. *J. Autom. Reasoning*, 46(3-4):261–291, 2011.

[88] Cas J. F. Cremers and Sjouke Mauw. Checking secrecy by means of partial order reduction. In Daniel Amyot and Alan W. Williams, editors, *SAM*, volume 3319 of *Lecture Notes in Computer Science*, pages 171–188. Springer, 2004.

[89] Anupam Datta, Ante Derek, John C. Mitchell, and Bogdan Warinschi. Computationally sound compositional logic for key exchange protocols. In *CSFW*, pages 321–334. IEEE Computer Society, 2006.

[90] Pierpaolo Degano and Joshua D. Guttman, editors. *Principles of Security and Trust - First International Conference, POST 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012, Proceedings*, volume 7215 of *Lecture Notes in Computer Science*. Springer, 2012.

[91] Stéphanie Delaune, Steve Kremer, and Daniel Pasaila. Security protocols, constraint systems, and group theories. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *IJCAR*, volume 7364 of *Lecture Notes in Computer Science*, pages 164–178. Springer, 2012.

[92] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Symbolic bisimulation for the applied pi calculus. In Vikraman Arvind and Sanjiva Prasad, editors, *FSTTCS*, volume 4855 of *Lecture Notes in Computer Science*, pages 133–145. Springer, 2007.

[93] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Verifying privacy-type properties of electronic voting protocols. *J. Comput. Secur.*, 17:435–487, December 2009.

[94] Stéphanie Delaune, Pascal Lafourcade, Denis Lugiez, and Ralf Treinen. Symbolic protocol analysis for monoidal equational theories. *Inf. Comput.*, 206:312–351, February 2008.

[95] Dorothy E. Denning and Giovanni Maria Sacco. Timestamps in key distribution protocols. *Commun. ACM*, 24:533–536, August 1981.

[96] Itai Dinur, Orr Dunkelman, Nathan Keller, and Adi Shamir. Key recovery attacks on 3-round Even-Mansour, 8-step led-128, and full AES2. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT (1)*, volume 8269 of *Lecture Notes in Computer Science*, pages 337–356. Springer, 2013.

[97] Hans Dobbertin, Lars R. Knudsen, and Matthew J. B. Robshaw. The cryptanalysis of the AES - a brief survey. In Hans Dobbertin, Vincent Rijmen, and Aleksandra Sowa, editors, *AES Conference*, volume 3373 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2004.

[98] Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983.

[99] Jannik Dreier, Cristian Ene, Pascal Lafourcade, and Yassine Lakhnech. On unique decomposition of processes in the applied pi-calculus. In Frank Pfenning, editor, *FoSSaCS*, volume 7794 of *Lecture Notes in Computer Science*, pages 50–64. Springer, 2013.

[100] Jannik Dreier, Pascal Lafourcade, and Yassine Lakhnech. Formal verification of e-auction protocols. In Basin and Mitchell [33], pages 247–266.

[101] Luca Durante, Riccardo Sisto, and Adriano Valenzano. Automatic testing equivalence verification of spi calculus specifications. *ACM Trans. Softw. Eng. Methodol.*, 12(2):222–284, 2003.

[102] Santiago Escobar, Catherine Meadows, and José Meseguer. Maude-NPA: Cryptographic protocol analysis modulo equational properties. In Alessandro Aldini, Gilles Barthe, and Roberto Gorrieri, editors, *FOSAD*, volume 5705 of *Lecture Notes in Computer Science*, pages 1–50. Springer, 2007.

[103] Santiago Escobar, Ralf Sasse, and José Meseguer. Folding variant narrowing and optimal variant termination. *J. Log. Algebr. Program.*, 81(7-8):898–928, 2012.

[104] David Galindo, Flavio D. Garcia, and Peter van Rossum. Computational soundness of non-malleable commitments. In Liqun Chen, Yi Mu, and Willy Susilo, editors, *ISPEC*, volume 4991 of *Lecture Notes in Computer Science*, pages 361–376. Springer, 2008.

[105] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[106] Flavio D. Garcia and Peter van Rossum. Sound and complete computational interpretation of symbolic hashes in the standard model. *Theor. Comput. Sci.*, 394(1-2):112–133, 2008.

[107] Daniel Genkin, Adi Shamir, and Eran Tromer. RSA key extraction via low-bandwidth acoustic cryptanalysis. *IACR Cryptology ePrint Archive*, 2013:857, 2013.

[108] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[109] David Gullasch, Endre Bangerter, and Stephan Krenn. Cache games - bringing access-based cache attacks on AES to practice. In *IEEE Symposium on Security and Privacy*, pages 490–505. IEEE Computer Society, 2011.

[110] Shai Halevi. A plausible approach to computer-aided cryptographic proofs. *IACR Cryptology ePrint Archive*, 2005:181, 2005.

[111] Jonathan Herzog. A computational interpretation of Dolev-Yao adversaries. *Theor. Comput. Sci.*, 340(1):57–81, 2005.

[112] Hans Hüttel. Deciding framed bisimilarity. *Electr. Notes Theor. Comput. Sci.*, 68(6):1–18, 2002.

[113] Romain Janvier, Yassine Lakhnech, and Laurent Mazaré. Computational soundness of symbolic analysis for protocols using hash functions. *Electr. Notes Theor. Comput. Sci.*, 186:121–139, 2007.

[114] Boris Köpf and David A. Basin. An information-theoretic model for adaptive side-channel attacks. In *CCS 2007*, pages 286–296. ACM, 2007.

[115] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. On the security of the TLS protocol: A systematic analysis. In Ran Canetti and Juan A. Garay, editors, *CRYPTO (1)*, volume 8042 of *Lecture Notes in Computer Science*, pages 429–448. Springer, 2013.

[116] Steve Kremer and Laurent Mazaré. Computationally sound analysis of protocols using bilinear pairings. *Journal of Computer Security*, 18(6):999–1033, 2010.

[117] Steve Kremer, Mark Ryan, and Ben Smyth. Election verifiability in electronic voting protocols. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *ESORICS*, volume 6345 of *Lecture Notes in Computer Science*, pages 389–404. Springer, 2010.

[118] Ralf Küsters and Max Tuengerthal. Computational soundness for key exchange protocols with symmetric encryption. In Ehab Al-Shaer, Somesh Jha, and Angelos D. Keromytis, editors, *ACM Conference on Computer and Communications Security*, pages 91–100. ACM, 2009.

[119] Sven Laur and Kaisa Nyberg. Efficient mutual data authentication using manually authenticated strings. In David Pointcheval, Yi Mu, and Kefei Chen, editors, *CANS*, volume 4301 of *Lecture Notes in Computer Science*, pages 90–107. Springer, 2006.

[120] Sven Laur and Sylvain Pasini. Sas-based group authentication and key agreement protocols. In Ronald Cramer, editor, *Public Key Cryptography*, volume 4939 of *Lecture Notes in Computer Science*, pages 197–213. Springer, 2008.

[121] Gavin Löwe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In Tiziana Margaria and Bernhard Steffen, editors, *TACAS*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 1996.

[122] Florian Mendel, Tomislav Nad, and Martin Schläffer. Improving local collisions: New attacks on reduced SHA-256. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 262–278. Springer, 2013.

[123] Florian Mendel, Thomas Peyrin, Martin Schläffer, Lei Wang, and Shuang Wu. Improved cryptanalysis of reduced ripemd-160. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT (2)*, volume 8270 of *Lecture Notes in Computer Science*, pages 484–503. Springer, 2013.

[124] Daniele Micciancio and Bogdan Warinschi. Soundness of formal encryption in the presence of active adversaries. In Moni Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 133–151. Springer, 2004.

[125] Jonathan Millen and Vitaly Shmatikov. Symbolic protocol analysis with an abelian group operator or Diffie-Hellman exponentiation. *J. Comput. Secur.*, 13:515–564, May 2005.

[126] Robin Milner. *Communicating and mobile systems - the Pi-calculus*. Cambridge University Press, 1999.

[127] Sebastian Mödersheim, Luca Viganò, and David A. Basin. Constraint differentiation: Search-space reduction for the constraint-based analysis of security protocols. *Journal of Computer Security*, 18(4):575–618, 2010.

[128] Bruno Montalto and Carlos Caleiro. Modeling and reasoning about an attacker with cryptanalytical capabilities. *ENTCS*, 253(3):143–165, 2009.

[129] Jorge Munilla and Alberto Peinado. Distance bounding protocols for RFID enhanced by using void-challenges and analysis in noisy channels. *Wireless Communications and Mobile Computing*, 8(9):1227–1232, 2008.

[130] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, 1978.

[131] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: The case of aes. In David Pointcheval, editor, *CT-RSA*, volume 3860 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2006.

[132] Lawrence C. Paulson. Inductive analysis of the internet protocol TLS. *ACM Trans. Inf. Syst. Secur.*, 2(3):332–351, 1999.

[133] Raphael Chung-Wei Phan and Adi Shamir. Improved related-key attacks on DESX and DESX+. *Cryptologia*, 32(1):13–22, 2008.

[134] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[135] Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors. *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*. ACM, 2013.

[136] Benedikt Schmidt, Simon Meier, Cas J. F. Cremers, and David A. Basin. Automated analysis of Diffie-Hellman protocols and advanced security properties. In Stephen Chong, editor, *CSF*, pages 78–94. IEEE, 2012.

[137] Manfred Schmidt-Schauß, editor. *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications, RTA 2011, May 30 - June 1, 2011, Novi Sad, Serbia*, volume 10 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[138] Marc Stevens, Arjen K. Lenstra, and Benne de Weger. Chosen-prefix collisions for MD5 and applications. *IJACT*, 2(4):322–359, 2012.

[139] Alwen Tiu and Jeremy E. Dawson. Automating open bisimulation checking for the spi calculus. In *CSF* [1], pages 307–321.

[140] Ralf Treinen, editor. *Rewriting Techniques and Applications, 20th International Conference, RTA 2009, Brasília, Brazil, June 29 - July 1, 2009, Proceedings*, volume 5595 of *Lecture Notes in Computer Science*. Springer, 2009.

[141] Ton van Deursen and Sasa Radomirovic. Security of RFID protocols - a case study. *Electr. Notes Theor. Comput. Sci.*, 244:41–52, 2009.

[142] Serge Vaudenay. Secure communications over insecure channels based on short authenticated strings. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 309–326. Springer, 2005.

[143] Bogdan Warinschi. A computational analysis of the Needham-Schröder-(Löwe) protocol. *Journal of Computer Security*, 13(3):565–591, 2005.

[144] P. Zimmermann, A. Johnston, and J. Callas (Ed.). Internet RFC 6189. *The RFC Series and RFC Editor, ISSN 2070-1721*, 2011.

Bruno Conchinha Montalto

ETH Zürich, Information Security Group
CAB F58.1, Universitätstrasse 6, 8092 Zürich
(+41) 44 632 62 79 bruno.conchinha@inf.ethz.ch

Education

Bachelor Degree (BSc) in Applied Mathematics and Computation (2006)
from Technical Higher Education Institute (IST), Lisbon,
with the final mark of 18/20.

Master Degree (MSc) in Mathematics and Applications (2008)
from Technical Higher Education Institute (IST), Lisbon,
with the final mark of 17/20.

PhD Candidate in Computer Science
from ETH Zürich, under Prof. Dr. David Basin
expected to finish in Summer 2014.

Extracurricular Activities

Silver medal in the Portuguese Mathematical Olympiads (2002/2003).

International Mathematical Olympiads (2003).
I was part of the Portuguese team in the 2003 edition of the International Mathematical Olympiads.

Participation in the Marktoberdorf Summer School (2011)
on *Tools for Analysis and Verification of Software Safety and Security*.

Awards and Honors

Two *Novos Talentos em Matemática* ("New Talents in Mathematics") scholarships (2003-2004 and 2004-2005).
These scholarships are awarded every year to 20 Mathematics students in Portugal. Each student can be awarded this scholarship at most twice.

***Prémio Professor Jaime Campos Ferreira* ("Professor Jaime Campos Ferreira Prize")**
(2006-2007).
This prize is awarded every year at IST, Lisbon, "to distinguish academic merit in the area of Mathematics". Each student can win at most once. On the scope of this project I worked with Prof. Dr. Carlos Caleiro (IST) on Security Protocol Analysis.

Three Technical Higher Education Institute Merit Diplomas
(2003-2004, 2004-2005 and 2005-2006).
These diplomas are awarded every year to distinguish the three best students of each degree at IST, Lisbon.

Google Europe Doctoral Fellowship in Computer Security (2011).
Google Europe Doctoral Fellowships are awarded every year to 15-20 doctoral students in European universities in different areas of computer science.

Professional Experience

Teaching Assistant for the Section of Logic and Computation of the Mathematics Department at IST. *September 2006 - September 2007*

SQIG-IT (Security and Quantum Information Group - Institute of Telecommunications) Research Grant for the KLog project. *July 2008 - February 2008*
On the scope of this research grant I continued my work with Prof. Dr. Carlos Caleiro on Security Protocol Analysis.

FCT (Foundation for Science and Technology)
Doctoral Grant SFRH/BD/44204/2008.
February 2009 - January 2013

Software Engineer Intern at Google. *March 2015 - June 2014*
I worked in the Security Team of the Google office in Zürich.

Teaching Experience

Teaching assistant in *Escola Diagonal* ("Diagonal School") (2006)
for the basic course *Recreative Mathematics to be Taken Seriously*, by Prof. Dr. Luís Sanchez (University of Lisbon).

Teaching Assistant at IST, Lisbon (2006-2007)
I was Teaching Assistant for the courses *Teoria da Computação* ("Computation Theory") and *Matemática Discreta* ("Discrete Mathematics").

Teaching assistant in *Escola Diagonal* ("Diagonal School") (2007)
for the advanced course *Equations*, by Prof. Dr. Dmitry Fuchs (University of California, Davis).

Teaching assistant in the *Estágio de Iniciação Científica em Topologia* ("Scientific Initiation in Topology Internship") (2008)
for highschool students, held at IST, Lisbon.

Teaching Assistant at ETH, Zürich (2009-2013)
I was Teaching Assistant for the courses "Computer Science I", "Computer Science II", "Computer Science for Mathematicians and Physicists", and "Information Security".

Computer skills

Design and Implementation of the FAST Tool.
The FAST tool is a C++ procedure for deciding message deducibility and static equivalence, two relevant problems in symbolic security protocol analysis.
It achieves better asymptotic complexity than competing tools and performs up to several orders of magnitudes faster in practice.

Languages
I am proficient with C++ and Java.
I have some experience with C, Wolfram (implemented by Mathematica), and scripting languages (e.g. Bash and AWK).
I have also used Scheme, PHP, HTML and SQL.

Language skills

Portuguese and English
I can speak, read and write fluently in Portuguese and English.

Publications

Montalto, B., *Modeling an attacker with cryptanalytical capabilities*, MSc thesis (2008), IST Press, `http://wslc.math.ist.utl.pt/ftp/pub/MontaltoB/08-M-MScThesis.pdf`

Montalto, B. and C. Caleiro, *Modeling and reasoning about an attacker with cryptanalytical capabilities*, Electronic Notes in Theoretical Computer Science **253(3)**, Proc. 7th Workshop on Quantitative Aspects of Programming Languages (2009)

Conchinha, B., D. Basin and C. Caleiro, *Efficient algorithms for message deducibility and static equivalence*, Lecture Notes in Computer Science **6561**, Proc. 7th Int. Workshop on Formal Aspects of Security and Trust (FAST'10) (2010)

Conchinha, B., D. Basin and C. Caleiro, *Efficient algorithms for message de-*

ducibility and static equivalence, Technical Reports 680, ETH Zürich, Information Security Group D-INFK (2010), ftp://ftp.inf.ethz.ch/pub/publications/tech-reports/6xx/680.pdf

Conchinha, B., D. Basin and C. Caleiro, *FAST: An efficient decision procedure for deduction and static equivalence*, Leibniz International Proceedings in Informatics **10**, Proc. 22nd International Conference on Rewriting Techniques and Applications (RTA'11) (2011)

Conchinha, B., D. Basin and C. Caleiro, *Symbolic probabilistic analysis of off-line guessing*, Lecture Notes in Computer Science **8134**, Proc. 18th European Symposium on Research in Computer Security (ESORICS'13) (2013)